МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт ИРЭ
Кафедра ОРТ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(бакалаврская работа)

Направление 11.03.01 Радиотехника
(код и наименование)

Направленность (профиль)

Форма обучения **очная**
(очная/очно-заочная/заочная)

Тема: **Мультизонная система мониторинга климатических параметров**

Студент ЭР-14-17 Бутин А.А.
группа подпись фамилия и инициалы

Научный руководитель К.т.н. доцент Стрелков Н.О.
уч. степень должность подпись фамилия и инициалы

Консультант
уч. степень должность подпись фамилия и инициалы

Консультант
уч. степень должность подпись фамилия и инициалы

«Работа допущена к защите»

Зав. кафедрой к.т.н. доцент Шалимова Е.В.
уч. степень звание подпись фамилия и инициалы

Дата

Москва 2021

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт ИРЭ
Кафедра ОРТ

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
(бакалаврскую работу)

Направление 11.03.01 Радиотехника
(код и наименование)

Направленность (профиль)

Форма обучения **очная**
(очная/очно-заочная/заочная)

Тема: **Мультизонная система мониторинга климатических параметров**

Студент ЭР-14-17 _____ Бутин А.А.
группа подпись фамилия и инициалы

Научный руководитель к.т.н. доцент _____ Стрелков Н.О.
уч. степень должность подпись фамилия и инициалы

Консультант _____
уч. степень должность подпись фамилия и инициалы

Консультант _____
уч. степень должность подпись фамилия и инициалы

Зав. кафедрой к.т.н. доцент _____ Шалимова Е.В.
уч. степень звание подпись фамилия и инициалы

Место выполнения работы НИУ «МЭИ»

СОДЕРЖАНИЕ РАЗДЕЛОВ ЗАДАНИЯ И ИСХОДНЫЕ ДАННЫЕ

1. Подключение климатических датчиков DHT11, DHT22, BMP180 к платформе Arduino
2. Программирование платформы Arduino для работы с датчиками
3. Реализация сетевого обмена между платформой Arduino и внешними устройствами

ПЕРЕЧЕНЬ ГРАФИЧЕСКОГО МАТЕРИАЛА

Количество листов 10

Количество слайдов в презентации 10

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Петин В.А. Проекты с использованием контроллера Arduino. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2016. – 464 с.: ил.
2. Иго Т. Arduino, датчики и сети для связи устройств: Пер. с англ. — 2-е изд. — СПб.: БХВ-Петербург, 2015. —544 с.: ил.
3. Петин В. Создание умного дома на базе Arduino. ДМК Пресс. 2018.

Примечания:

1. Задание брошюруется вместе с выпускной работой после титульного листа (страницы задания имеют номера 2, 3).
2. Отзыв руководителя, рецензия(и), отчет о проверке на объем заимствований и согласие студента на размещение работы в открытом доступе вкладываются в конверт (файловую папку) под обложкой работы.

АННОТАЦИЯ

Выпускная квалификационная работа на тему: «Мульзонная система мониторинга климатических параметров».

Целью данной работы является рассмотрение модели системы Интернет вещей на плате Arduino UNO и плате WeMos D1 R1, макетное проектирование данной системы, а также программная реализация и выбор оптимальной системы вывода данных в сеть.

Во введении описывается актуальность представленной темы, формируются цели и задачи, которые выполняются в данной работе.

В первой главе описывается общая сводка определений и протоколов, используемых для передачи данных в сеть Интернет. Так же рассматривается структура систем Интернета вещей.

Во второй главе описываются платы, использующиеся для проектирования модели системы Интернета вещей, а также их технические данные.

В третьей главе описываются климатические датчики, выбранные для проектирования модели системы Интернета вещей, а так же их технические данные.

В четвертой главе проводится проектирование прототипа мультизонной системы мониторинга климатических параметров, отправление данных с датчиков в сеть и оценка работоспособности системы.

Работа содержит 63 страницы, 49 рисунков, 5 таблиц, 10 источников использованной литературы, а также 3 приложения.

ВВЕДЕНИЕ

Технологический прогресс никогда не стоит на месте. Каждый год в различных сферах деятельности внедряются новые технические решения для упрощения жизни человека. Когда-то таким технологическим новшеством была сеть Интернет – всемирная система объединенных компьютерных сетей. Интернет дал новый рывок в использовании и передачи информации между людьми. Технический прогресс распространился так же на Интернет, поэтому даже на сегодняшний день развитие этой сети является актуальной темой.

Одним из новых ветвей развития Интернета является концепт Интернета вещей. Точной временной границы начала использования интернета вещей нет. Впервые термин «Интернет вещей» был использован еще в 1999 году, когда сотрудник Procter & Gamble Кэвин Эштон предложил использовать для оптимизации логистики корпорации радиочастотные метки (radio-frequency identification – RFID) [1]. Но на тот момент интернет-индустрия только начинала свое развитие. После технологии RFID активно начали развиваться коммуникационные сети, интегрируемые в Интернет, такие как беспроводные сенсорные сети WSN (Wireless Sensor Network), коммуникации малого радиуса действия NFC (Near Field Communication) и межмашинные коммуникации M2M (Machine-to-Machine). Такие виды коммуникаций позволили обеспечить простую связь между техническими устройствами для обмена данными. С этого момента начинается активное развитие Интернета вещей, которое продолжается и сегодня. В 2008-2009 годах по расчетам консалтингового подразделения Cisco IBSG количество подключенных к интернету предметов превысило население Земли, а на 2015 год количество подключенных устройств уже составляло 25 миллиардов и с каждым годом это число пропорционально увеличивается, что говорит об актуальности развития Интернета вещей [2].

Интернет вещей (англ. Internet of Things, IoT) – это это глобальная сеть компьютеров, датчиков (сенсоров) и исполнительных устройств

(актуаторов), связывающихся между собой с использованием интернет протокола IP (Internet Protocol) [2]. Интернет вещи посредством обмена информацией между различными датчиками и сенсорами должны полностью автоматизировать процессы управления группами устройств. Другими словами, это не просто множество сенсоров, датчиков и приборов, объединённых в одну сеть. Главная особенность данной сети — это однозначное идентифицирование каждого объекта, а также более тесная интеграция реального и «виртуального» миров, где взаимодействие происходит между людьми и устройствами.

Целью дипломной работы является изучение основных аспектов Интернета вещей с теоретической и практической стороны. Изучение практической стороны будет вестись через моделирование простейшей системы интернета вещей, а именно, мультизонной системы мониторинга климатических параметров (далее «модель»). На заданной модели будет рассмотрена архитектура Интернета вещей и способы передачи/хранения данных. Так же будет произведено сравнение между несколькими способами передачи/ хранения данных и сделаны выводы по каждому их способов.

В роли центрального процессора модели будет использованы платы Arduino Uno и Arduino-совместимая плата WeMos D1 R1. Данные платы будут считывать данные с датчиков, переводить в удобный для человека вид и передавать в сеть. Передача данных в сеть будет осуществляться двумя способами: с помощью платы-расширения Ethernet Shield и непосредственным подключением к Wi-Fi роутеру через Ethernet-кабель, и с помощью беспроводного подключения к Wi-Fi роутеру с использованием возможностей платы WeMos D1 R1.

1. Интернет вещей

Идея Интернета вещей сама по себе очень проста. При наличии необходимых каналов связи между устройствами можно не только отслеживать объекты и их параметры в пространстве и во времени, но и управлять ими, а также включать информацию о них в общую «умную планету» [2]. Как говорилось ранее, Интернет вещей – это глобальная сеть датчиков и исполнительных устройств, связанных между собой.Интернет вещей основывается на трех базовых принципах: во-первых, повсеместно распространенную коммуникационную инфраструктуру, во-вторых, глобальную идентификацию каждого объекта и, в-третьих, возможность каждого объекта отправлять и получать данные посредством персональной сети или сети Интернет, к которой он подключен [2]. Главными отличиями IoT от Интернета является фокусировка внимания на «вещах» (на физических объектах), а не на человеке; большее количество соединенных в сеть объектов; меньшие, по сравнению с Интернетом, скорости передачи данных.

Для дальнейшего рассмотрения Интернета вещей стоит сначала разобраться с архитектурой и стандартами данной технологии.

1.1. Структура Интернета вещей

Любая сеть Интернета вещей состоит из четырех основных уровней: уровень элементов, сеть, уровень обслуживания и прикладной уровень (см. рисунок 1.1).

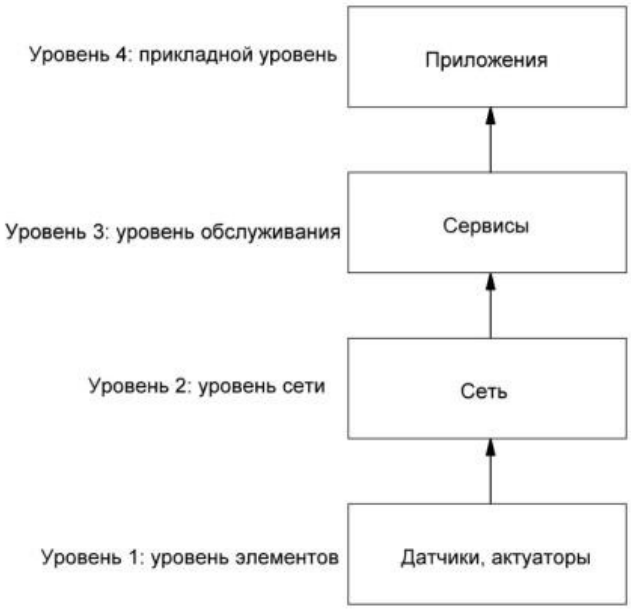

Рисунок 1.1 – Структура Интернета вещей

Уровень элементов – самый нижний уровень архитектуры IoT, состоящий из датчиков и актуаторов вместе с объектами обработки данных. Датчики обеспечивают связь физического объекта с цифровым пространством, занимаются сбором зачастую в реальном времени. Развитие технологий позволило миниатюризировать размеры датчиков, что привело к их интеграции непосредственно в физические объекты для более точного сбора данных. Существуют различные типы сенсоров для соответствующих целей, например, для измерения температуры, давления, скорости движения, местоположения и др. Сенсоры могут иметь небольшую память, давая возможность записывать некоторое количество результатов измерений [2]. Актуаторы же служат для воздействия на физический объект (изменение температуры, освещение, непосредственное физическое воздействие). Большинство сенсоров требуют соединения с агрегатором сенсоров (шлюзом), которые могут реализоваться быть реализованы с использованием локальной вычислительной сети (LAN, Local Area Network), таких как Ethernet и Wi-Fi или персональной сети (PAN, Personal Area Network). Для сенсоров, которые не требуют подключения к агрегатору, их связь с серверами/приложениями может предоставляться с использованием

глобальных беспроводных сетей WAN, таких как GSM, GPRS и LTE [2]. Зачастую данные со всех датчиков внутри одной системы IoT поступают на центральный процессор (например, в «умном доме») для обработки и передачи на следующий уровень.

Второй уровень IoT – это уровень сетей. Данный уровень обеспечивает передачу данных между датчиками и уровнем приложений. Здесь в качестве сети подразумевается не только Интернет, но и мобильные сети или спутниковая связь. Большой объем данных, создаваемых на первом уровне IoT многочисленными миниатюрными сенсорами, требует надежной и высокопроизводительной проводной или беспроводной сетевой инфраструктуры в качестве транспортной среды [2]. Сетевой уровень – это составная модель, которая создается путем объединения различных сетей в единую сетевую платформу. С помощью такой модели сети одни и те же ресурсы могут использовать сразу несколько пользователей.

Передача данных по сети осуществляется с помощью специализированных протоколов. Главным требованием для них является направленная передача данных. Протоколы передачи для сети IoT будут более подробно рассмотрены в следующих главах.

Третий уровень – это уровень обслуживания. На данном уровне осуществляется конечная обработка данных и их хранение на специализированных сервисах. Сервисный уровень содержит набор информационных услуг, призванных автоматизировать технологические операции в IoT: поддержка операционной деятельности, различной аналитической обработки информации, хранения данных, обеспечения информационной безопасности и др.

Прикладной уровень используется в основном только человеком. На этом уровне собранная информация с датчиков визуализируется в удобный для восприятия человеком вид для непосредственного анализа, а так же присутствует удобный интерфейс неавтоматического управления актуаторами. На уровне приложений человек может взаимодействовать с

физическим объектом, подключенным к Интернету вещей, даже не находясь рядом с ним.

1.2. Передача данных внутри Интернета вещей

Любая IoT сеть всегда использует несколько протоколов для передачи данных. Это вытекает из ее уровневой структуры: используется конкретный протокол, который будет удобно использовать для задач каждого уровня. Например, для уровня элементов, расположенных на небольшом расстоянии, удобно использовать Bluetooth или Ethernet, для обеспечения быстрого сбора данных, а для транспортировки данных на сетевом уровне удобно использовать протокол TCP/IP, который обеспечит защищенность и целостность данных.

Рассмотрим более подробно протоколы передачи данных на различных уровнях IoT:

Физический уровень. На физическом уровне происходит коммутация между устройствами в определенной среде. Для такого уровня характерна передача данных битами, а не пакетами данных. На этом уровне главными атрибутами передачи данных является синхронизация между устройствами, малые помехи и высокая скорость передачи данных.

Для физического уровня самыми распространенными стандартами передачи данных являются 100BASE-T (или 1000BASE-T) и набор стандартов IEEE 802.3 (определяет канальный и физический уровни в сети Ethernet). Так же в настоящее время широко используют другие способы передачи данных на физическом уровне, такие как: BLE (Bluetooth с низким энергопотреблением), LTE (стандарт беспроводной широкополосной мобильной связи), NFC (связь ближнего действия), PLC (связь по линиям электросети), стандарт Wi-Fi/802.11 (стандарт передачи данных для жилых домов и офисов), Z-Wave (многосвязная сеть, использующая

низкоэнергетические радиоволны для связи между устройствами) и Zigbee (спецификация на основе стандарта IEEE 802.15.4 для набора протоколов связи высокого уровня, используемых для создания персональных сетей с небольшими цифровыми радиостанциями с низким энергопотреблением).

Канальный уровень. Канальный уровень предназначен для передачи данных по узлам одной локальной сети. На канальном уровне данные комплектуются в кадры (frame) и передаются между устройствами одного сетевого сегмента. Каждый кадр состоит из преамбулы (биты, определяющие начало кадра), начала кадра (байт, с которого начинается кадр), заголовка (содержит данные об отправителе и получателе), длины кадра и непосредственно данных. Основными задачами канального уровня являются осуществление доступа к среде передачи, аппаратная адресация, обеспечение целостности передаваемых данных, адресация протоколов для верхних уровней, а так же может использоваться для обнаружения и исправления ошибок, возникающих на физическом уровне.

Для канального уровня используют стандарт IEEE 802.15.4, разделенный на два подуровня: MAC (Media Access Control), регулирующий доступ к разделяемой физической среде, и LLC (Local Link Control), обеспечивающий обслуживание сетевого уровня.

Уровень сети. Сетевой уровень обеспечивает объединение участков сети и маршрутизацию данных. Каждое устройство внутри сети должно иметь уникальный сетевой адрес. Это необходимое требование для корректной работы маршрутизаторов (роутеров), обеспечивающих распределение пакетов данных между разными сегментами сети.

Самыми распространенными протоколами сетевого уровня являются протоколы IP (Internet Protocol), IPv4 или IPv6. Отличием IPv6 от IPv4 является адресация большего количества узлов, что говорит о возможности увеличения числа подключенных к сети устройств.

Уровень транспортировки. Транспортный уровень предназначен для обеспечения передачи данных между уровнями и их защиты. Протоколы транспортного уровня решают проблему негарантированной доставки сообщений, а также определяют правильную последовательность прихода данных к адресату.

Для транспортного уровня существует множество различных протоколов, выполняющих различные функции, например, функция передачи данных без подтверждения приема или функция доставки нескольких последовательных пакетов данных. Но основными протоколами для данного уровня являются протоколы TCP (Transmission Control Protocol) и UDP (User Datagram Protocol).

TCP протокол используется в тех случаях, когда важна целостность передаваемых данных. Перед передачей данных осуществляется соединение между получателем и адресатом, далее происходит запрос на принятие данных. После подтверждения запроса данные передаются пакетами и принятие каждого пакета сопровождается уведомлением со стороны получателя. В случае потери данных пакет дублируется, что и гарантирует их целостность. Протокол TCP обеспечивает надежную передачу данных, но из-за подтверждения каждого пакета данных и их дублирования передача данных происходит довольно медленно.

Преимуществом UDP протокола является высокая скорость передачи данных. В отличие от TCP протокола, в UDP протоколе нет уведомлений о принятии данных запроса на передачу данных. Таким образом, данные могут идти не по порядку, дублироваться или теряться при передаче. UDP протокол подразумевает, что исправления ошибок передачи либо не нужны, либо происходят на уровне приложений. Данный протокол удобен в случаях, когда скорость передачи данных стоит в большем приоритете, чем их целостность (например, трансляция голоса или видео).

Уровень приложений. Уровень приложений выступает посредником между пользователем и устройством. Уровень позволяет пользователю через приложения получить доступ к сетевым службам, таким, как обработчик запросов к базам данных или доступ к файлам. Так же уровень отвечает за передачу служебной информации, формирует информацию об ошибках и запросы к другим уровням сети.

Для уровня приложений существует множество протоколов передачи данных для различных целей и задач. Одними из основных протоколов являются протоколы HTTP, HTTPS, MQTT. MQTT – это протокол обмена сообщениями с неинтенсивным обменом данных. В основном этот протокол используется для устройств с низкой пропускной способностью. Протокол HTTP – это протокол для передачи гипертекстовых документов в формате HTML страниц. Основой HTTP протокола является технология «клиент-сервер», то есть существует потребитель (клиент), который посылает запрос, и поставщик (сервер), который обрабатывает запросы потребителя и отправляет ему ответ. Протокол HTTPS работает по такому же принципу. Существенным отличием от протокола HTTP является то, что протокол HTTPS обеспечивает защиту передаваемых данных с помощью кодировки информации.

2. Описание аппаратной части

В данном разделе будут подробно рассмотрены платы, используемые в модели. Важным параметром, на который стоит обратить внимание, является напряжение питания плат и напряжение, выдаваемое на выходах плат. Выделяются именно эти параметры для того, что бы подобрать наилучший способ питания для плат и так же, что бы датчики, подключенные к плате, не конфликтовали по напряжению с самой платой. В случае неправильно поданного напряжения датчик может искажать посылаемые данные или вовсе выйти из строя.

2.1. Микроконтроллер Arduino

Для обработки данных с датчиков в данной работе будет использована плата Arduino Uno на базе 8-ми разрядного микроконтроллера AVR ATmega328 (см. рисунок 2.1). Преимуществом данного микроконтроллера является его дешевизна и доступность. Так же к достоинствам можно отнести низкое энергопотребление, что позволяет плате работать на автономном питании. Дальнейшее описание платы и ее характеристики взяты из официальной документации к плате [3].

Рисунок 2.1 – Плата Arduino Uno

Таблица 2.1 – Технические характеристики Arduino Uno

Микроконтроллер	ATmega328
Рабочее напряжение	5В
Напряжение питания (рекомендуемое)	7-12В
Напряжение питания (предельное)	6-20В
Цифровые входы/выходы	14 (из них 6 могут использоваться в качестве ШИМ-выходов)
Аналоговые входы	6
Максимальный ток одного вывода	40 мА

Продолжение таблицы 2.1

Максимальный выходной ток вывода 3.3V	50 мА
Flash-память	32 КБ (ATmega328) из которых 0.5 КБ используются загрузчиком
SRAM	2 КБ (ATmega328)
EEPROM	1 КБ (ATmega328)
Тактовая частота	16 МГц

Описание контактов:

На плате имеются 14 цифровых входов-выходов (0-13) и 6 аналоговых (A0-A5), которые используются для подключения периферийных устройств. К каждому контакту программно может быть подключен встроенный подтягивающий резистор на 20-50 кОм. Так же для удобства некоторые контакты объединяют в себе несколько функций (например, контакты A4 и A5, которые являются аналоговыми контактами, так же используются как контакты SDA и SCL шины I2C, см. таблицу 2.2).

На плате присутствуют дополнительные контакты: AREF – выдает опорное напряжение для встроенного АЦП, RESET – при подаче логического нуля на вход устройство будет перезагружено.

Таблица 2.2 – Описание контактов

Контакт Arduino	Контакт на плате	Специальное назначение
Цифровой контакт 0	0	RX
Цифровой контакт 1	1	TX
Цифровой контакт 2	2	Вход для прерываний
Цифровой контакт 3	3	Вход для прерываний
Цифровой контакт 4	4	
Цифровой контакт 5	5	
Цифровой контакт 6	6	
Цифровой контакт 7	7	
Цифровой контакт 8	8	
Цифровой контакт 9	9	
Цифровой контакт 10	10	SPI (SS)
Цифровой контакт 11	11	SPI (MOSI)
Цифровой контакт 12	12	SPI (MISO)
Цифровой контакт 13	13	SPI (SCK)
Аналоговый контакт A0	A0	
Аналоговый контакт A1	A1	
Аналоговый контакт A2	A2	
Аналоговый контакт A3	A3	
Аналоговый контакт A4	A4	I2C (SCA)
Аналоговый контакт A5	A5	I2C (SCL)

На плате расположены контакты питания. На контакт питания 5V плата подает напряжение 5 вольт для питания внешних устройств. Соответственно на контакт 3.3V плата подает 3.3 вольта за счет встроенного стабилизатора

напряжения. Три контакта GND соединены между собой и используются для подключения устройств к земле. VIN – контакт для подачи внешнего напряжения. Контакт IREF используется для информирования внешних устройств о рабочем напряжении платы.

Стоит отметить важный момент с питанием внешних устройств от платы: питание внешних устройств возможно от контактов с вольтажом в 5 и 3.3 вольта. Эти параметры будут учитываться при подборе датчиков для размещения на плате.

Питание платы:

Как видно из таблицы 2.1 рабочее напряжение платы – 5 вольт. Но из-за наличия встроенного стабилизатора напряжения плату можно питать от различных источников напряжением от 7 до 12 вольт. На плате размещены два разъема для подключения питания: USB-B и DC 2.1 мм. Так же для питания можно использовать контакт питания VIN. Данные способы подключения дают свободу в выборе источника питания: USB-порт компьютера, блок регулируемого напряжения, батарейка и.т.д. То есть плату можно удобно использовать как и на рабочем столе рядом с персональным компьютером, так и в местах без доступа к питанию с использованием автономного питания через вход DC.

2.2. Плата расширения Ethernet Shield

Используя Arduino Uno в качестве центрального процессора, который будет заниматься сбором и обработкой данных с датчиков, необходимо предусмотреть соединение платы с Интернетом. Для подключения Arduino Uno к сети отлично подойдет плата Ethernet Shield W5100 (см. рисунок 2.2). Данная плата позволит производить обмен данными как внутри локальной сети, так и в сети Интернет. Одной из главных особенностей Ethernet Shield является то, что это плата расширения для Arduino Uno: соединение двух плат не составит никаких сложностей, а библиотека для работы с Ethernet

Shield уже есть в среде программирования с подробно расписанными примерами. Сейчас важным вопросом является обмен данными между платами, какие контакты при этом задействуются, а так же обмен данных между Ethernet Shield и сетью. Дальнейшее описание платы и ее характеристики взяты из официальной документации к плате [4].

Наиболее популярные Ethernet модули для Arduino сегодня выпускаются на основе микросхемы WIZnet W5100, которая способна поддерживать обмен данными с постоянной скоростью в 100 Мбит/сек.

Подключение платы расширения к сети происходит через соединение RJ-45. Плата обладает встроенным слотом SD/MicroSD , который используется для хранения файлов, используемых для подключения и передачи по локальной сети. Такой слот совместим со всеми платами Arduino. Работать с данными на карте можно с помощью стандартной библиотеки SD Library.

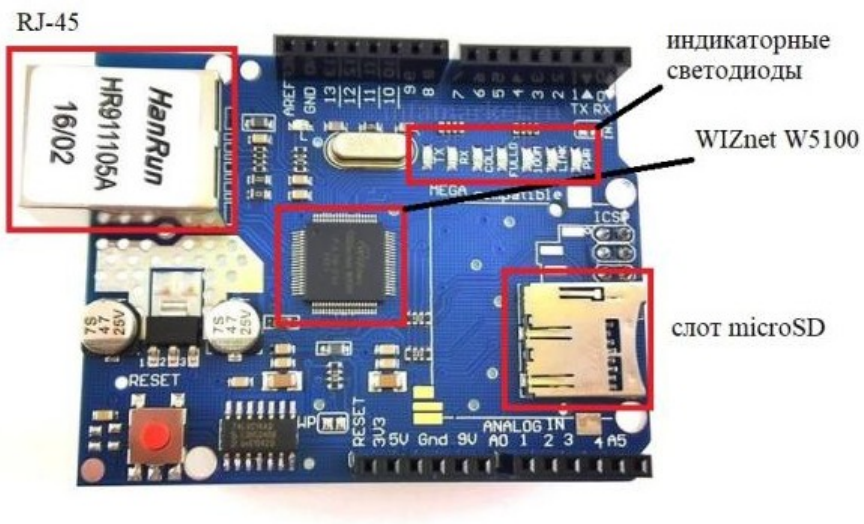

Рисунок 2.2 – Плата расширения Ethernet Shield W5100

Ключевые характеристики модулей на базе W5100: рабочее напряжение – 5 Вольт, подходит питание с платы Arduino; внутренний буфер 16 Кб; скорость соединения достигает значения в 10-100 Мбит/сек; связь с платой Arduino осуществляется посредством шины SPI, соединение

двух плат представлено на рисунке 2.3; W5100 поддерживает протоколы передачи данных TCP и UDP.

Контакты платы Ethernet Shield дублируют контакты платы Arduino Uno, соответственно подключение сторонних устройств к Arduino Uno осуществляется через контакты Ethernet Shield.

В ранее рассмотренном пункте 2.1 по таблице 2.2 видно, что для шины SPI используются цифровые контакты 10, 11, 12, 13. Так же при использовании слота microSD занимается 4й контакт Arduino Uno для обмена данными с SD-картой. Соответственно при подключении Ethernet Shield к Arduino Uno нельзя использовать контакты 4, 10, 11, 12, 13 для подключения сторонних устройств.

Рисунок 2.3 – соединение платы расширения Ethernet Shield W5100 и Arduino Uno

Так же на некоторых платах предусмотрен POE модуль. При использовании этого модуля питание на плату подается от витой пары. При этом Ethernet Shield будет соответствовать следующим спецификациям: совместим со стандартом IEEE3af; имеет низкие пульсации на выходе;

защищает от явлений перегрузки и короткого замыкания; эффективно преобразует напряжение.

Есть аппаратная поддержка протокола PPPoE (Point-to-point over Ethernet) с PAP/CHAP протоколами аутентификации, что позволяет осуществлять удаленное подключение встраиваемого устройства к провайдеру через простой и дешевый DSL-модем, работающий в мостовом (bridge) режиме и не имеющий собственной аппаратной поддержки PPPoE [5].

На плате предусмотрены индикаторные светодиоды, назначения светодиодов представлено в таблице 2.3.

Таблица 2.3 – Назначение светодиодов на плате Ethernet Shield W5100

Обозначение на плате	Назначение светодиода
PWR	Указывает на наличие питания на плате
LINK	Светится при наличии сети, мигает при передаче/приеме данных
FULLD	Указывает на сетевое полнодуплексное соединение
100M	Указывает на сетевое соединение со скоростью 10мбит/сек
RX	Указывает на прием экраном данных
TX	Указывает на отправку данных экраном
COLL	Мигает при обнаружении сетевых конфликтов
Светодиод разъема RJ-45 (желтый)	Мигает при поступлении данных
Светодиод разъема RJ-45 (зеленый)	Горит при подключенном Ethernet-шнуре

2.3. Микроконтроллер ESP8266

WeMos – это китайская компания, которая производит Arduino-совместимые микроконтроллеры с широким спектром возможностей. Главным достоинством плат WeMos является расширение функционала для Arduino-совместимых устройств, а так же их ценовая доступность. Блочная схема микроконтроллера WeMos дает возможность превратить каждый кристалл на плате устройства с записанной программой в полноценный функциональный модуль с низкой производственной себестоимостью. Интеграция таких модулей позволяет создавать на плате универсальное контрольное устройство с любой требуемой схемой управления. Функциональная гибкость микроконтроллеров WeMos заключается в возможности в любой момент внести ряд изменений в программный алгоритм без изменений в архитектуре микросхемы на плате. Под новую задачу достаточно будет загрузки новой прошивки.

В дипломной работе будет использоваться плата WeMos D1 R1, представленная на рисунке 2.4.

Основу данной платы составляет ESP8266 – микроконтроллер с интерфейсом Wi-Fi, имеющий следующие технические параметры: поддерживает Wi-Fi протоколы 802.11 b/g/n с WEP, WPA, WPA2; обладает 14 портами ввода и вывода, SPI, I2C, UART, 10-бит АЦП; поддерживает внешнюю память до 16 МБ; необходимое питание от 2,2 до 3,6 В, потребляемый ток до 300 мА в зависимости от выбранного режима [6]. Важной особенностью является отсутствие пользовательской энергонезависимой памяти на кристалле. Программа выполняется от внешней SPI ПЗУ при помощи динамической загрузки необходимых элементов программы. Доступ к внутренней периферии можно получить не из документации, а из API набора библиотек. Производителем указывается приблизительное количество ОЗУ – 50 кБ. На модуле имеется Wi-Fi антенна для покрытия Wi-Fi сетью.

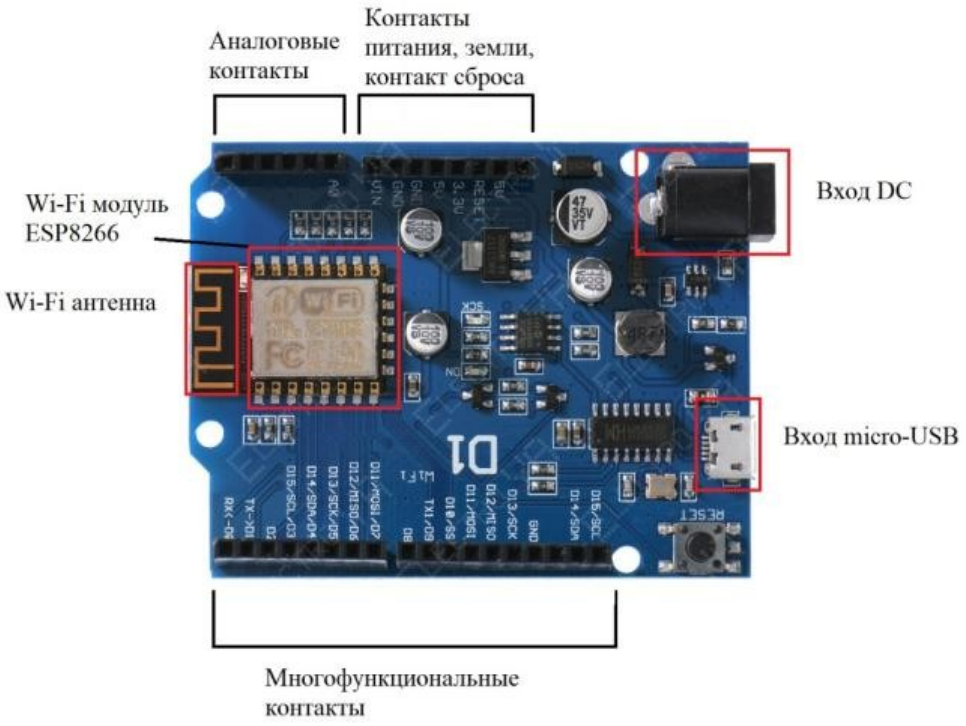

Рисунок 2.4 – плата WeMos D1 R1

Так как плата Arduino-совместимая, то распиновка платы схожа с распиновкой Arduino Uno, но все же имеет ряд отличий. Самое главное отличие: увеличенное количество интерфейсов передачи данных на плате. На WeMos D1 R1 присутствует 2 группы контактов под интерфейс I2C и 2 еще 2 группы под интерфейс SPI (см. таблицу 2.4). Каждый цифровой контакт платы является портом общего назначения и служит для низкоуровневого обмена цифровыми сигналами с внешними устройствами (контакт GPIO – general-purpose I/O port). Так же на WeMos D1 R1 присутствует только один аналоговый контакт.

Таблица 2.4 – Описание контактов

Контакт WeMos D1 R1	Контакт на плате	Специальное назначение
Цифровой контакт 0	D0/GPIO3	RX
Цифровой контакт 1	D1/GPIO1	TX

Продолжение таблицы 2.4

Цифровой контакт 2	D2/GPIO16	
Цифровой контакт 3	D3/GPIO5	I2C(SCL)
Цифровой контакт 4	D4/GPIO4	I2C(SDA)
Цифровой контакт 5	D5/GPIO14	SPI (SCK)
Цифровой контакт 6	D6/GPIO12	SPI (MISO)
Цифровой контакт 7	D7/GPIO13	SPI (MOSI)
Цифровой контакт 8	D8/GPIO0	
Цифровой контакт 9	D9/GPIO2	
Цифровой контакт 10	D10/GPIO15	SPI (SS)/ TX
Цифровой контакт 11	D11/GPIO13	SPI (MOSI)/RX
Цифровой контакт 12	D12/GPIO12	SPI (MISO)
Цифровой контакт 13	D13/GPIO14	SPI (SCK)
Цифровой контакт 14	D14/GPIO4	I2C(SDA)
Цифровой контакт 15	D15/GPIO5	I2C(SCL)
Аналоговый контакт A0	A0	

Так же как и на Arduino Uno на плате расположены контакты питания. На контакт питания 5V плата подает напряжение 5 вольт для питания внешних устройств. Соответственно на контакт 3.3V плата подает 3.3 вольта за счет встроенного стабилизатора напряжения. Три контакта GND соединены между собой и используются для подключения устройств к земле. VIN – контакт для подачи внешнего напряжения.

3. Климатические датчики

3.1. Описание климатических датчиков

Датчики и исполнительные устройства являются основой любой системы Интернета вещей. Независимо от индивидуальных требований и перечня задач, которые должна решать система в целом, именно датчики обеспечивают необходимую степень автоматизации и передают другим устройствам сигнал о необходимости включения или выключения в определенный момент [7].

На сегодняшний день на рынке радиомодулей можно найти разнообразные климатические датчики от различных производителей. Это разнообразие объясняется требованиями заказчиков. Зачастую датчики используются для пользовательских проектов, на которые наложены определенные условия в виде точности измерения и условий работы (например, мониторинг температуры в помещении, или на улице, кратковременное измерение или постоянный мониторинг). Наиболее распространенными климатическими датчиками являются: датчик влажности и температуры DHT11(DHT22), датчик атмосферного давления BMP180(BMP280).

Все характеристики датчиков, описываемых ниже, взяты с официальных даташитов [8] [9] [10].

Датчик влажности и температуры DHT11(DHT22):

DHT11 (Digital Humidity Temperature) (см. рисунок 3.1) – недорогой цифровой датчик температуры и влажности. Он использует емкостной датчик влажности и терморезистор для измерения температуры окружающего воздуха, данные выдает в цифровой форме по шине типа 1-wire. В использовании он довольно прост, но требует точного определения

длительности временных сигналов, чтобы декодировать данные. Единственный недостаток это возможность получения данных не чаще 1 раза в две секунды.

Рисунок 3.1 – Датчик влажности и температуры DHT11

Для преобразования данных внутри датчика используется 8-битный микроконтроллер, В процессе производства датчики калибруются и калибровочная константа записывается вместе с программой в память микроконтроллера. Однопроводный последовательный интерфейс дает возможность быстрой интеграции в устройство [7].

DHT11 определяет влажность в диапазоне от 20 до 80 % с точностью 5 % и температуру в диапазоне от 0 до 50 °C с точностью 2 %. Частота опроса датчика не более одного раза в секунду.

Между выводом питания и выводом данных необходимо разместить подтягивающий резистор. Рекомендуемый номинал 10 кОм , если расстояние от датчика к плате Arduino небольшое. Для расстояния больше 20 метров рекомендуется резистор номиналом 5,1 кОм. Так же необходим сглаживающий конденсатор между выводом питания и землей. Но для

удобства подключения предусмотрен модульный вариант датчика, в который уже включен резистор и конденсатор (см. рисунок 3.2).

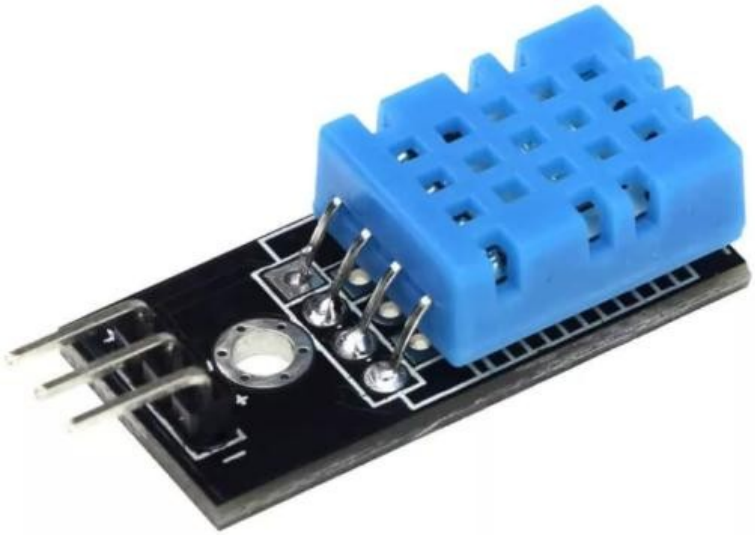

Рисунок 3.2 – Модуль датчика DHT11

DHT22 является улучшенным аналогом DHT11 с интервалами определения влажности в диапазоне от 0 до 100 % с точностью 2-5 % и температуры в диапазоне от -40 до 125 °C с точностью 0.5 °C с частотой опроса не более раза в 2 секунды. В остальном он повторяет датчик DHT11. Так же имеется безмодульный (см. рисунок 3.3) и модульный (см. рисунок 3.4) варианты.

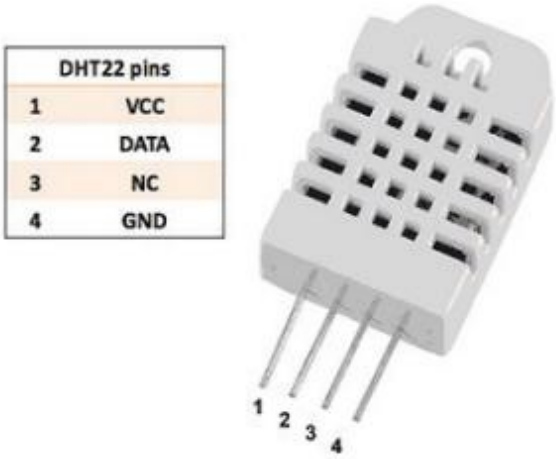

Рисунок 3.3 – Датчик DHT22 с распиновкой

Рисунок 3.4 – Модуль датчика DHT22

Датчик атмосферного давления BMP180:

Датчик BMP180 (см. рисунок 3.5) (Digital Pressure Sensor) — является недорогим и простым в использовании сенсорным датчиком, позволяющий измерить атмосферное давления и температуру окружающей среды.

Выпускается только в модульном варианте. На модуле расположен сам сенсорный датчик BMP180 фирмы Bosch. Так как датчик BMP 180, работает от 3.3В (а почти все платы Arduino работают на 5В), на плате предусмотрен стабилизатор напряжения XC6206P332MR в корпусе SOT-23, который выдает на выходе напряжение в 3.3В, рядом установлена обвязка стабилизатора, состоящая из двух керамических конденсаторов на 1 мкФ. Подключение осуществляется по интерфейсу I2C, линии SCL и SDA выведены на группу контактов на другой стороне модуля, туда же выведено и питание. Последние два резистора на 4.7 кОм, необходимы подтяжки линии SCL и SDA к питанию.

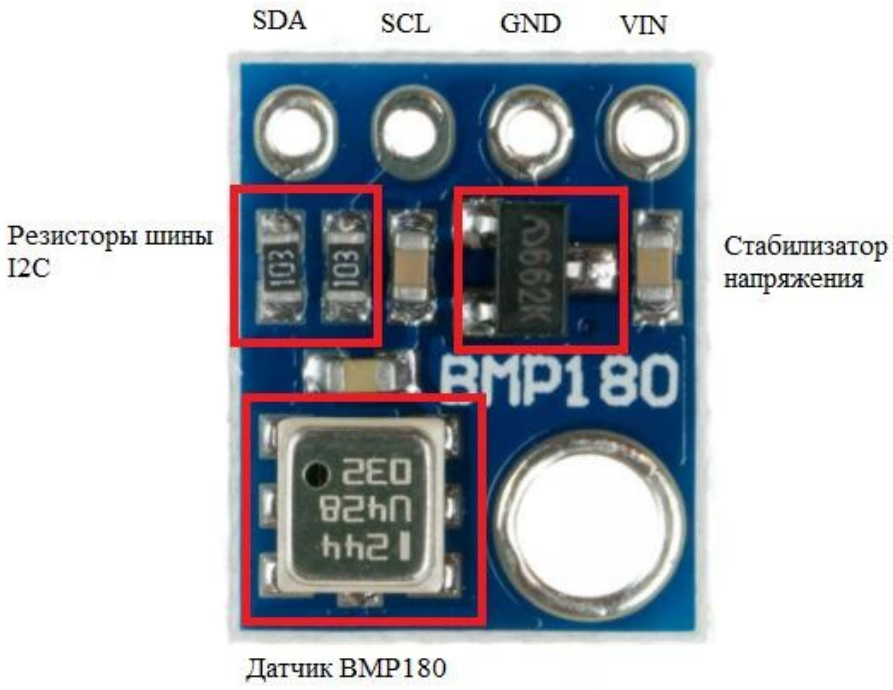

Рисунок 3.5 – Модуль датчика BMP180 с распиновкой

Характеристики датчика:

- Рабочая температура -40 … 80 °C предельно допустимые значения

- Рабочее давление 0 … 10 МПа предельно допустимые значения

- Диапазон давления: 300 … 1100 гПа разрешение 0,06 гПа точность ±0,12 гПа (на пределах ±1 гПа)

- Диапазон температуры: 0 … 65 °C разрешение 0,1°C точность ±0.5 °C (на пределах ±2 °C)

- Время преобразований 3 … 51 мс зависит от режима точности

3.2. Подключение датчиков к микроконтроллеру Arduino

Подключение DHT11/DHT22:

Для проекта были выбраны модули датчиков, поэтому нет необходимости использовать подтягивающий резистор и сглаживающий конденсатор. Модули подключаются по однопроводным последовательным

интерфейсом непосредственно к пину приема данных (в проекте используется пин 3 для DHT11 и пин 5 для DHT22). Схемы подключения модулей представлены на рисунках 3.6 и 3.7.

Схема подключения к плате WeMos принципиально ничем е отличается, так как на плате находятся цифровые контакты (D1-D15) как и на плате Arduino (см. рисунки 3.8, 3.9).

Описание передачи данных по однопроводному интерфейсу:

Микроконтроллер (МК) инициирует передачу данных путем отправки сигнала «Старт». Для этой цели данный вывод МК должен быть сконфигурирован на выход. МК сначала переводит линию в состояние низкого уровня, по крайней мере на 18 мсек, а затем переводит ее в высокое состояние на 20-40 мксек, прежде чем МК освобождает линию. Далее, датчик реагирует на сигнал «Старт» от МК, отправив низкий уровень сигнала длительностью 80 мксек, а затем высокий уровень с такой же продолжительностью. После обнаружения сигнала отклика от датчика, МК должен быть готов к приему данных от датчика. Датчик посылает 40 бит (5 байт) данных, непрерывно в линию передачи данных. Следует отметить, что во время передачи байта, датчик посылает старший значащий бит первым [9] [10].

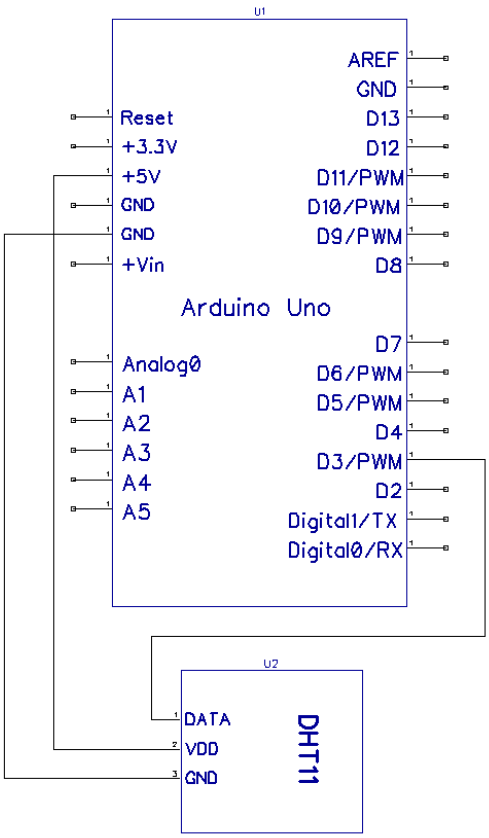

Рисунок 3.6 – Схема подключения DHT11 к Arduino Uno

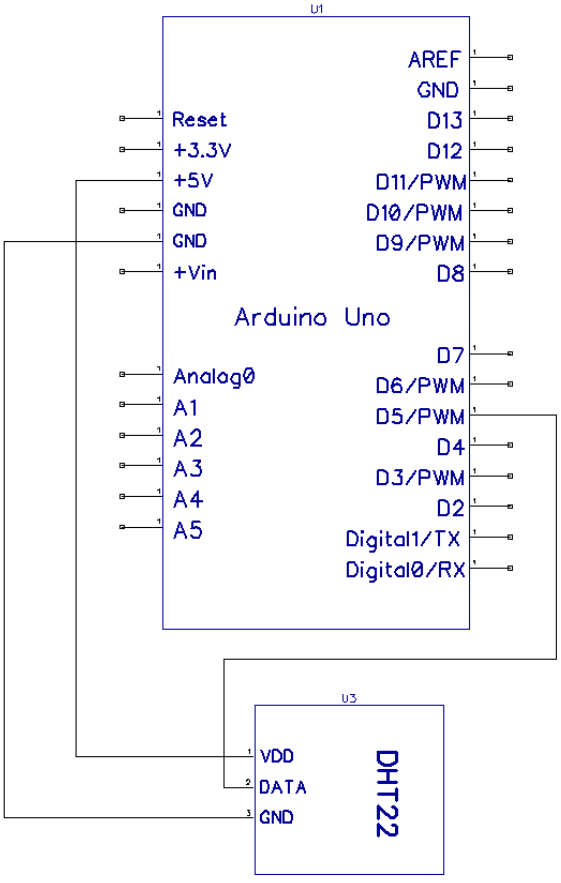

Рисунок 3.7 – Схема подключения DHT22 к Arduino Uno

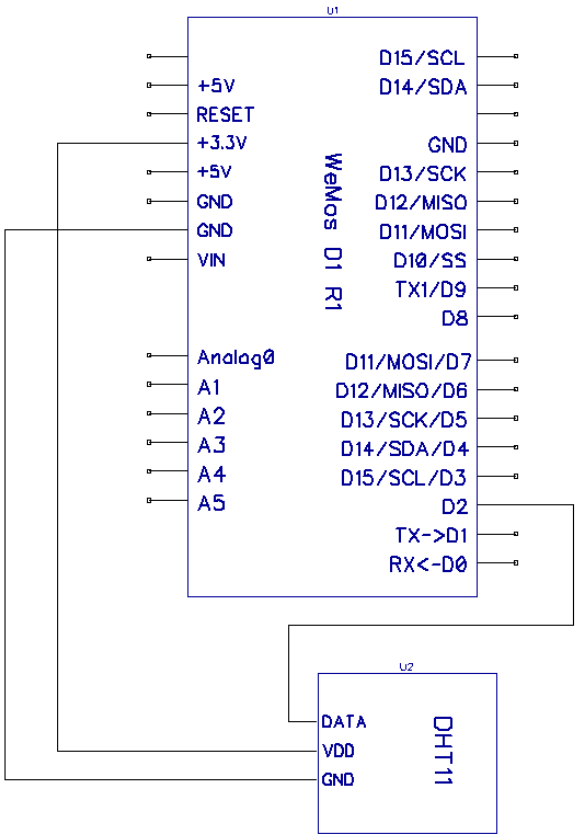

Рисунок 3.8 – Схема подключения DHT11 к WeMos D1 R1

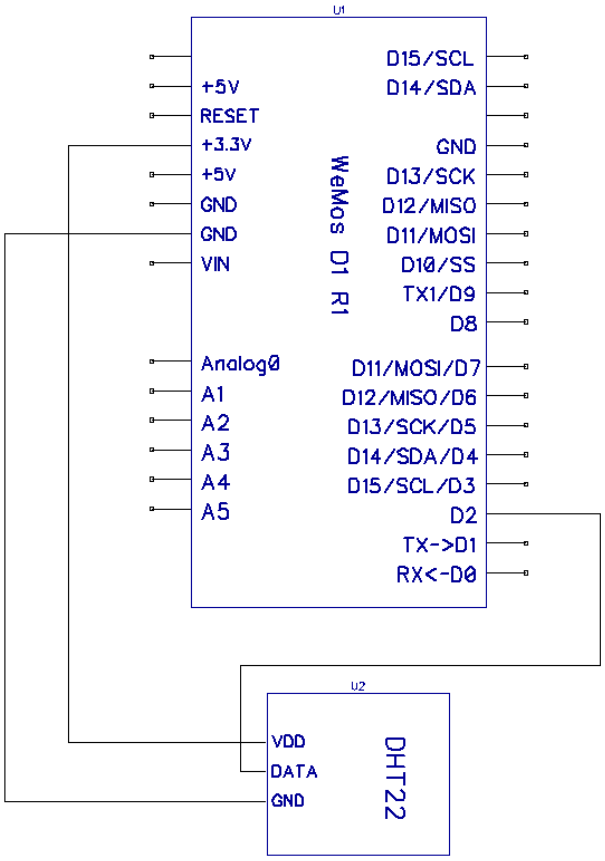

Рисунок 3.9 – Схема подключения DHT22 к WeMos D1 R1

Подключение BMP180:

Модуль датчика BMP180 подключается к Arduino по последовательному интерфейсу I2C. Для подключения датчика используется следующая распиновка: SDA (шина данных) подключается к контакту A4 (контакт данных для шины I2C), SCL (шина тактирования) подключается к контакту A5 (контакт тактирования шины I2C). Подключение датчика к Arduino представлено на рисунке 3.10.

Для подключения датчика к плате WeMos D1 R1 используется такой же I2C интерфейс, как и для платы Arduino Uno. Однако на плате предусмотрены контакты сразу для двух I2C интерфейсов (D15, D14 и D3, D4). Для подключения используем D15, D14 (SCL, SDA соответственно). Подключение датчика к WeMos представлено на рисунке 3.11.

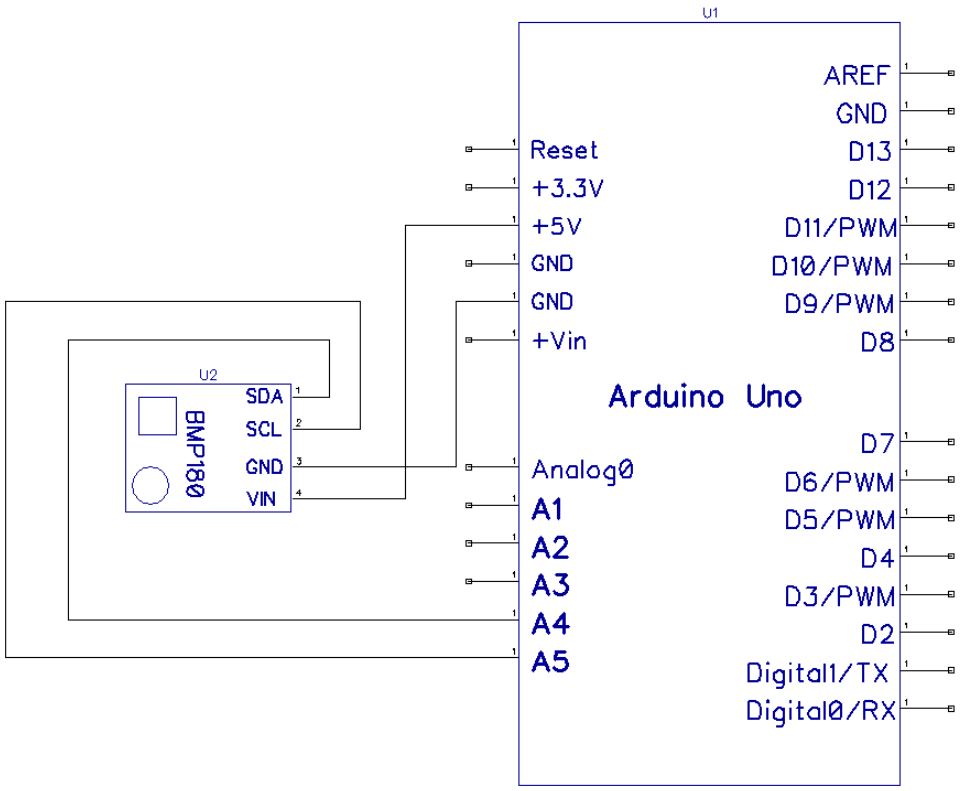

Рисунок 3.10 – Подключение BMP180 к Arduino Uno

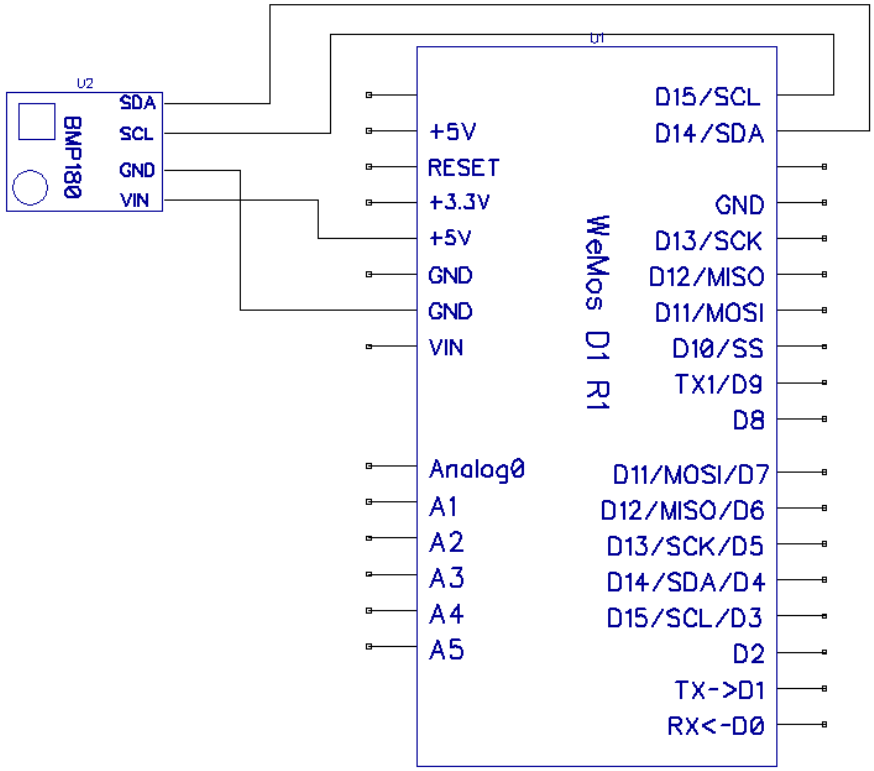

Рисунок 3.11 – Подключение BMP180 к WeMos D1 R1

Передача данных между датчиком и МК устроена более сложно, чем в DHT, так как на датчике организована EEPROM и RAM память.

Доступ к данным регистров датчика BMP180:

Каждый регистр датчика хранит 1 байт данных. Так как модуль использует интерфейс передачи данных I2C, то и доступ к данным охарактеризован им.

Запись данных в регистры:

Отправляем 1й байт (адрес датчика 0x77 и бит «R/W»=«0»); отправляем 2ой байт (адрес нужного нам регистра); отправляем 3й байт (данные для записи); после каждого отправленного байта, получаем ответ от датчика в виде одного бита «ACK». На рисунке 3.12 представлен пример записи в регистр 0xF4 значения 0xB4.

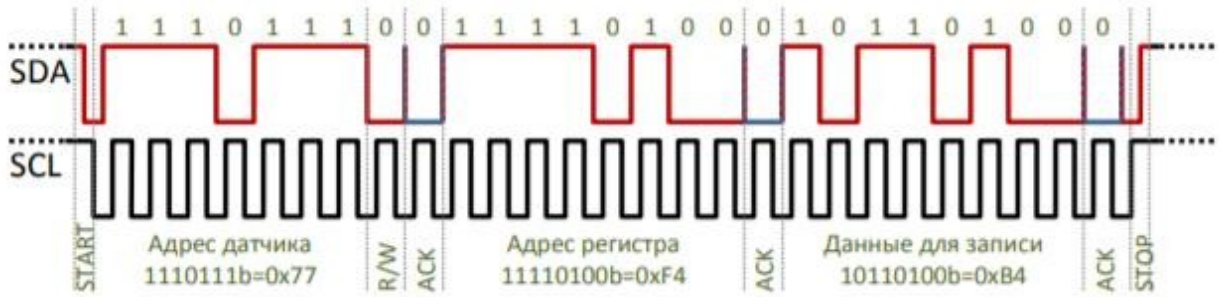

Рисунок 3.12 – Пример записи данных в регистр

Чтение данных из регистров:

Отправляем 1й байт, (адрес датчика 0x77 и бит «R/W»=«0»); отправляем 2ой байт (адрес нужного нам регистра); отправляем сигнал «RESTART»; отправляем 3й байт, (адрес датчика 0x77 и бит «R/W»=«1»); датчик ответит одним байтом данных из указанного регистра; если подать сигнал «ACK», то датчик передаст байт данных следующего регистра и т.д. пока мы не передадим сигнал «NACK». Если на шине только один ведущий, то после передачи двух первых байт (адреса датчика с битом «R/W» = «0» и адреса регистра) допустимо завершить пакет подачей сигнала «STOP» и начать новый пакет сигналом «START» передать адрес датчика с битом «R/W» после чего начать принимать или передавать данные. Такой вариант передачи данных позволяет использовать библиотеки, в которых нет сигнала «RESTART». На рисунке 3.13 представлен пример чтения байта из регистра 0xF6 (датчик ответил значением 0x5C).

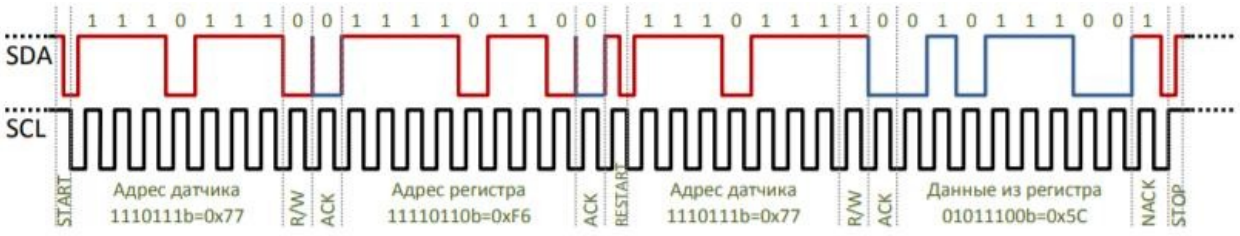

Рисунок 3.13 – Пример чтения данных из регистра

4. Моделирование системы IoT

4.1. Модель IoT с обменом данных внутри локальной сети

Рассмотрим работу Ethernet Shield в локальной сети. Для начала подключим сам Ethernet Shield к плате Arduino. Затем к Ethernet Shield подключим выбранные ранее климатические датчики. Схема подключения представлена на рисунке 4.1.1.

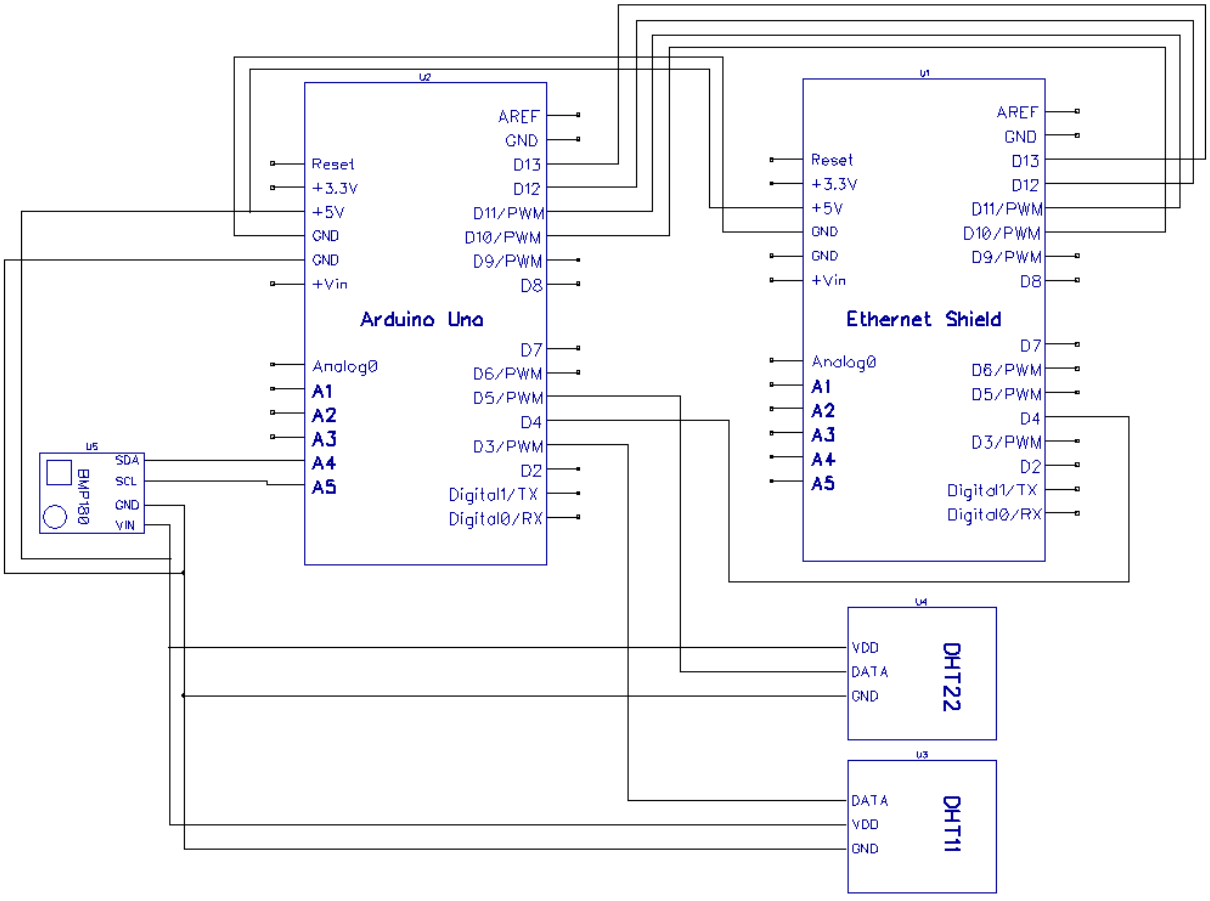

Рисунок 4.1.1 – Схема подключения климатических датчиков и Arduino Uno через Ethernet Shield

Далее подключаем Arduino Uno к компьютеру и соединяем Ethernet Shield с Wi-Fi роутером. Запускаем Arduino IDE и копируем в окно программы скетч из Приложения А. Скетч подробно закомментирован и для удобства визуально разделен на блоки.

После компиляции скетча открываем монитор порта, так как в нем отображается состояние подключения датчика BMP180. Если датчик будет некорректно подключен, то программа завершит работу, микроконтроллер перейдет в бесконечный цикл while до тех пор, пока не будет отключено питание. Если все датчики подключены правильно, то в мониторе порта будет выводиться запрос клиента. Это означает, что программа работает верно.

Теперь на любом устройстве, подключенном к локальной сети, можно зайти в браузер и ввести в нем IP адрес Ethernet Shield. Ответом браузера на данный запрос будет html страница с данными с датчиков (см. рисунок 4.1.2). Внешний вид HTML страницы формируется в блоке 7 скетча.

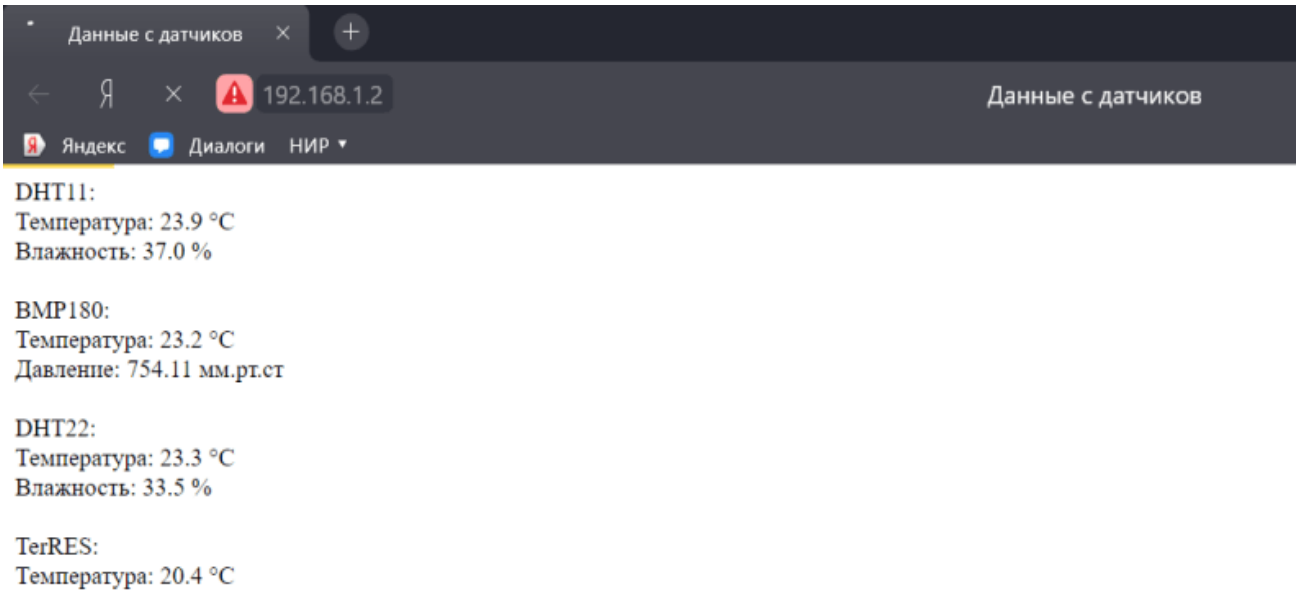

Рисунок 4.1.2 – Вид HTML страницы, формируемой сервером

Обратим внимание на то, что Arduino Uno в данном случае работает как сервер и по протоколу HTTP формирует HTML страницу по запросу клиента (в роли клиента выступает браузер).

4.2. Модель IoT с передачей данных на IoT сервис

Что бы выводить данные в сеть Интернет необходимо иметь свой сервер, который будет принимать запросы, обрабатывать их и выводить в виде html страницы. Но для данной работы открывать собственный сервер не целесообразно ввиду того, что поток данных на сервер небольшой. Так же домен для сайта, на котором будет отображаться обработанная сервером информация, стоит достаточно дорого. Поэтому в данном случае будем использовать услуги хостинга.

Сервис ThingSpeak (https://thingspeak.com/) – открытая платформа данных для проектов Internet of Things, включающая в себя сбор данных с датчиков в реальном времени, обработку этих данных, их визуализацию и использование в приложениях и плагинах.

Для начала работы с ThingSpeak необходимо зарегистрироваться на сайте. После регистрации откроется вкладка с каналами. Добавляем новый канал (см. рисунок 4.2.1).

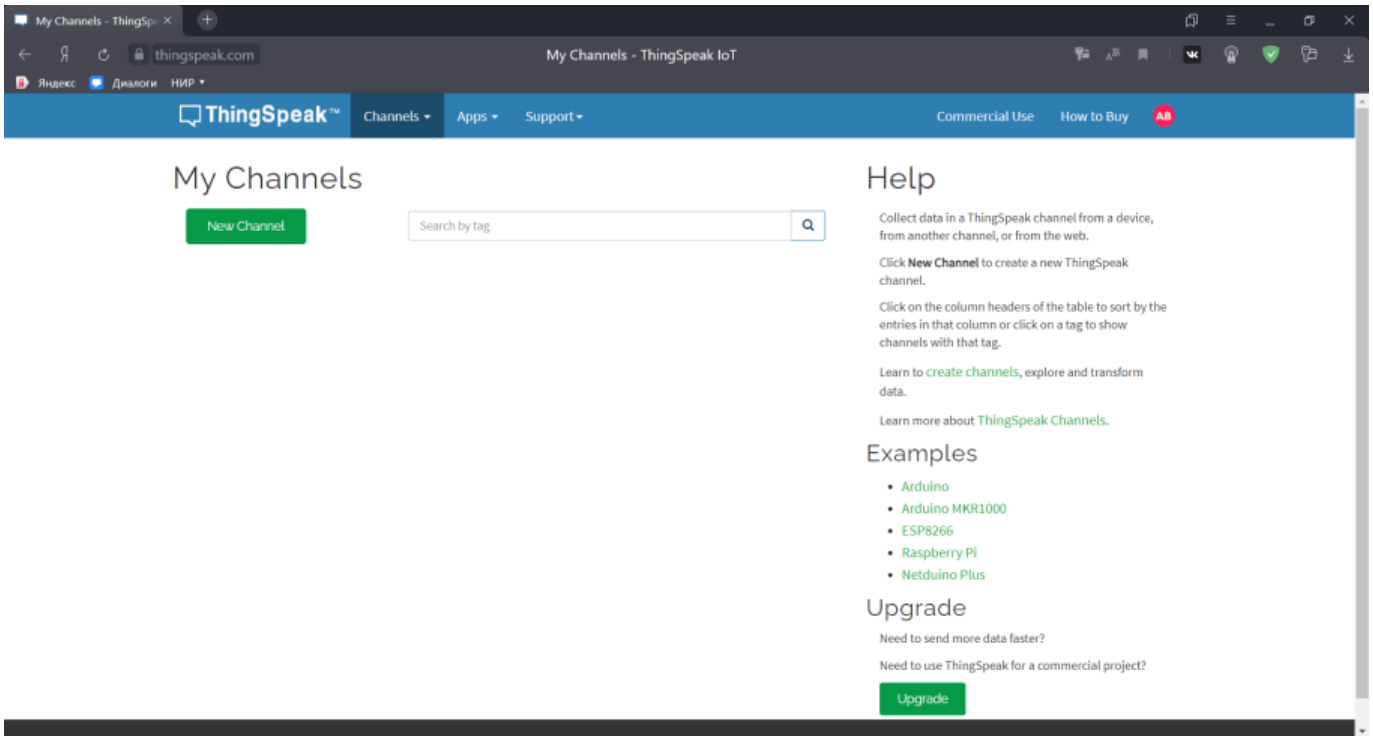

Рисунок 4.2.1 – Вкладка «Мои каналы» сайта ThingSpeak

Появляется окно с настройкой канала. В нашем случае нужно отслеживать 6 типов данных (по 2 с каждого датчика). Поэтому включаем 6 полей для отслеживания. Так же в этой вкладке можно настроить имя канала, описание, название полей, указать тэги к каналу, запустить трансляцию видео, указать геолокацию датчика (см. рисунки 4.2.2, 4.2.3). Запишем название отслеживаемых параметров в поле по принципу ДАТЧИК_ПАРАМЕТР (hum – влажность, temp – температура, pres – давление) (см. рисунок 4.2.2).

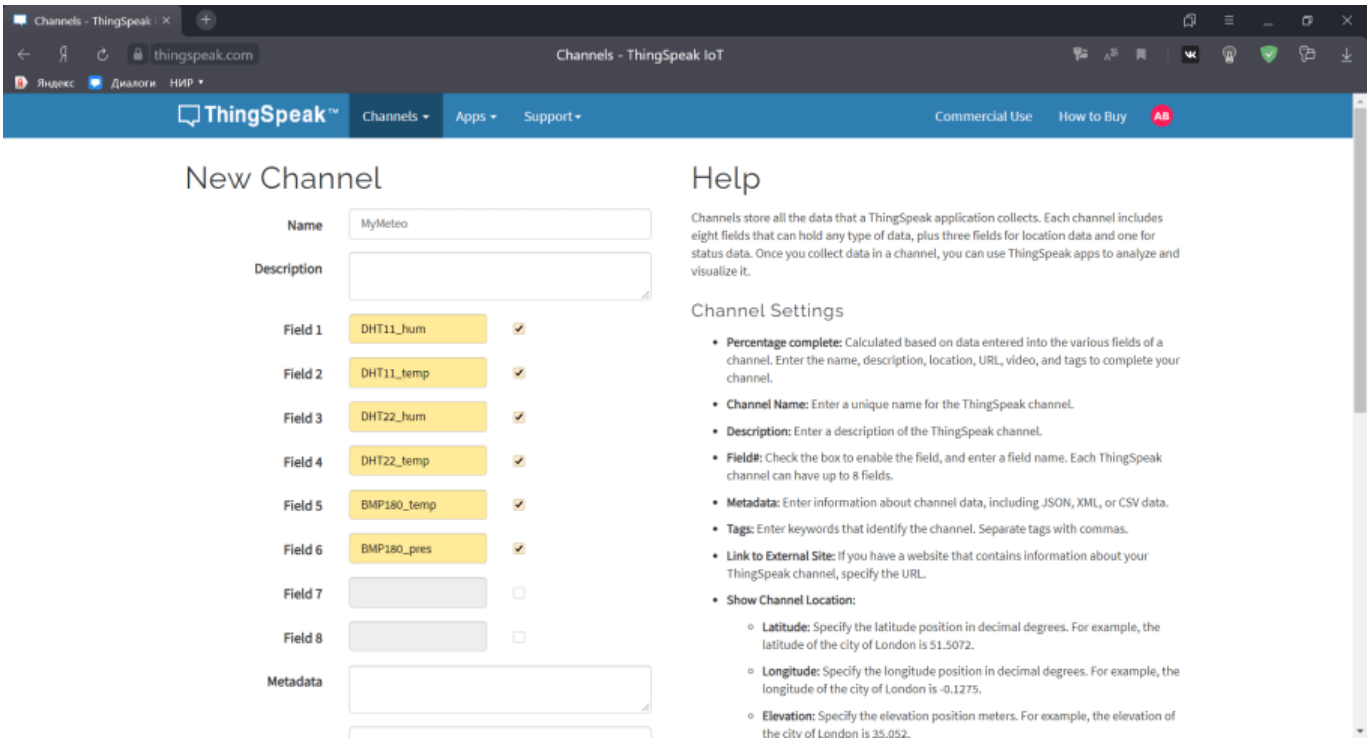

Рисунок 4.2.2 – Вкладка «настройки канала»

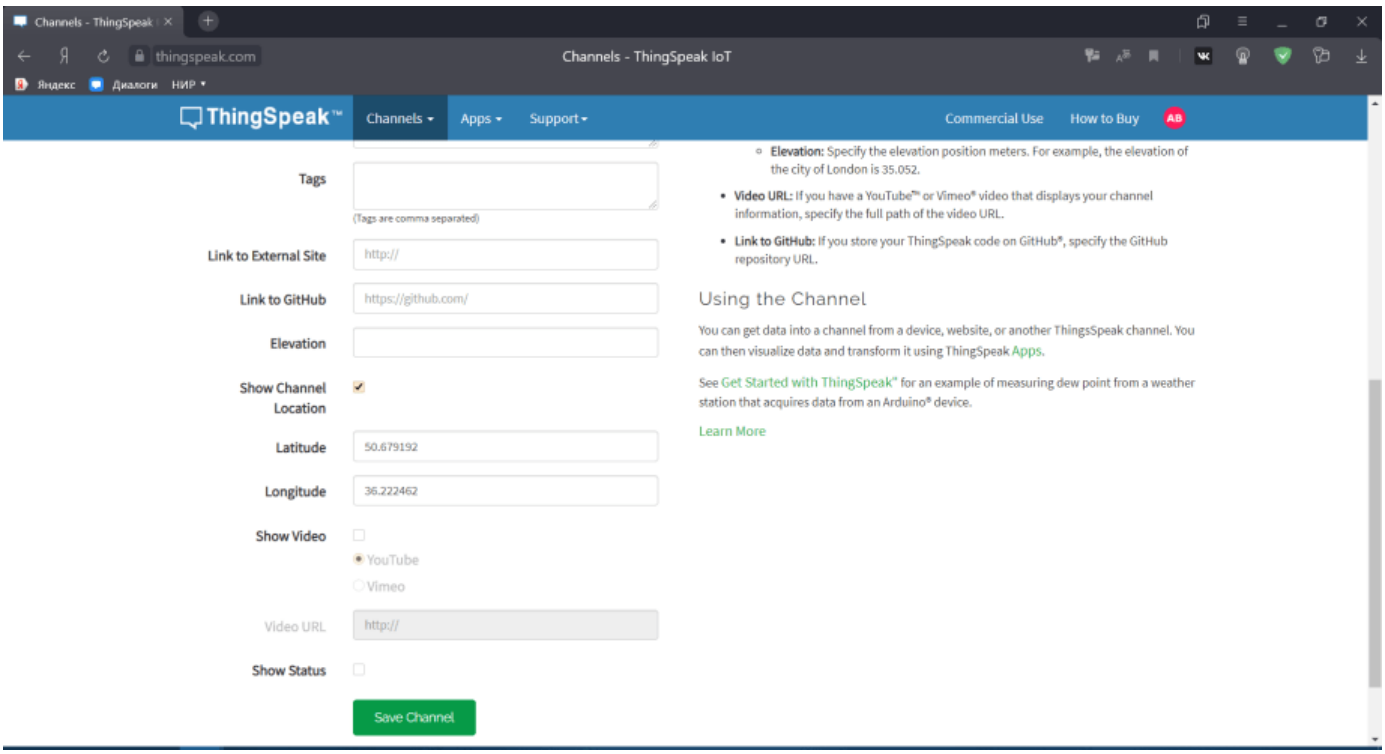

Рисунок 4.2.3 – Вкладка «настройки канала» (продолжение)

Сохраняем настройки канала, и открывается вкладка закрытого просмотра. В данной вкладке будут отображаться данные с датчиков, так же эти данные можно увидеть, только зайдя на сайт по логину и паролю, которые вводились при регистрации.

Для того, что бы начать передачу данных в «поля» нужно настроить передачу данных между сервисом ThingSpeak и Arduino. Для организации соединения между сервисом и платой используется уникальный API ключ. Когда Arduino посылает запрос на сервер, сервер проверяет API ключ, посылаемый в запросе. Если ключ в запросе совпадает с ключом на канале, то данные попадают на сервер и впоследствии выводятся на сайте.

Что бы узнать API ключ переходим во вкладку «API Keys» и копируем ключ записи (см. рисунок 4.2.4).

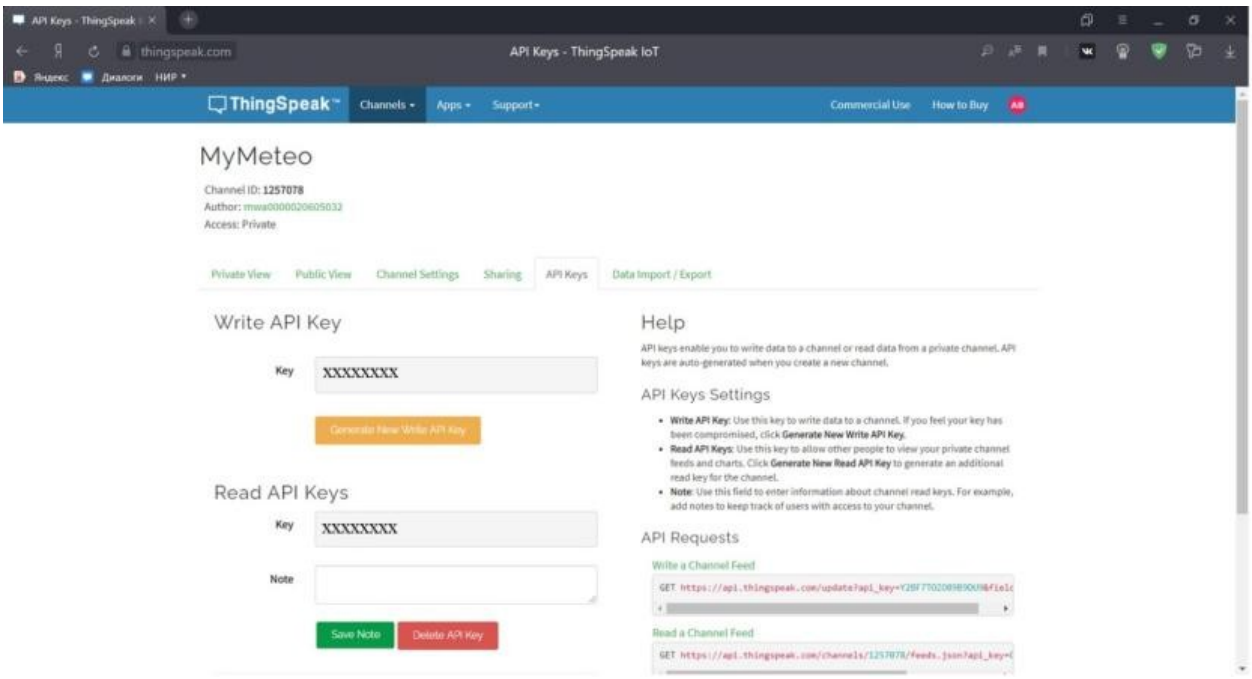

Рисунок 4.2.4 – Вкладка «API Keys»

Схема подключения датчиков остается такой же, как и для локальной сети (см. рисунок 4.1.1). Скетч для передачи данных в ThingSpeak представлен Приложении Б. Многие блоки остались такими же, как и в работе с локальной сетью, поэтому они не нуждаются в комментировании. Существенным отличием является работа Ethernet Shield как клиента. В данном случае Ethernet Shield формирует POST запрос на сайт, в котором указывается API ключ и данные с датчиков. Данный запрос представлен в блоке 8 скетча. Что бы сформировать POST запрос с нужным API ключем нужно вписать API ключ, скопированный ранее, в переменную «writeAPIKey», находящуюся в блоке 5 скетча. Так же стоит отметить, что данные передаются в одной строке (блок 7 Приложения Б). В этой строке используются специальный символ «&» для разделения данных и ключевое слово «fieldX=» для обозначения поля, в котором нужно отобразить данные.

После компиляции скетча начнется передача данных на сервера ThingSpeak. В мониторе порта будут отображаться посылаемые данные и сформированный POST запрос. При правильной последовательности действий во вкладке «Private View» начнут отображаться данные в виде

графиков (см. рисунок 4.2.5, 4.2.6, 4.2.7). Обновление данных происходит каждые 16 секунд.

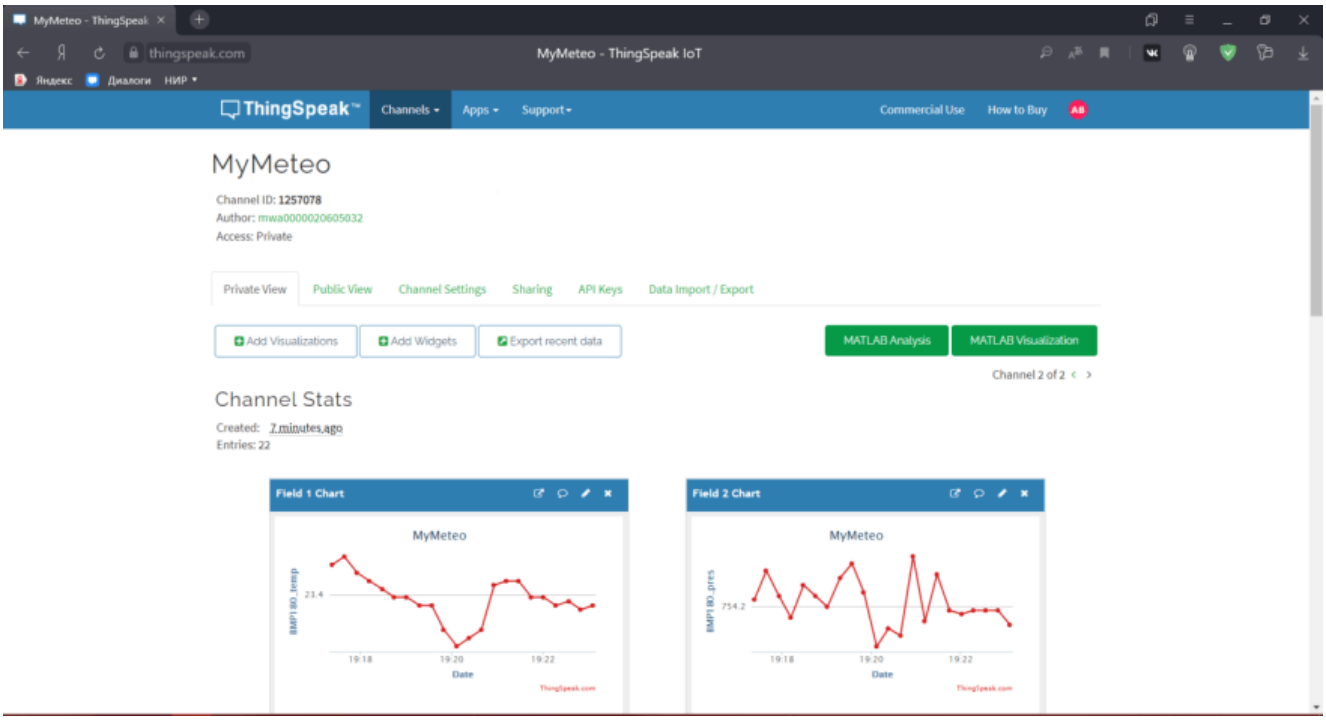

Рисунок 4.2.5 – Вкладка «Private View» после компиляции скетча

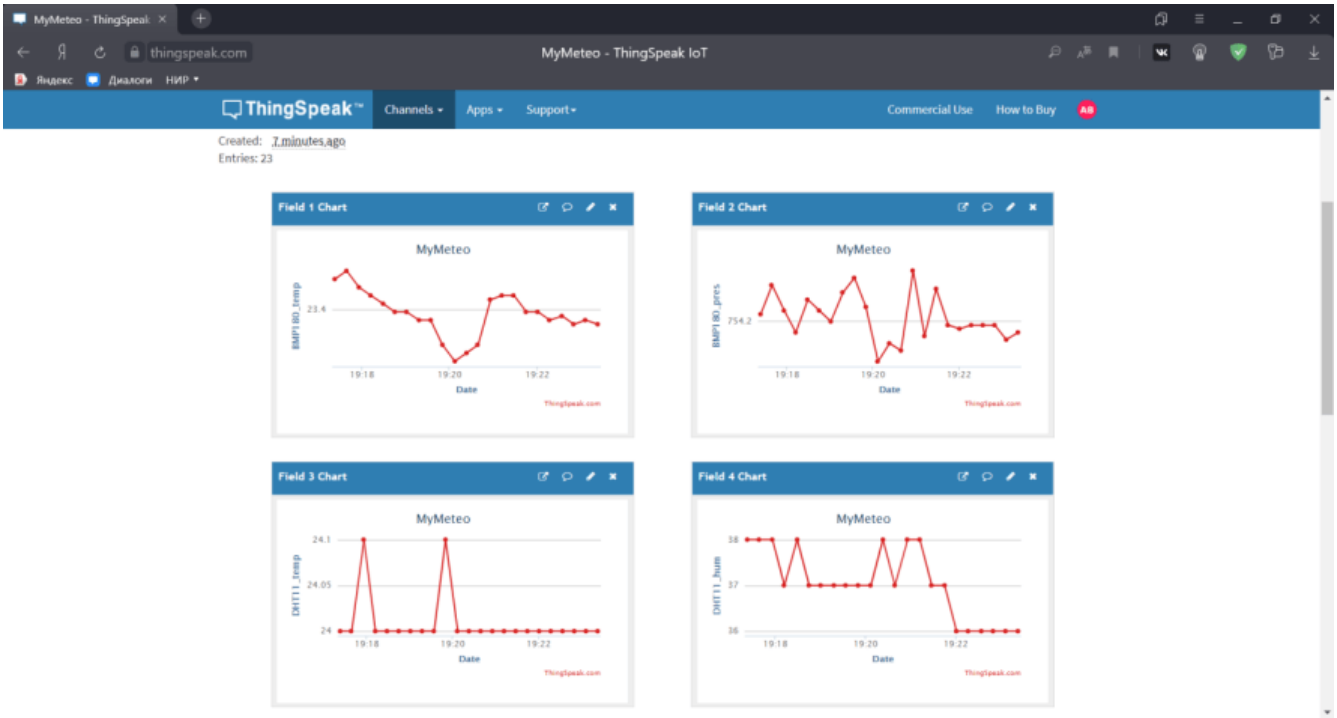

Рисунок 4.2.6 – Вкладка «Private View» после компиляции скетча (продолжение)

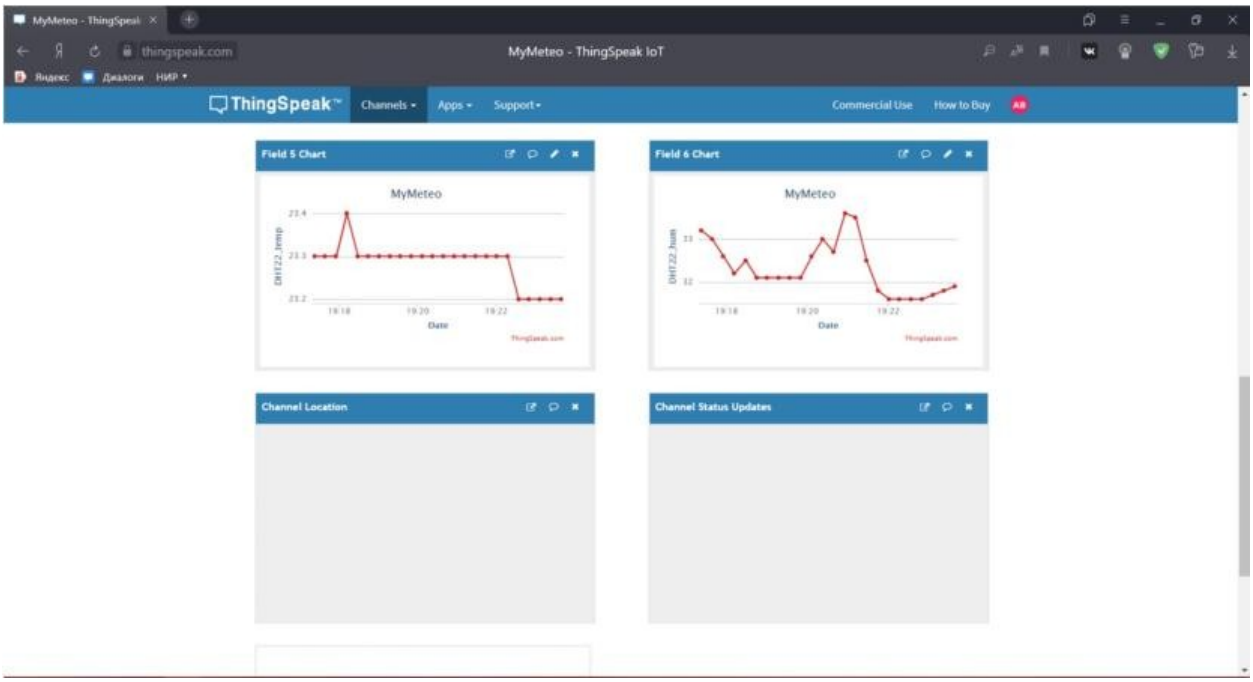

Рисунок 4.2.7 – Вкладка «Private View» после компиляции скетча (продолжение)

Теперь, даже после кратковременного мониторинга можно заметить, насколько отличается точность датчиков DHT11 и DHT22 (DHT22 более точно проводит измерения). Данные результаты совпадают с изложенными ранее в пункте 3.1.

У сервиса ThingSpeak есть дочернее приложение для операционной системы Android под названием Thing View (https://play.google.com/store/apps/details?id=com.cinetica_tech.thingview&hl=ru&gl=US). Данное приложение работает так же, как и сайт ThingThpeak и полностью повторяет его функционал. Присутствует авторизация проекта, добавление полей, настройки полей и данные об API-ключе (рисунки 4.2.8-4.2.13).

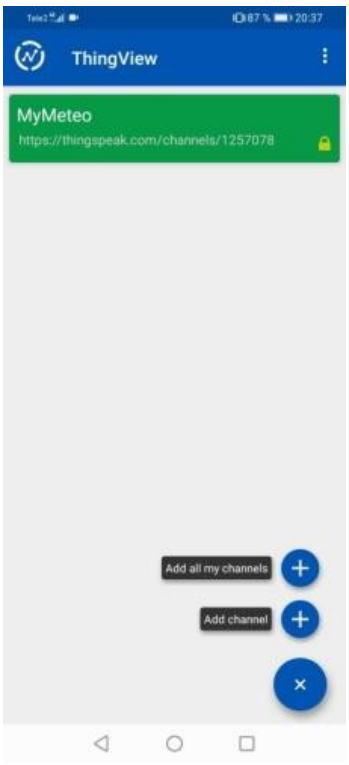

Рисунок 4.2.8 – Вкладка «Мои каналы» приложения Thing View

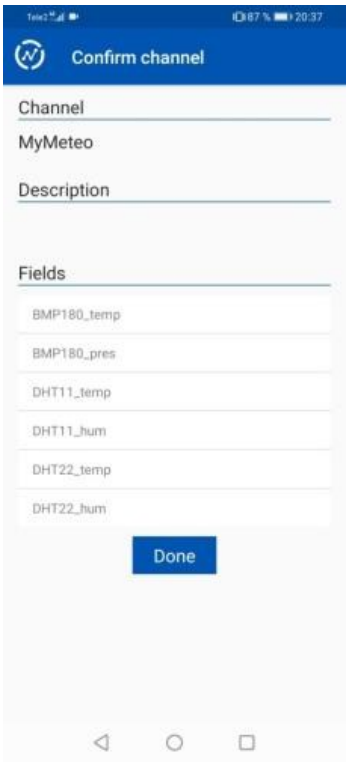

Рисунок 4.2.9 – Вкладка «Настройки канала»

Рисунок 4.2.10 – Вкладка «API Keys»

Рисунок 4.2.11 – Вкладка «Private View»

Рисунок 4.2.12 – Вкладка «Private View» (продолжение)

Рисунок 4.2.13 – Вкладка «Private View» (продолжение)

Из приведенных скриншотов видно, что приложение функционирует так же, как и сайт Thing Speak. На графиках, на рисунках 4.2.11-4.2.13

заметен скачек значений на всех датчиках. Связано это с тем, что время по оси X устанавливается в момент получения данных и из-за того, что точки на графиках соединяются последовательно по времени.

4.3. Модель IoT с передачей данных в Firebase

Зачастую, даже при наличии множества настроек, услуги хостинга не подходят для различных индивидуальных целей IoT систем. Хостинговые услуги ограничивают свободу реализации проектов и не допускают настроек дизайна приложений, web-приложений и сайтов. В случае, когда такие сервисы не подходят для конкретных проектов, используют собственный сервер для хранения, обработки и перенаправления данных. Но, как отмечалось в пункте 4.2., создание собственного сервера для небольшого потока данных IoT систем нецелесообразно.

Эта проблема решается в многофункциональном сервисе от кампании Google под названием «Firebase» (https://firebase.google.com/). В общем виде данный сервис представляет собой облачную систему управления базами данных, которая позволяет хранить данные на серверах кампании, при этом свободно добавлять и перенаправлять их нескольким пользователям. Передача данных на сервис происходит так же, как и на сервисе ThingSpeak, с помощью API ключей. Но при использовании данного сервиса необходимо иметь собственное приложение или web-приложение, в котором данные будут направляться или браться с базы данных Firebase.

Для начала работы с Firebase необходимо пройти авторизацию пользователя с помощью Google аккаунта. После авторизации мы попадаем на страницу проектов (рисунок 4.3.1). При первом использовании сервиса на странице будет присутствовать только одна рамка: добавить проект.

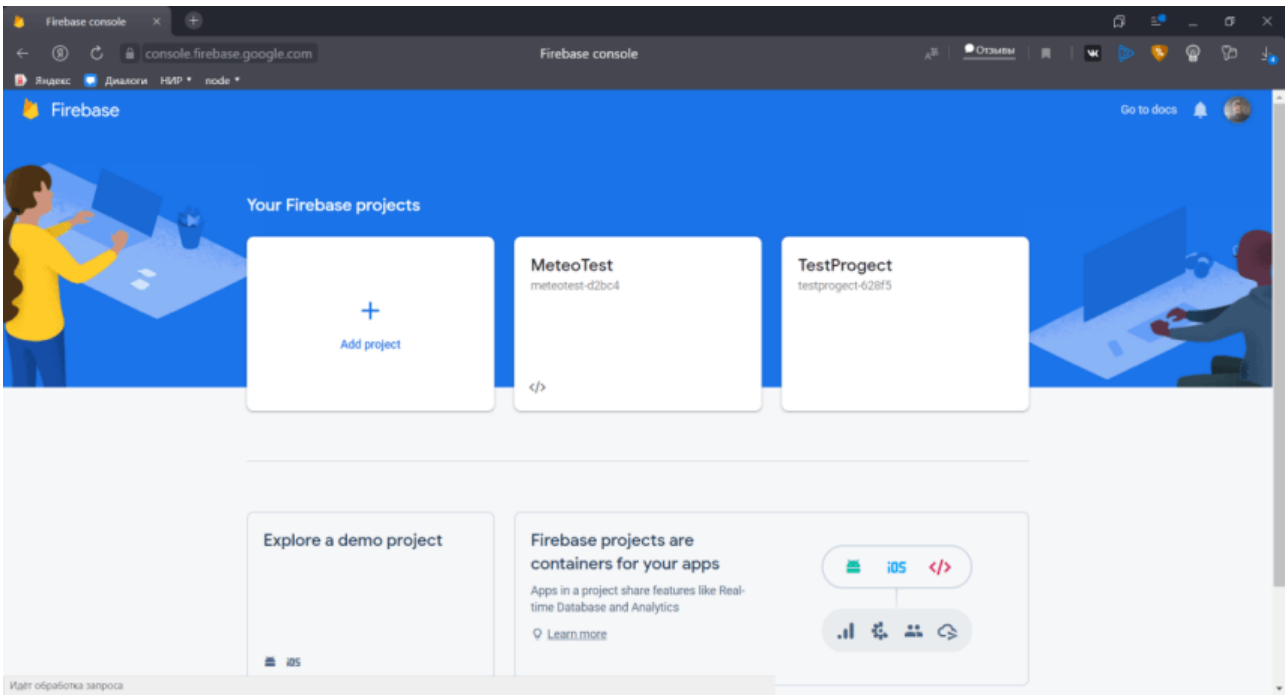

Рисунок 4.3.1 – вкладка «Проекты»

При нажатии на эту рамку мы переходим к созданию проекта и его настройки. Первым делом необходимо назвать проект (рисунок 4.3.2).

Далее нам предлагают использовать Google аналитику в проекте (рисунок 4.3.3). Эта функция полезна для коммерческих приложений, направленных на большую группу пользователей, так как аналитика помогает в мониторинге использования приложения пользователями (например, мониторинг пользователей, зашедших в приложение в заданные сроки). Но для данного проекта Google аналитика не нужна.

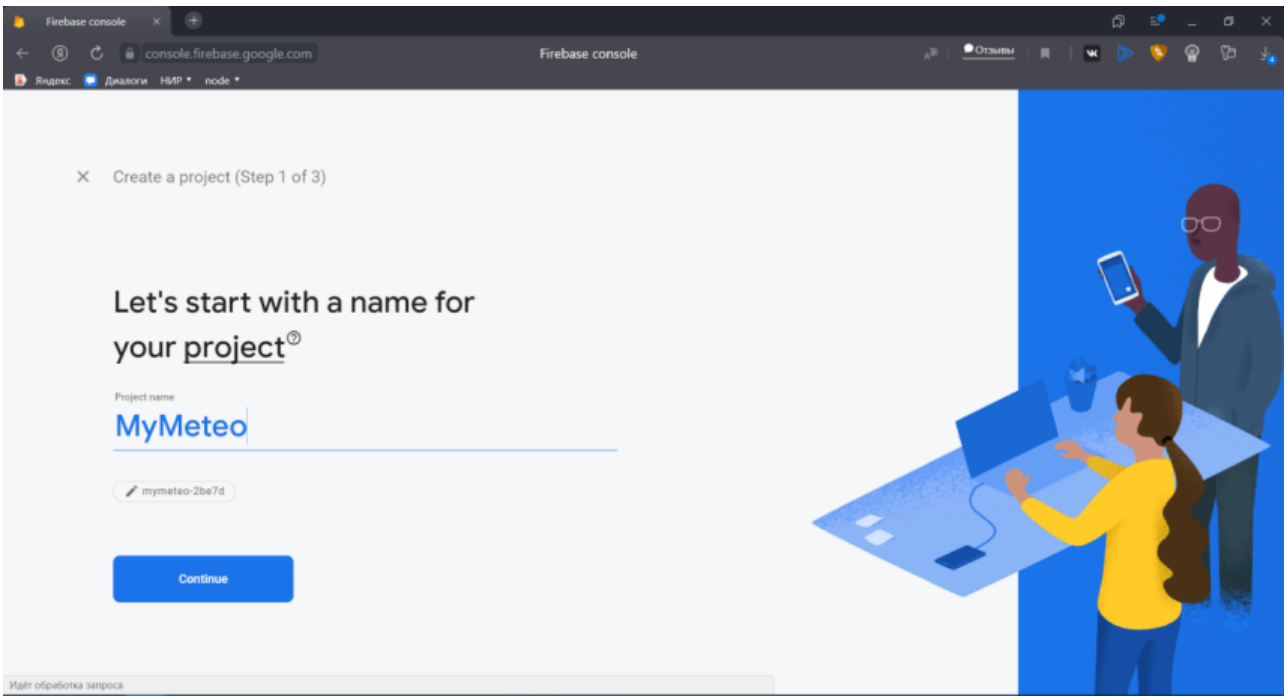

Рисунок 4.3.2 – вкладка «Добавление проекта»

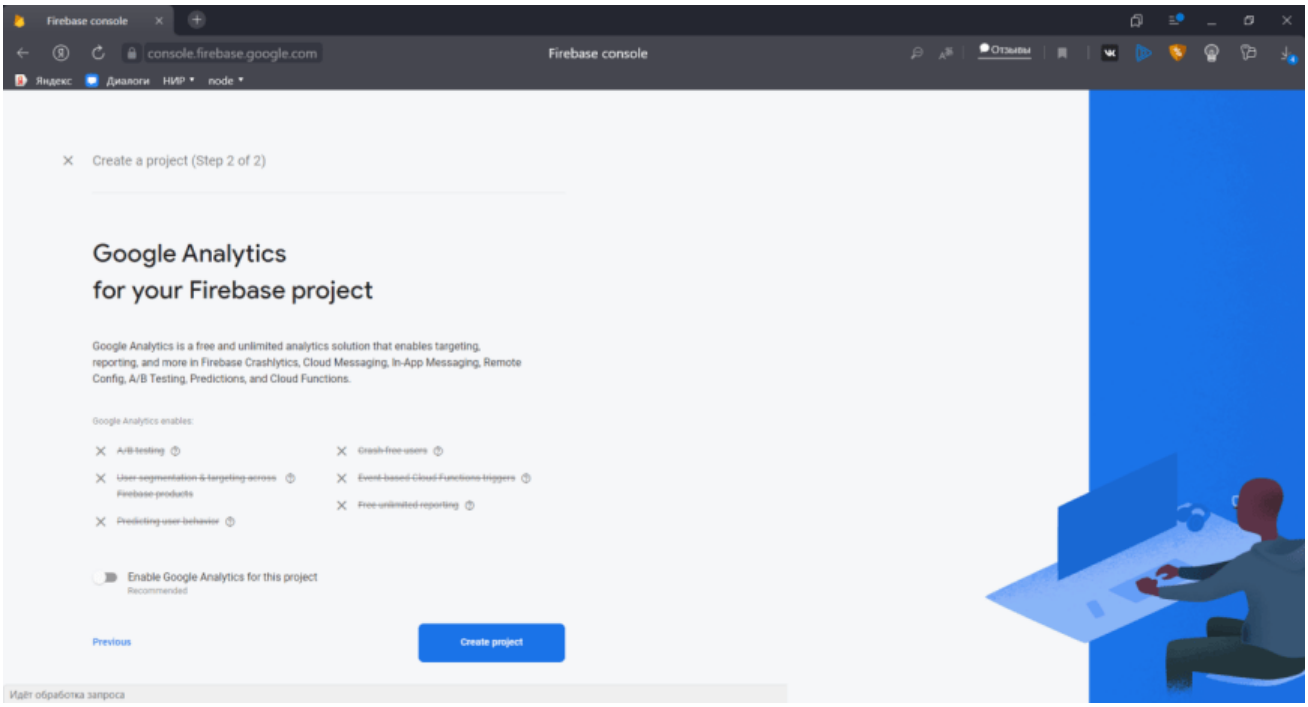

Рисунок 4.3.3 – вкладка «Google аналитика»

После создания проекта открывается вкладка домашней страницы проекта (рисунок 4.3.4). На этой вкладке располагаются различные функции для проекта, например, добавление аутентификации в приложение или мониторинг качества приложения. Так же на вкладке имеются кнопки

добавления сервисов Firebase в приложение. Слева находится панель функций для приложения. Нас интересует база данных реального времени (Realtime Database).

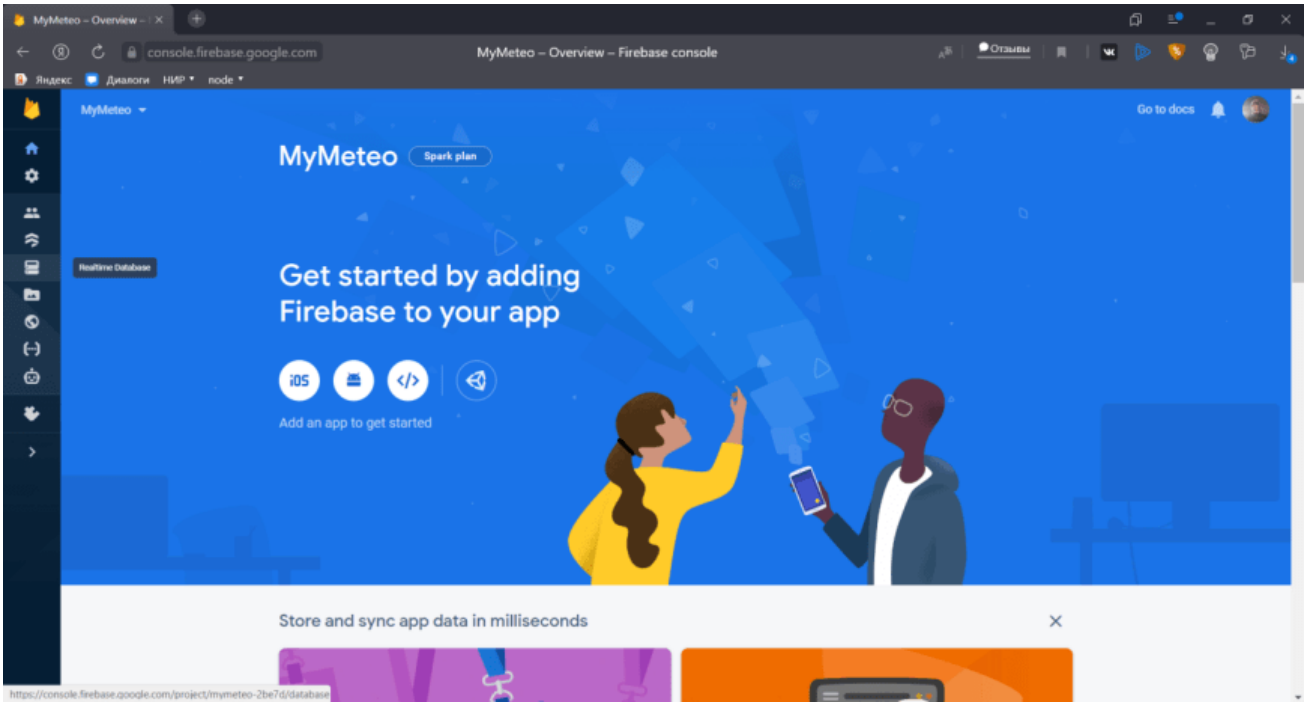

Рисунок 4.3.3 – вкладка «Домашняя страница проекта»

При нажатии на вкладку «Realtime Database» мы переходим на страницу базы данных (рисунок 4.3.4). Здесь находятся различные мануалы по работе с базой данных и кнопка создания базы данных. Создадим базу данных для приложения. В данную базу данных будут записываться данные, отсылаемые платой WeMos D1 R1, и из этой базы данных приложение будет брать значения для графиков. Данные передаются таким же образом, как и в сервисе ThingSpeak: с помощью POST запроса с передачей API ключа в нем.

При создании базы данных сначала выбирается регион, в котором находятся сервера кампании, на которые будут передаваться данные (рисунок 4.3.5).

Далее настраиваются правила безопасности (рисунок 4.3.6). На выбор даются два режима работы с базой данных: заблокированный режим

(клиентский доступ к базе данных настраивается в соответствии с правилами, которые задает создатель базы данных) и тестовый режим (свободный клиентский доступ к базе данных). В тестовом режиме правила безопасности позволяют любому пользователю со ссылкой на базу данных просматривать, редактировать и удалять все данные в базе данных в течение следующих 30 дней. По истечении срока, при условии, что правила безопасности не изменялись, база данных переходит в заблокированный режим.

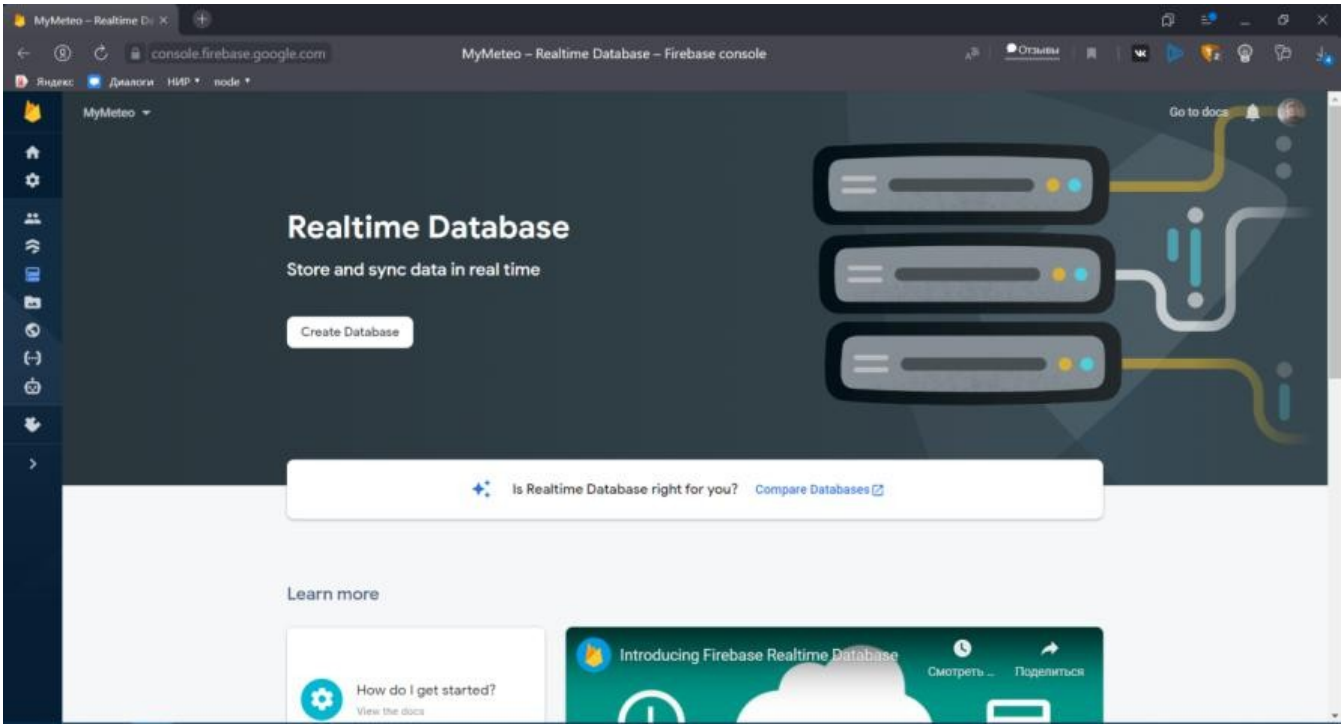

Рисунок 4.3.4 – вкладка «Realtime Database»

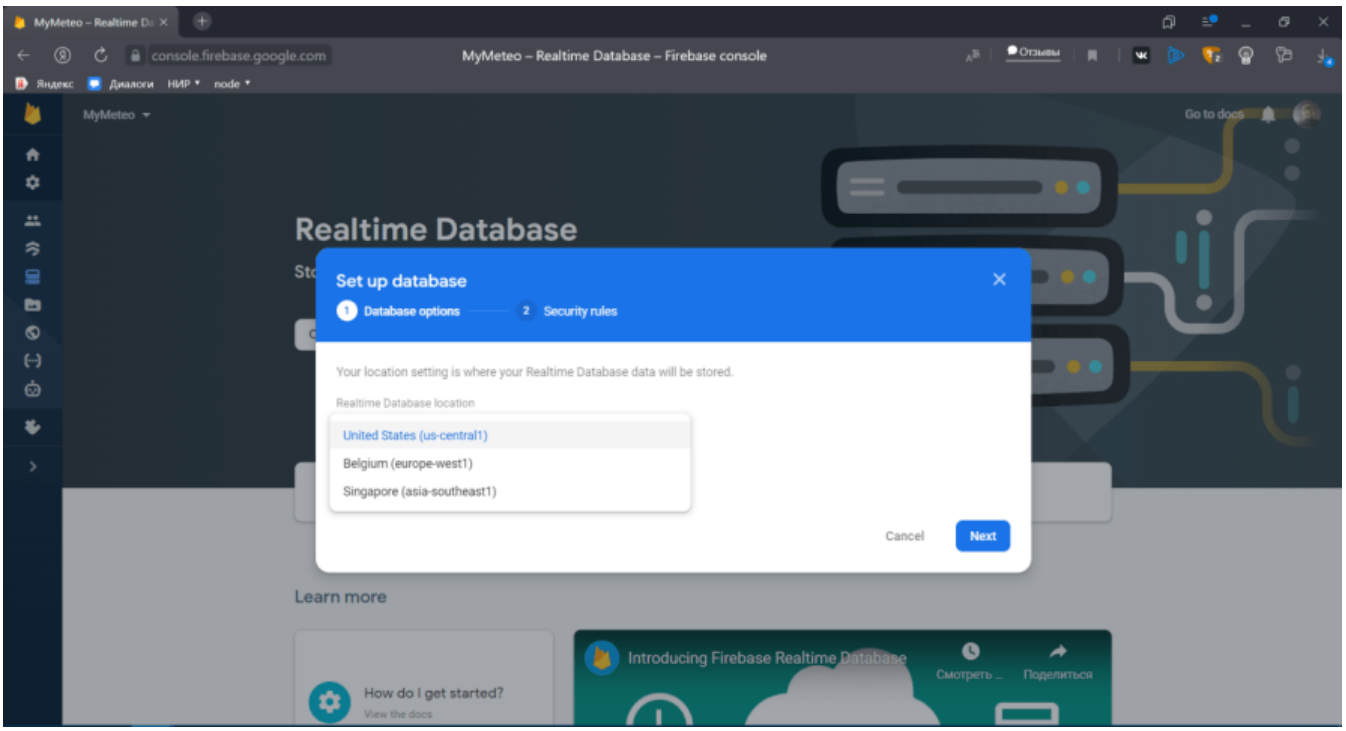

Рисунок 4.3.5 – Создание базы данных, выбор региона

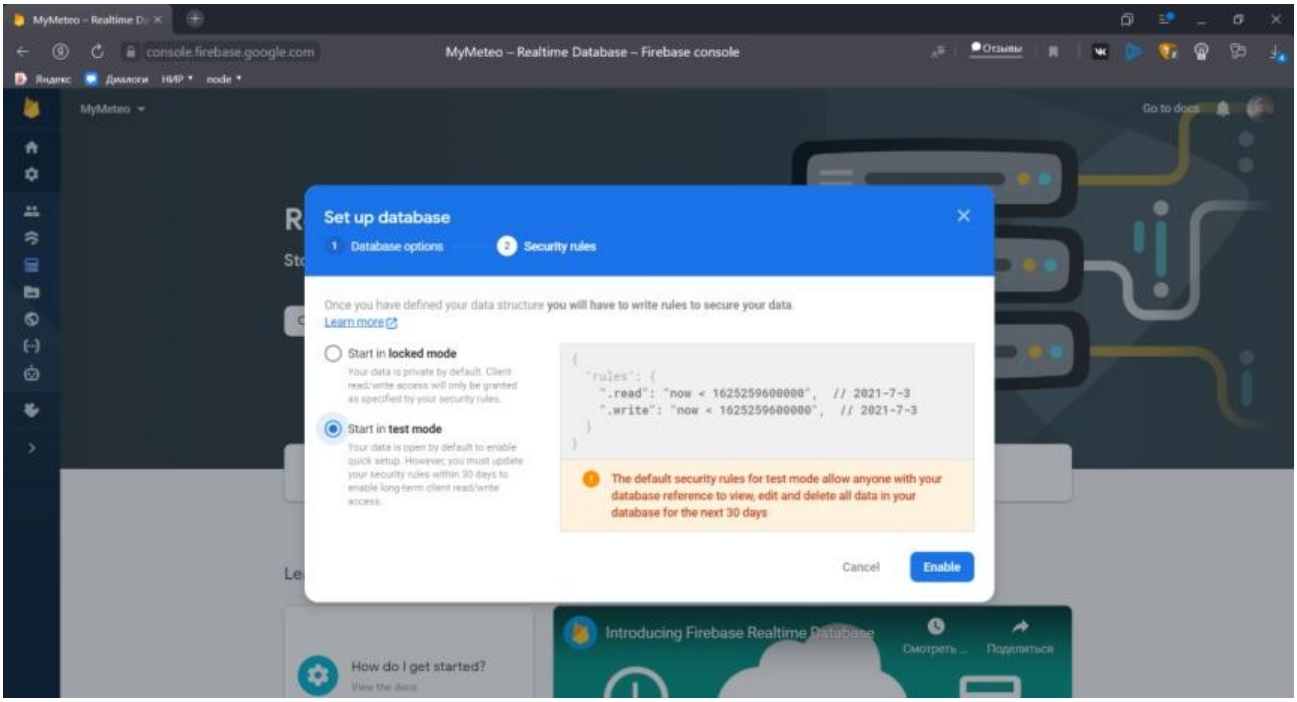

Рисунок 4.3.6 – Создание базы данных, настройка правил безопасности

В результате мы получим открытую, пустую базу данных, в которую будут записываться данные с датчиков (рисунок 4.3.7).

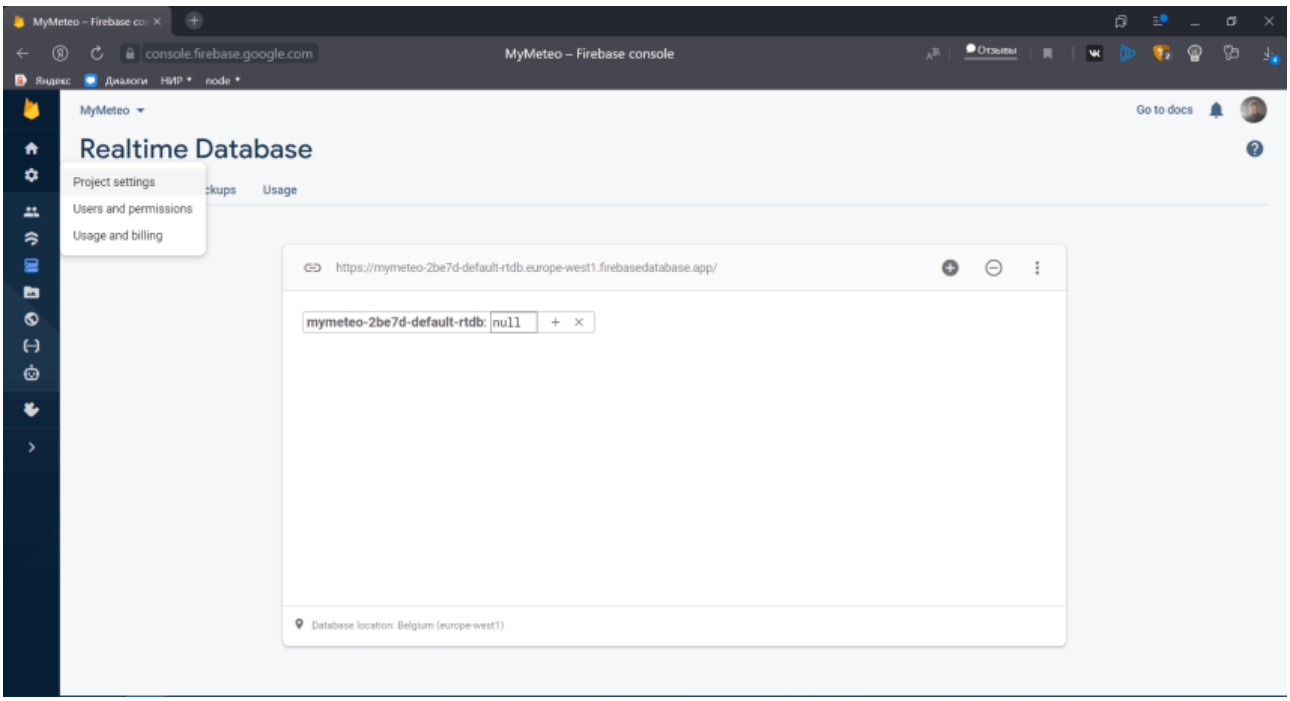

Рисунок 4.3.7 – База данных Firebase

Теперь нужно настроить правила обмена данными с базой данных, что бы можно было свободно записывать и считывать данные. Для этого нужно перейти во вкладку «Rules» и в открывшемся текстовом окне изменить значения чтения и записи на «true» (см. рисунок 4.3.8).

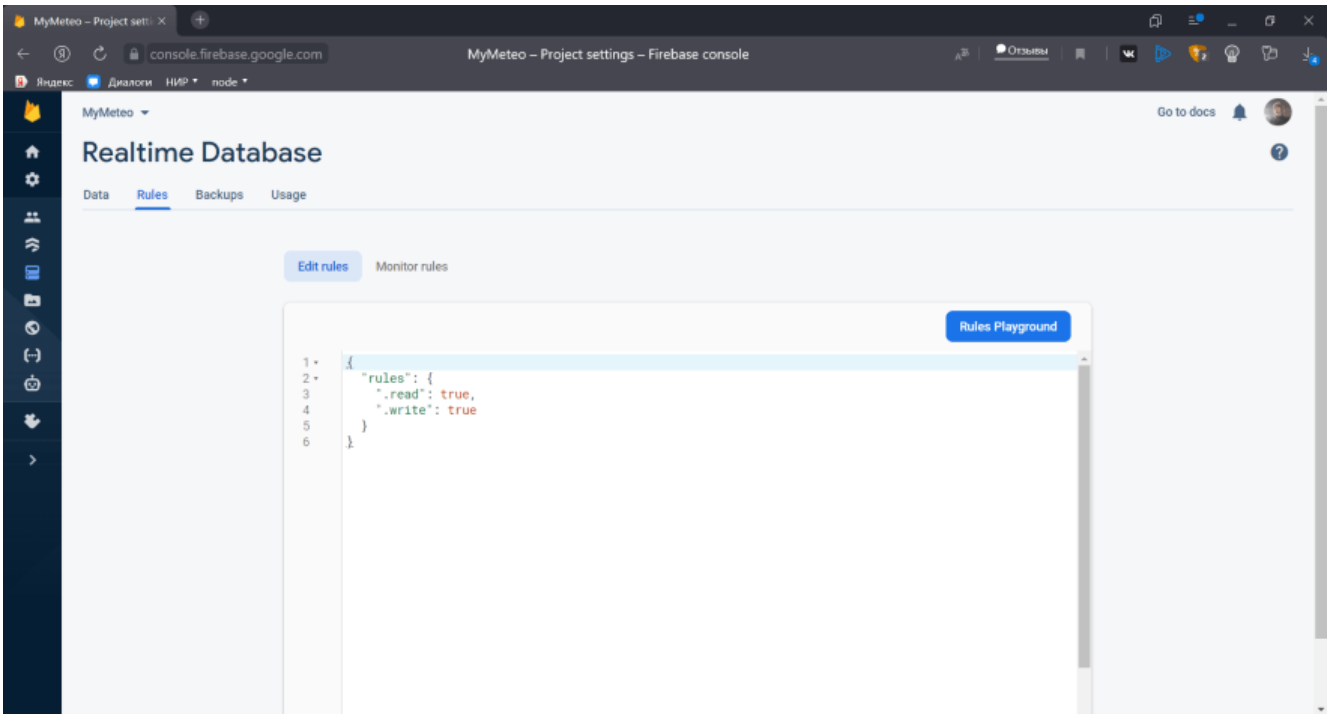

Рисунок 4.3.8 – Настройка правил базы данных

Теперь база данных полностью готова к использованию. Так как обмен данными с базой данных происходит с использованием API ключа, то его необходимо включить в запросы, как на плате WeMos D1 R1, так и в приложении. Что бы узнать API ключ сначала нужно перейти в настройки проекта. Иконка настроек находится в вертикальной панели слева (см. рисунок 4.3.7). На странице настроек можно изменить название проекта, узнать ID проекта и его номер, изменить публичное название проекта (см. рисунок 4.3.9). Далее переходим во вкладку «Service accounts» и нажимаем на «Database secrets» (см. рисунок 4.3.10). В этой вкладке располагается информация об API ключе проекта. Данный ключ необходимо скопировать и вставить в переменную «FIREBASE_AUTH», расположенную в блоке 3 приложения В.

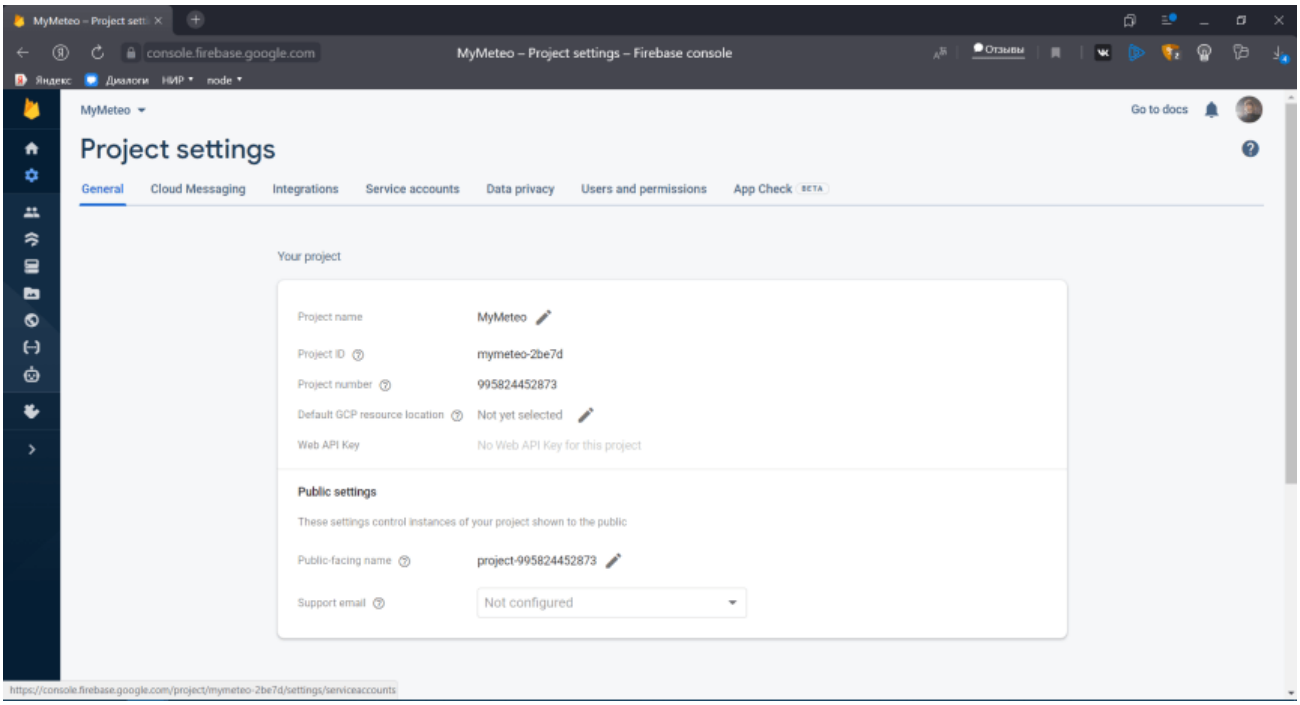

Рисунок 4.3.9 – Вкладка «Project settings»

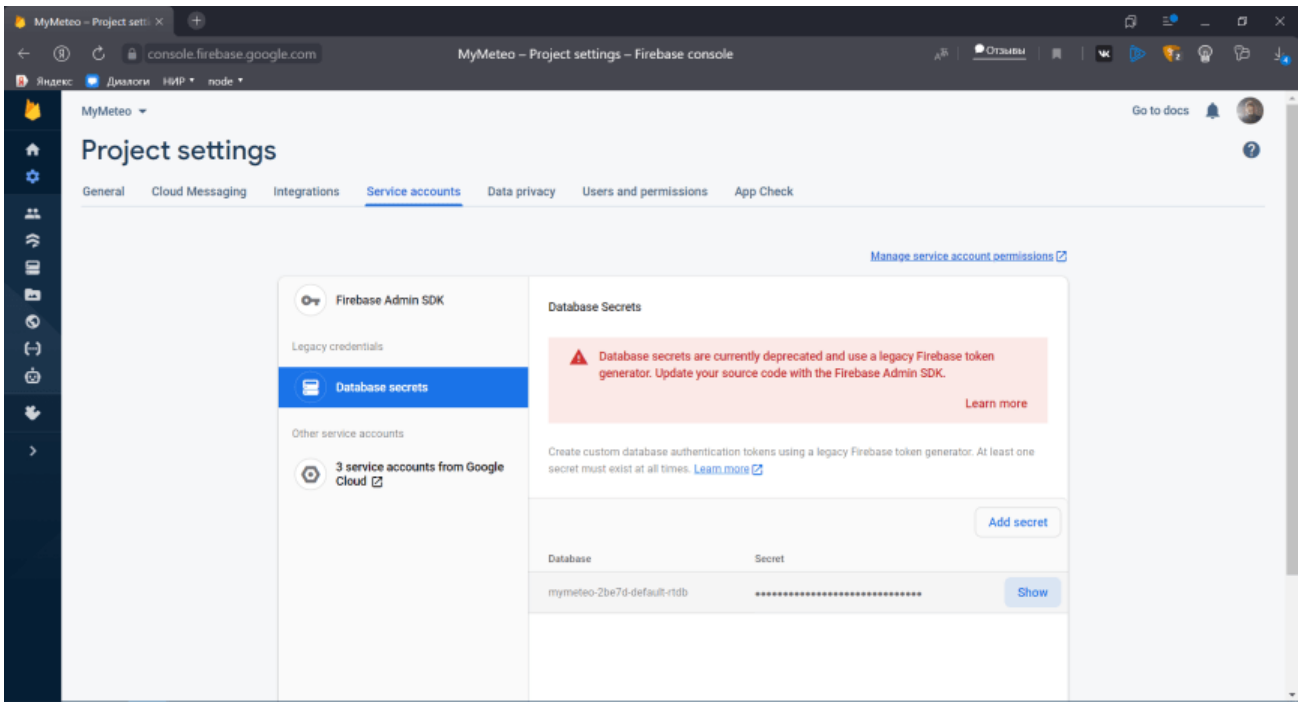

Рисунок 4.3.10 – Вкладка «Service accounts»

Далее подключаем выбранные датчики к плате WeMos D1 R1 (см. рисунок 4.3.11). Подключение датчиков к этой плате принципиально ничем не отличается от их подключения к плате Arduino UNO. Единственным существенным отличием является инициализация контактов датчиков в коде для платы WeMos R1 D1. В случае Arduino UNO в функции «#define» указывался порядковый номер контакта (например, контакту D6 соответствует цифра 6), а в случае WeMos D1 R1 указывается номер контакта GPIO (см. таблицу 2.4). Обмен данными между датчиками и платой WeMos D1 R1 осуществляется так же, как и с Arduino UNO.

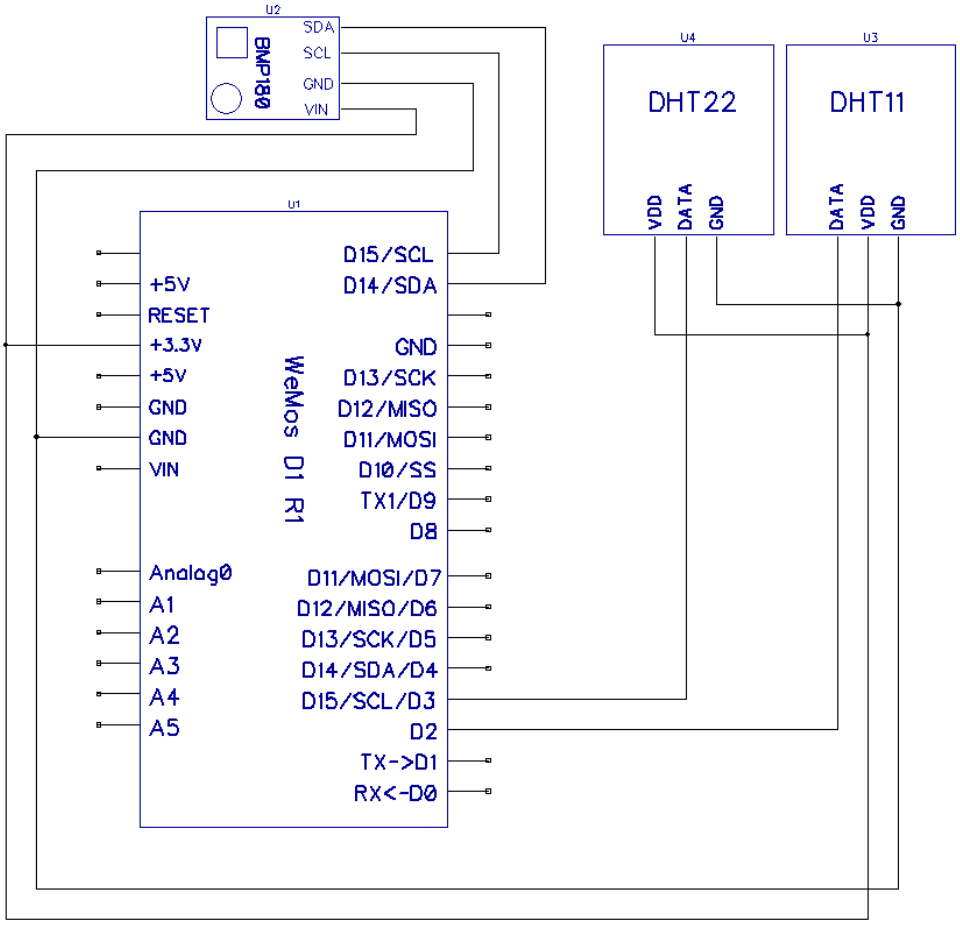

Рисунок 4.3.11 – Схема подключения климатических датчиков к плате WeMos D1 R1

Стоит подробнее разобрать код прошивки платы WeMos D1 R1, так как он имеет существенные отличия от кода для передачи данных на сервис ThingSpeak. Во-первых, используется библиотека «FirebaseESP8266» для обмена данными между платой и Firebase. Данная библиотека преобразует данные с платы в json формат и передает на базу данных Firebase. Передача данных осуществляется функцией «Firebase.setString(FirebaseData, String, String)». Данная функция преобразует и передает данные типа «строка». Первым оператором функции указывается класс базы данных, вторым оператором является ячейка базы данных, в которую будут передаваться данные, третьим оператором являются непосредственно передаваемые данные. В случае, если передача данных не

удалась, функция инициализирует переподключение платы к Wi-Fi сети и повторно передаст данные.

Так же в коде используется обмен данных с NTP-сервером для передачи времени снятия данных в базу данных. NTP-сервер (Network Time Protocol) – это сетевой протокол времени в Интернете, который включает в себя алгоритмы для синхронизации внутренних часов компьютера. Для обмена данными с NTP-сервером используется библиотека «NTPClient». Данная библиотека позволяет с помощью простых функций получить строчные данные о времени и дате в заданном часовом поясе.

После загрузки кода на плату начнется передача данных с датчиков в базу данных Firebase. Данные передаются одной строкой (блок 7 приложения В), так же, как и в случае с ThingSpeak, используются разделительные символы (в коде используется символ «q») для более удобного разделения в приложении. В результате заполненная база данных выглядит как на рисунке 4.3.12. Оранжевым цветом отмечается блок, в который были внесены изменения, зеленым цветом отмечаются добавленные данные. Так же красным цветом отмечаются данные перед удалением их из базы.

Так же стоит отметить, что база данных Firebase ограничена. В бесплатном пользовании доступно 100 Гбайт места, занимаемого базой данных, что более чем достаточно для данного проекта. Это ограничение можно снять, купив платную подписку.

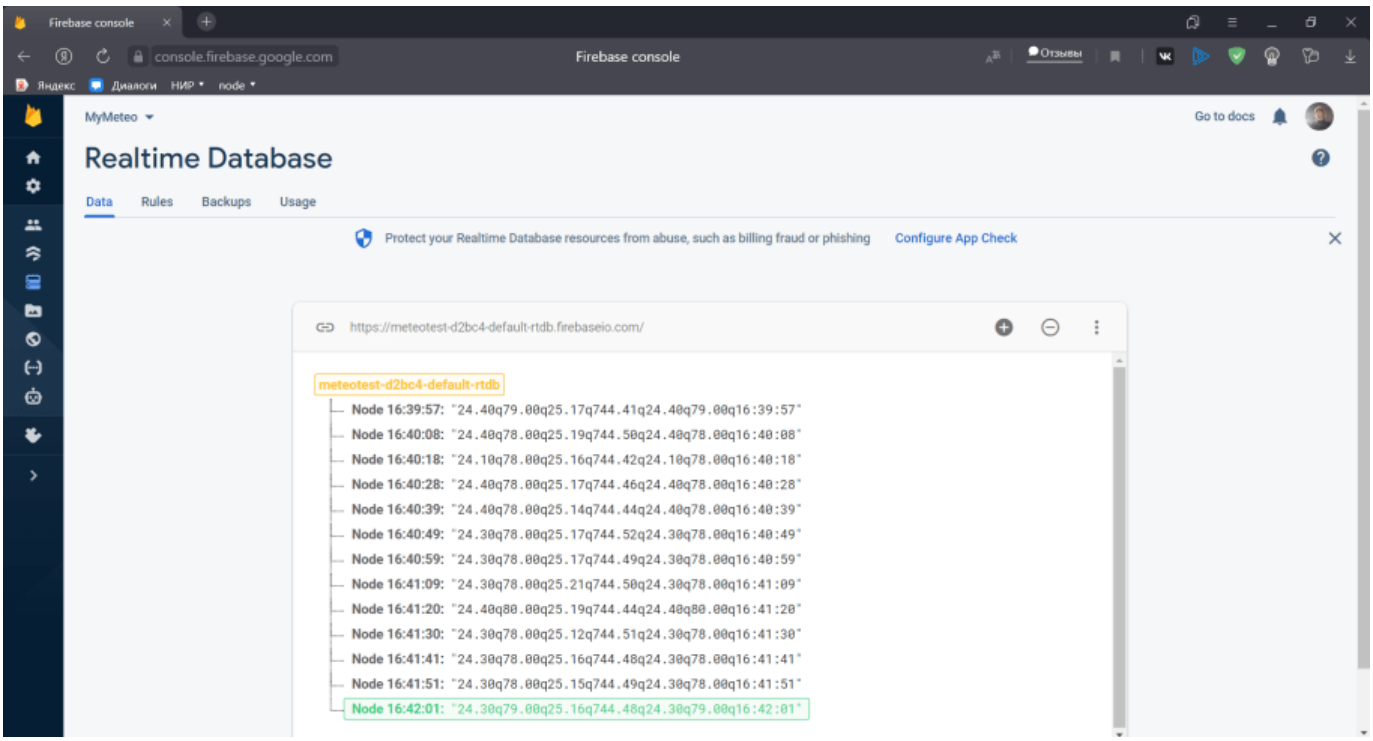

Рисунок 4.3.12 – Заполненная база данных Firebase

Последним шагом остается визуализация данных, расположенных в базе. При написании кода для приложения так же используется API ключ и ссылка на базу данных для соединения приложения с Firebase. Используем простое приложение, находящееся в открытом доступе и с открытым исходным кодом – Get Meteo (https://github.com/nvsces/meteo). В приложении присутствует только одно окно с несколькими формами для графиков (рисунок 4.3.13-4.3.15). Данные поточечно, в соответствии со временем их отправления в базу данных, отображаются в соответствующих формах.

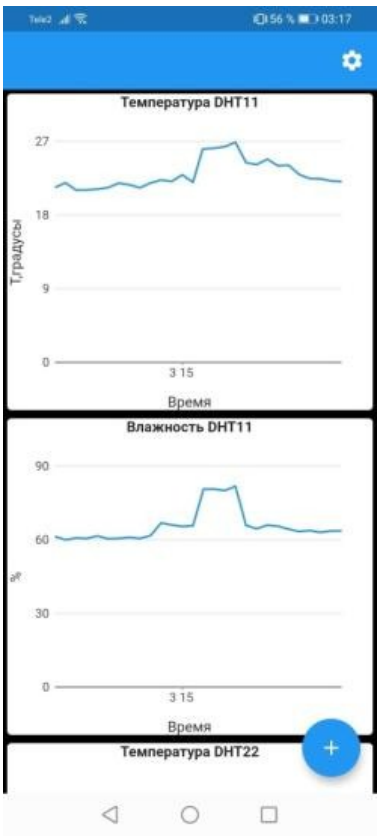

Рисунок 4.3.13 – Окно приложения Get Meteo

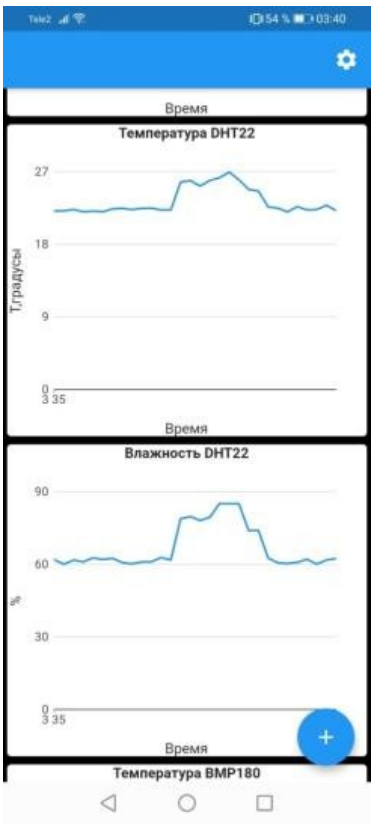

Рисунок 4.3.14 – Окно приложения Get Meteo (продолжение)

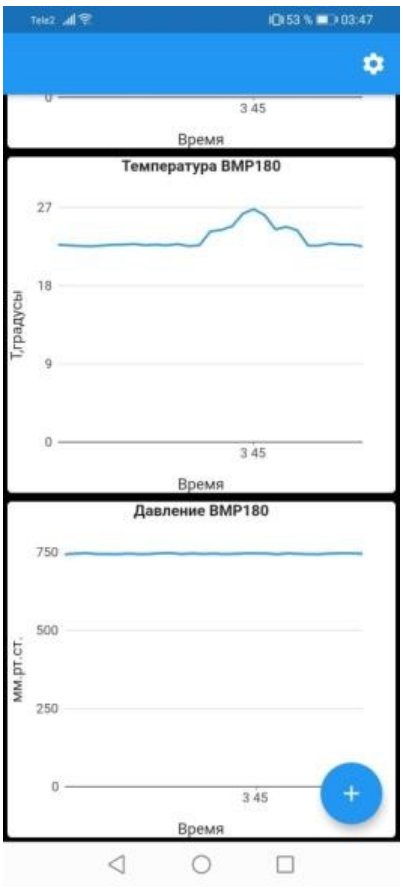

Рисунок 4.3.15 – Окно приложения Get Meteo (продолжение)

5. Экономический расчет составляющих модели

Составим список затрат на данный проект (см. таблицу 5).

Таблица 5 – Смета компонентов

Компонент	Цена, ₽
Arduino UNO	1780
WeMos D1 R1	380
Ethernet Shield W5100	1040
BMP 180	80
DHT11	160
DHT22	275
Итог	3715

Как видно из таблицы, самыми дорогими элементами для модели являлись платы Arduino UNO и Ethernet Shield W5100. Разница в цене между этими платами и платой WeMos D1 R1 обусловлена заводом изготовителем. Поставщик платы WeMos D1 R1 – Китай, а Arduino UNO – Италия. Для платы Arduino UNO выбирались качественные и, соответственно, более дорогие детали, так как упор был сделан на долговечность платы, когда на плате WeMos D1 R1 упор был сделан на доступную цену, из чего следует выбор более дешевых и низкокачественных деталей.

Заключение

В результате выпускной квалификационной работы была спроектирована простейшая IoT система – мультизонная система мониторинга климатических параметров.

По структуре данной системы можно сказать, что она во всех 3-х случаях проектирования имеет 4 уровня структуры, что соответствует теоретическим выкладкам пункта 1.1. При передаче данных внутри локальной сети: уровень элементов (непосредственно климатические датчики и Arduino UNO в роли процессора), уровень сети (локальная Wi-Fi сеть), уровень обслуживания и прикладной уровень (Ethernet Shield W5100, выступающий в роли сервера и web-браузер, отображающий HTML страницу с Ethernet Shield). При передаче данных на сервис ThingSpeak: уровень элементов (непосредственно климатические датчики и Arduino UNO в роли процессора), уровень сети (в данном случае использовалась передача данных внутри Интернета посредством Wi-Fi роутера и Ethernet Sheild), уровень обслуживания (сервера сервиса ThingSpeak, которые обрабатывали полученные данные), прикладной уровень (визуализация данных на сайте ThingSpeak). При передаче данных на сервис Firebase: уровень элементов (непосредственно климатические датчики), уровень сети (в данном случае использовалась передача данных внутри Интернета посредством Wi-Fi роутера и платы WeMos D1 R1), уровень обслуживания (база данных Firebase, осуществляющая хранение полученных данных с датчиков), прикладной уровень (пользовательское приложение GetMeteo, осуществляющее визуализацию данных).

Ошибок в работе систем выявлено не было, все три системы устойчиво работали.

Так же была проверена работа климатических датчиков, выбранных в пункте 3. Данные, получаемые с датчиков, соответствовали реальным климатическим параметрам среды. Сбоев в работе датчиков не наблюдалось.

Сравнивая спроектированные три системы между собой, можно сделать следующие выводы: IoT система с использованием локальной сети отлично подходит для использования в помещениях, площадь которых покрывает Wi-Fi сеть, например, офис или жилое помещение, с условием того, что нет необходимости получать данные из этой системы вне помещения. В данном случае отпадает необходимость передавать данные на сторонние сервисы, так же есть возможность подстроить визуализацию данных под конкретные нужды. Использование специализированных IoT сервисов позволяет легко построить IoT систему, так как отпадает необходимость проектировать уровень обслуживания и прикладной уровень. Данная система хорошо подходит для частного использования. Последняя система, с передачей данных на Firebase, самая сложная в реализации, так как в данной системе каждый уровень сети IoT проектируется пользователем. Но, с другой стороны, данная система, полностью подстраиваемая под конкретные задачи, что является ее главным достоинством, по сравнению с системой, использующей сторонние сервисы. Так же эта система имеет преимущество перед системой с передачей данных внутри локальной сети, так как доступ к данным может осуществляться в любой точке планеты с доступом к Интернету.

Список использованных источников

1. Промышленный интернет вещей. 2020. – URL: https://investmoscow.ru/media/3340535/03 // стр. 6
2. Росляков А.В. Р75 Интернет вещей: учебное пособие / А.В. Росляков, С.В. Ваняшин, А.Ю. Гребешков. – Самара: ПГУТИ, 2015. – 200 с. // стр. 5, 7-8, 13-14
3. Arduino UNO & Genuino UNO. DataSheet – URL: https://www.arduino.cc/en/Main/ArduinoBoardUno/
4. Ethernet shield W5100. DataSheet – URL: https://www.farnell.com/datasheets/1638960.pdf
5. Петин В.А. Проекты с использованием контроллера Arduino. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2016. – 400 с. // стр. 170-173, 178-193
6. WeMos D1 R1. Техническое описание – URL: https://arduinomaster.ru/datchiki-arduino/esp8266-wemos-d1-mini-raspinovka/
7. Петин В. Создание умного дома на базе Arduino. ДМК Пресс. 2018. – 180 с. // стр. 28-31
8. BMP180 Digital Pressure Sensor. Bosh Sensortec. DataSheet – URL: https://robot-kit.ru/wa-data/public/blog/download/BST-BMP180.pdf
9. DHT11 Humidity & Temperature Sensor. D-Robotics. DataSheet – URL: https://static.chipdip.ru/lib/426/DOC001426894.pdf
10. Digital-output relative humidity & temperature sensor/module DHT22. Aosong Electronics. DataSheet – URL: https://cdn-shop.adafruit.com/datasheets/DHT22.pdf

ПРИЛОЖЕНИЕ А

```
/* Блок 1: подключения библиотек */

#include <math.h> // Библиотека мат. расчетов

#include <SFE_BMP180.h> // Библиотека для BMP180

#include <Wire.h> // Библиотека для I2C интерфейса

#include <DHT.h> // Библиотека для DHT11/DHT22

#include <SPI.h> // Библиотека для SPI интерфейса

#include <Ethernet.h> // Библиотека для Ethernet Shield

/* Блок 2: подключение датчиков */

#define DHTPIN 5 // Подключение DHT22 к пину 5

#define DHTPIN1 3 // Подключение DHT11 к пину 3

SFE_BMP180 pressure; // Переменная для датчика BMP180

#define ALTITUDE 151.0 // Высота над уровнем моря в метрах

DHT dht(DHTPIN, DHT22); //Инициация датчиков

DHT dht1(DHTPIN1, DHT11);

/* Блок 3: запись адресов для Ethernet Shield */

byte mac[] = {

  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED // MAC адрес Ethernet Shield

};

IPAddress ip(192, 168, 1, 2); // IP адрес Ethernet Shield

EthernetServer server(80); // Инициализация сервера, работающего через 80
порт

/* Блок 4: объявление переменных для данных с датчиков */

float h1; // Переменная хранения влажности с DHT11

float t1; // Переменная хранения температуры с DHT11

float h; // Переменная хранения влажности с DHT22

float t; // Переменная хранения температуры с DHT22

char status; // Переменная отслеживания состояния измерения 1 - измерение
проведено, 0 - измерение не проведено

double T,P; // Переменные температуры и давления с BMP180

/* Блок 5: Инициализация устройств*/

void setup() {

  Serial.begin(9600); // Настраиваем соединение с монитором порта на скорости
9600 бит/сек

  dht.begin(); // Инициализируем DHT22
```

```
  dht1.begin(); // Инициализируем DHT11
  Ethernet.begin(mac, ip); // Присваиваем MAC и IP адреса Ethernet Shield
  server.begin(); // Инициализируем Ethernet Shield как сервер
  // Проверка на подключение BMP180
  if (pressure.begin())
  {
    Serial.print("BMP180 ok");
  }
  else
  {
    Serial.print("BMP180 fail");
    delay(2000);
    while(1); // Бесконечный цикл. Если есть сообщение об ошибке, нужно
проверить подключение BMP180 и перезапустить плату (выключить и включить
питание)
  }
}


void loop() {
  /*Блок 6: проверка подключения клиента*/
  EthernetClient client = server.available(); // Получаем ссылку на клиента
для обмена данными
  // Проверяем, подключен ли клиент
  if (client) {
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
       if (c == '\n' && currentLineIsBlank) {
/*Блок 7: Формируем стандартный HTTP ответ в виде html страницы*/
          client.println("HTTP/1.1 200 OK");
          client.println("Connection: close");
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
```

```
          client.println("<meta http-equiv='refresh' content='5'/>");

          client.println("<meta http-equiv='content-type' content='text/html;
charset=UTF-8'>");

          client.println("<title>Данные с датчиков</title>");



          h1 = dht1.readHumidity(); //Измеряем влажность на DHT11

          t1 = dht1.readTemperature(); //Измеряем температуру на DHT11

          if (isnan(h1) || isnan(t1)) {  /* Проверка. Если не удается считать
показания, в монитор порта выводится «Ошибка считывания», и программа
завершает работу*/

            Serial.println("Ошибка считывания");

            return;

          }

          // Вывод данных с DHT11

          client.print("DHT11: </br>");

          client.print("Температура: ");

          client.print(t1,1);

          client.print(" °C </br>");

          client.print("Влажность: ");

          client.print(h1,1);

          client.print(" %");

          client.print("</br>");



          status = pressure.startTemperature(); // Старт измерения
температуры

          if (status != 0) // Проверка состояния измерения

          {

            delay(status);

            // Получаем температуру

            status = pressure.getTemperature(T);

            if (status != 0) // Проверка состояния измерения

            {

                // Вывод дынных с BMP180

                client.print("</br>");

                client.print("BMP180: </br>");

                client.print("Температура: ");
```

```
                client.print(T,1);
                client.print(" °C </br>");
                // Определяем атм. давление:
                // Параметр указывает разрешение, от 0 до 3 (чем больше
разрешение, тем выше точность, тем дольше идет измерение).
                status = pressure.startPressure(1);
                if (status != 0) // Проверка состояния измерения
                {
                    delay(status);
                    // Получаем давление
                    status = pressure.getPressure(P,T);
                    if (status != 0) // Проверка состояния измерения
                    {
                        client.print("Давление: ");
                        client.print(P*0.750064,2);
                        client.print(" мм.рт.ст </br>");
                    }
                    else Serial.println("Ошибка получения давления\n");
                 }
                    else Serial.println("Ошибка запуска получения давления\n");
               }
               else Serial.println("Ошибка получения температуры\n");
             }
             else Serial.println("Ошибка запуска получения температуры\n");

           h = dht.readHumidity(); //Измеряем влажность на DHT22
           t = dht.readTemperature(); //Измеряем температуру на DHT22

           if (isnan(h) || isnan(t)) {  /* Проверка. Если не удается считать
показания, выводится «Ошибка считывания», и программа завершает работу*/
           Serial.println("Ошибка считывания DHT22");
           return;
           }
           // Вывод данных с DHT22
           client.print("</br>");
```

```
            client.print("DHT22: </br>");
            client.print("Температура: ");
            client.print(t,1);
            client.print(" °C </br>");
            client.print("Влажность: ");
            client.print(h,1);
            client.print(" %");
            client.print("</br>");
            client.print("</br>");

            client.println("</html>"); // Конец html страницы
            break;
          }
          if (c == '\n') {
            // Старт новой линии
            currentLineIsBlank = true;
          }
          else if (c != '\r') {
            // Получение символа на текущей строке
            currentLineIsBlank = false;
          }
        }
      }
      // Даем время веб-браузеру для получения данных
      delay(1);
      // Закрываем соединение
      client.stop();
  }
}
```

ПРИЛОЖЕНИЕ Б

```
/* Блок 1: подключения библиотек */

#include <math.h>

#include <SFE_BMP180.h>

#include <Wire.h>

#include <DHT.h>

#include <SPI.h>

#include <Ethernet.h>

/* Блок 2: подключение датчиков */

#define DHTPIN 5

#define DHTPIN1 3

SFE_BMP180 pressure;

#define ALTITUDE 151.0

DHT dht(DHTPIN, DHT22);

DHT dht1(DHTPIN1, DHT11);

/* Блок 3: объявление переменных для данных с датчиков */

float h1;

float t1;

float h;

float t;

char status;

double T,P;

/* Блок 4: запись адресов для Ethernet Shield */

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

byte ip[] = { 192, 168, 1, 111 };

// IP адрес, dns сервера:

byte sdns[] = { 8, 8, 8, 8 };

// Адрес шлюза по которому роутер подключен к Интернету

byte gateway[] = { 192, 168, 1, 1 };

// Маска подсети

byte subnet[] = { 255, 255, 255, 0 };

/* Блок 5: параметры ThingSpeak*/

char thingSpeakAddress[] = "api.thingspeak.com"; //Адрес, по которому будет организовываться соединение
```

```
String writeAPIKey = "xxxxxxxx"; // Уникальный API ключ, для организации соединения

// Интервал отправки данных на сервер – 16 сек

const int updateThingSpeakInterval = 16 * 1000;

// Служебные переменные

long lastConnectionTime = 0;

boolean lastConnected = false;

int failedCounter = 0;

// Инициализация Ethernet как клиента

EthernetClient client;


void setup()

{

  Serial.begin(9600);

  startEthernet(); // Запуск Ethernet на Arduino

  dht.begin(); // Инициализируем DHT22

  dht1.begin(); // Инициализируем DHT11

  if (pressure.begin())

  {

    Serial.print("BMP180 ok");

  }

  else

  {

    Serial.print("BMP180 fail");

    delay(2000);

    while(1);

  }

}


void loop()

{

  /* Блок 6: Получение данных с датчиков*/

  String analogValue0;

  String analogValue1;

  h1 = dht1.readHumidity();
```

```
t1 = dht1.readTemperature();
if (isnan(h1) || isnan(t1)) {
  Serial.println("Ошибка считывания");
  return;
}
status = pressure.startTemperature();
delay(status);
status = pressure.getTemperature(T);
analogValue0 = String(T,DEC);
status = pressure.startPressure(1);
delay(status);
status = pressure.getPressure(P,T);
analogValue1 = String(P*0.750064);
h = dht.readHumidity();
t = dht.readTemperature();
if (isnan(h) || isnan(t)) {
  Serial.println("Ошибка считывания DHT22");
  return;
}
// Запись полученых данных в строку
String analogValue2 = String(t1,DEC);
String analogValue3 = String(h1);
String analogValue4 = String(t, DEC);
String analogValue5 = String(h);
// Печать запроса в последовательный порт
if (client.available())
{
  char c = client.read();
  Serial.print(c);
}
// Разъединение с ThingSpeak
if (!client.connected() && lastConnected)
{
  Serial.println("...disconnected");
```

```
    Serial.println();
    client.stop();
  }
  /* Блок 7: отправление данных на ThingSpeak*/
  if(!client.connected()     &&     (millis()     -     lastConnectionTime     >
updateThingSpeakInterval))
    {
    // Формирование запроса и отображение снятых значений в мониторе порта

updateThingSpeak("&field4="+analogValue3+"&field5="+analogValue4+"&field6="+a
nalogValue5+"&field1="+analogValue0+"&field2="+analogValue1+"&field3="+analog
Value2);
    Serial.println(analogValue0);
    Serial.println(analogValue1);
    Serial.println(analogValue2);
    Serial.println(analogValue3);
    Serial.println(analogValue4);
    Serial.println(analogValue5);
    }
  // При кол-ве неуспешных попыток >10 – перезапуск интернет-соединения
  if (failedCounter > 10 ) {startEthernet();}
  lastConnected = client.connected();
}
/* Блок 8: отправление данных в канал ThingSpeak*/
void updateThingSpeak(String tsData)
{
  if (client.connect(thingSpeakAddress, 80))
  {
    // Формирование POST запроса
    client.print("POST /update HTTP/1.1\n");
    client.print("Host: api.thingspeak.com\n");
    client.print("Connection: close\n");
    client.print("X-THINGSPEAKAPIKEY:  "+writeAPIKey+"\n");  //  Запись  API
ключа
    client.print("Content-Type: application/x-www-form-urlencoded\n");
    client.print("Content-Length: ");
```

```
    client.print(tsData.length()); // Запись данных
    client.print("\n\n");
    // Отображение состояния подключения в мониторе порта
    client.print(tsData);
    lastConnectionTime = millis();
    if (client.connected())
      {
      Serial.println("Connecting to ThingSpeak...");
      Serial.println();
      failedCounter = 0;
      }
    else
      {
      failedCounter++;
      Serial.println("Connection to ThingSpeak failed ("+String(failedCounter, DEC)+")");
      Serial.println();
      }
    }
  else
    {
    // Увеличение счетчика неуспешных попыток отправки данных
    failedCounter++;
    Serial.println("Connection to ThingSpeak Failed ("+String(failedCounter, DEC)+")");
    Serial.println();
    lastConnectionTime = millis();
    }
  }
/* Блок 10: перезапуск интернет-соединения*/
void startEthernet()
  {
  client.stop();
  Serial.println("Connecting Arduino to network...");
  Serial.println();
```

```
  delay(1000);
  Ethernet.begin(mac,ip,sdns,gateway,subnet);
  delay(1000);
    Serial.print("My IP address: ");
  for (byte thisByte = 0; thisByte < 4; thisByte++)
   {
    // Печать ip-адреса в монитор порта
    Serial.print(Ethernet.localIP()[thisByte], DEC);
    Serial.print(".");
    }
  Serial.println();
  delay(1000);
}
```

ПРИЛОЖЕНИЕ В

```
// Блок 1: подключение библиотек для датчиков и esp
#include <ESP8266WiFi.h> // Библиотека для ESP
#include "FirebaseESP8266.h" // Библиотека для работы с БД
#include <math.h> // Библиотека мат. расчетов
#include <SFE_BMP180.h> // Библиотека для BMP180
#include <Wire.h> // Библиотека для I2C шины
#include <DHT.h> // Библиотека для DHT
// Библиотеки для NTP сервера
#include <NTPClient.h>
#include <WiFiUdp.h>

// Блок 2: инициализация и подключение датчиков
#define DHTPIN 16 // Подключение DHT22 к контакту 5 GPIO16
#define DHTPIN1 0 // Подключение DHT11 к контакту 3 GPIO0
SFE_BMP180 pressure; // Переменная для датчика BMP180

#define ALTITUDE 151.0 // Высота над уровнем моря в метрах

DHT dht(DHTPIN, DHT22); //Инициация датчиков DHT
DHT dht1(DHTPIN1, DHT11);

const long utcOffsetInSeconds = 10800; // Часовой пояс (смещение в секундах)
WiFiUDP ntpUDP; // Инициализация NTP сервера
NTPClient timeClient(ntpUDP);

// Блок 3: объявление глобальных переменных
// Переменные для хранения даты/времени
String formattedDate;
String dayStamp;
String timeStamp;

float h1; // Переменная хранения влажности для DHT11
float t1; // Переменная хранения температуры для DHT11
```

```
float h; // Переменная хранения влажности для DHT22

float t; // Переменная хранения температуры для DHT22

char status; // Переменная отслеживания состояния измерения 1 -
 измерение проведено, 0 - измерение не проведено

double T,P; // Переменные температуры и давления для BMP180


#define WIFI_SSID "WIFI_SSID" // Имя wi-fi сети

#define WIFI_PASSWORD "WIFI_PASSWORD" // Пароль от wi-fi

#define FIREBASE_HOST "firebase-host.firebaseio.com"  // Имя хоста БД

#define FIREBASE_AUTH "xxxxxxxx" // Секретный ключ БД


FirebaseData firebaseData; // Объявление класса БД


int tact_int = 0;

String tact = "";


void setup() {

    // Блок 4: подключение и проверка датчиков

    Serial.begin(115200);

    dht.begin();

    dht1.begin();


    timeClient.begin(); // Подключение NTP сервера

    timeClient.setTimeOffset(10800); // Задаем часовой пояс в секундах


    if (pressure.begin()) {

        // Проверка на подключение BMP180

        Serial.print("BMP180 ok");

    } else {

        Serial.print("BMP180 fail");

        delay(2000);

        while(1); // Бесконечный цикл. Если есть сообщение об ошибке, нужно п
роверить подключение BMP180 и перезапустить плату (выключить и включить питан
ие)

    }
```

```
    // Подключние к wi-fi
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    Serial.print("connecting");
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    Serial.println();
    Serial.print("connected: ");
    Serial.println(WiFi.localIP());

    Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
    Firebase.reconnectWiFi(true);
}

void loop() {
    // Блок 5: получение даты/времени
    while(!timeClient.update()) {
        timeClient.forceUpdate();
    }
    formattedDate = timeClient.getFormattedTime();
    Serial.println(formattedDate);

    // Получение даты от NTP сервера
    int splitT = formattedDate.indexOf("T");
    dayStamp = formattedDate.substring(0, splitT);
    Serial.print("DATE: ");
    Serial.println(dayStamp);
    // Получение времени от NTP сервера
    timeStamp = formattedDate.substring(splitT+1, formattedDate.length()-1);
    Serial.print("HOUR: ");
    Serial.println(timeStamp);
```

```
    // Блок 6: чтение данных с датчиков

    String analogValue0;

    String analogValue1;

    h1 = dht1.readHumidity(); // Измерение влажности

    t1 = dht1.readTemperature(); // Измерение температуры

    if (isnan(h1) || isnan(t1)) {  // Проверка. Если не удается считать показ
ания, выводится «Ошибка считывания», и программа завершает работу

        Serial.println("Ошибка считывания");

        return;

    }


    status = pressure.startTemperature(); // Старт измерения температуры

    delay(status);

    status = pressure.getTemperature(T);

    analogValue0 = String(T);

    // Определение атм. давления:

    // Параметр указывает разрешение, от 0 до 3 (чем больше разрешение, тем в
ыше точность, тем дольше ждать).

    status = pressure.startPressure(1);

    delay(status);

    status = pressure.getPressure(P,T);

    analogValue1 = String(P*0.750064);


    h = dht.readHumidity(); // Измерение влажности

    t = dht.readTemperature(); // Измернеие температуры

    if (isnan(h) || isnan(t)) {  // Проверка. Если не удается считать показан
ия, выводится «Ошибка считывания», и программа завершает работу

        Serial.println("Ошибка считывания DHT22");

        return;

    }

    // Запись данных в строки

    String analogValue2 = String(t1);

    String analogValue3 = String(h1);

    String analogValue4 = String(t);

    String analogValue5 = String(h);
```

```
    //Блок 7: формирование данных на отправку

    String data = analogValue2+"q"+analogValue3+"q"+analogValue0+"q"+analogVa
lue1+"q"+formattedDate;



    Firebase.setString(firebaseData, " Node "+formattedDate, data); // Переда
ча данных в БД



    delay(10000); // 10 секунд задержки

}
```