Каталин Батрину

Проекты домашней автоматики на ESP8266

Используйте возможности этого крошечного Wi-Fi-чипа для создания захватывающих умных домашних проектов

Проекты домашней автоматики на ESP8266

Используйте возможности этого крошечного Wi-Fi-чипа для создания захватывающих умных домашних проектов.

Каталин Батрину

Об авторе

Каталин Батрину окончил Политехнический университет Бухареста по специальности «Электроника, телекоммуникации и информационные технологии». Последние 16 лет он работал разработчиком программного обеспечения в области телекоммуникаций. Он работал со старыми протоколами и новейшими сетевыми протоколами и технологиями, поэтому он испытал все недавние преобразования в телекоммуникационной отрасли. Он реализовал множество телекоммуникационных протоколов, от адаптаций доступа и магистральных коммутаторов до высокопроизводительных коммутаторов операторского класса на различных аппаратных платформах от Wintegra и Broadcom.

Интернет вещей стал для него естественной эволюцией, и теперь он сотрудничает с разными компаниями, чтобы построить мир завтрашнего дня, который сделает нашу жизнь более комфортной и безопасной. Используя ESP8266, он создал прототипы устройств, таких как контроллеры полива, интеллектуальные розетки, оконные ставни, цифровые адресные элементы управления освещением, элементы управления окружающей средой, и все они управляются непосредственно из мобильного приложения через облако. Брокер MQTT с мостом и сервером WebSockets был даже разработан для ESP8266. Скоро эти устройства станут частью нашей повседневной жизни, и мы все будем наслаждаться их функциональностью.

Не забудьте затем следить за его блогом https://myesp8266.blogspot.com и за новой платформой как услугой, которую вы скоро обнаружите на https://iotcentral.eu.

Оглавление

Предисловие 1

Глава 1. Начало работы с ESP8266 7
- **Начиная с микросхемы ESP8266** 8
- **Установка Arduino IDE** 9
  - Загрузка программы Arduino IDE 9
  - Настройка Arduino IDE 11
  - Установка ESP8266 SDK 13
- **Как установить библиотеку** 17
  - Библиотеки из репозитория Ардуино 18
  - Библиотеки нет в репозитории 19
- **Пришло время для вашей первой программы** 20
  - Увидеть результат 22
- **Подключение ESP8266 к Wi-Fi** 24
- **Получение данных из Интернета** 26
- **Отправка данных в Интернет** 35
- **Резюме** 41

Глава 2: Создание и настройка собственного сервера MQTT 42
- **Транспорт телеметрии очереди сообщений - MQTT** 42
  - Качество обслуживания 43
  - Безопасность 43
  - Сохранять сообщения и последнюю волю 44
  - Основная терминология 44
  - Подстановочные знаки в темах 46
- **Представляющий брокер Mosquitto** 47
- **ESP8266 и MQTT** 51
- **Публикация данных с ESP8266** 53
- **Получение сообщений MQTT в ESP8266** 56
- **Защита Mosquitto** 59
- **Резюме** 64

Глава 3: Создание домашнего термостата с ESP8266 65
- **SPIFFS** 65
  - Объекты SPIFFS 67
  - Объект каталога 69
  - Файловый объект 70
- **Датчик температуры** 73
- **Резюме** 85

Глава 4: Устройства управления с ESP8266 86
Использование библиотеки WiFiManager 86
Добавление параметров на страницу настройки WiFiManager и сохранение их в файл 94
ESP8266 и инфракрасная связь 98
Компоненты устройства 99
Программы и библиотеки для этого проекта 100
Резюме 104

Глава 5: Использование ESP8266 для создания системы безопасности 105
Пассивный инфракрасный датчик 105
Как работают PIR 105
Тестирование модуля PIR 109
Подключение модуля PIR к Интернету 113
Код безопасности ESP8266 PIR 122
Резюме 126

Глава 6: Защита ваших данных 127
Включение шифрования на mosquitto 127
Установка пакета openssl 127
Создание собственных сертификатов 128
Защита соединения между ESP8266 и брокером MQTT 131
Работа офлайн 135
Сохранение данных на SD-карту 140
Резюме 143

Глава 7: Связь в реальном времени 144
Веб-сокеты 144
Детали протокола 144
Потоковая передача данных с ESP8266 145
Акселерометр ADXL345 146
Подключение к ESP8266 147
Код ESP8266 148
Код бэкэнда 156
Общедоступная веб-страница 158
Резюме 159

Глава 8: Добавление мобильного приложения в умный дом 160
Установка Docker и использование контейнеров 160
Получение проявочного образа 163
Образы Docker 164
Настройка местного брокера 168
Спецификации кода ESP8266 171
Резюме 177

Предисловие

С тех пор, как был построен первый дом, человек сознательно старался его улучшить, сделать более комфортным и безопасным. Домашняя автоматизация или домотика существуют уже несколько десятилетий с точки зрения освещения и простого управления приборами, и только недавно технологии догнали идею взаимосвязанного мира, позволяющего полный контроль над вашим домом из любого места, чтобы стать реальностью.

ESP8266 - недорогой чип, позволяющий эффективно создавать систему домашней автоматики.

В этой книге будет продемонстрировано несколько простых в реализации проектов домашней автоматики - от управления реле до считывания всех видов параметров, таких как температура, влажность, освещенность или присутствие,которые позволят отправлять данные из модулей ESP8266 в ваше частное облако. Более того, вы спроектируете и создадите безопасное облако и мобильное приложение, которые обеспечат комфорт и безопасность . К концу этой книги вы сможете создавать свои собственные взаимосвязанные устройства для лучшей жизни.

На рынке доступно множество вариантов, и вы можете выбрать свой собственный модуль ESP8266 в зависимости от потребностей вашего проекта. Некоторые из широко распространенных проектов домашней автоматики включают создание портативного монитора окружающей среды, беспроводного удаленного ЖК-дисплея, диммера / переключателя переменного тока Air Gesture, интеллектуальных гаражных ворот Wi-Fi, освежителя воздуха IoT и дымовой пожарной сигнализации с подключением к Интернету.

Домашняя автоматика определенно никуда не денется, поскольку продолжает удовлетворять потребности потребителей, которые ищут более эффективные способы доступа к информации и управления домашней средой.

С домашней автоматизацией вы можете управлять своим устройством так, как хотите.

О чем эта книга

В главе 1 «Начало работы с ESP8266» рассказывается об основах работы с микросхемой ESP8266 Wi-Fi, в том числе о том, как выбрать модуль и как настроить микросхему ESP8266. Вы узнаете, как настроить плату ESP8266, чтобы ее можно было использовать в оставшейся части книги. Вы будетезнать, как выбрать правильный модуль ESP8266, поскольку на рынке доступно множество вариантов. После этого вы изучите основы чипа ESP8266 Wi-Fi и научитесь считывать данные с датчика, подключенного к чипу.

В главе 2 «Создание и настройка собственного сервера MQTT» рассказывается о создании и настройке сервера MQTT для использования с ESP8266. В этой главе вы узнаете, как получить, скомпилировать, установить и настроить сервер MQTT, который будет использоваться в главах в качестве центрального шлюза MQTT.

В главе 3 «Создание домашнего термостата с помощью ESP8266» рассказывается, как создать домашний термостат с помощью ESP8266. Вы узнаете, как измерить температуру в вашем доме с помощью термостата, как отобразить эту температуру на экране, а также как отрегулировать температуру в соответствии с вашими пожеланиями.

Глава 4, Управляющие устройства из ESP8266, показывает, как управлять бытовой техникой, которая часто присутствует в доме, например, лампами, светодиодами и другими приборами. Вы узнаете, как управлять несколькими бытовыми приборами, используя только микросхему Wi-Fi ESP8266. Благодаря возможности подключения чипа к Wi-Fi вы сможете удаленно управлять всеми устройствами.

Глава 5, Использование ESP8266 для создания системы безопасности, описывает, как построить полную систему безопасности на основе ESP8266. Вы узнаете, как подключиться к элементам модуля ESP8266, которые необходимы для системы безопасности, таким как датчики движения, камеры и сигнализация. После этого вы сможете построить полную систему безопасности на основе этих элементов.

Глава 6, Защита ваших данных. Эта глава посвящена добавлению SSL для безопасной связи между модулями ESP8266 и брокером. Вы научитесь шифровать пакеты, чтобы защитить данные и убедиться, что ваше общение остается конфиденциальным.

В главе 7, Связь в реальном времени, есть несколько случаев использования, когда вам нужно видеть данные, полученные с датчиков в реальном времени, регистрировать значения в базе данных или отображать их на красивом графике.

Глава 8 «Добавление мобильного приложения в умный дом». Чтобы завершить путешествие в мир умного дома и управлять своим домом с телефона, вы создадите мобильное приложение для Android.

Что вам понадобится для этой книги

Чтобы начать работу с ESP8266, вам понадобится ряд программных и аппаратных компонентов.

Микросхема ESP8266 и ее модуль, датчики, такие как ADXL345, датчики температуры, датчики PIR и кабели, инструменты для пайки. В каждой главе я старался сделать хорошую картинку и дать хорошее описание.

Для программ нам потребуется ПК с установленной Windows и VirtualBox. Для приложения Arduino IDE с версией SDK не ниже 1.5.3 для ESP8266, Lubuntu Linux 16.04 установлен в Virtual Box, Docker установлен в Virtual Box, InfluxDB и Grafana установлены в Virtual Box.

Для кого эта книга

Эта книга предназначена для людей, которые хотят реализовать недорогие проекты домашней автоматики с использованием микросхемы Wi-Fi ESP8266, а также полностью автоматизировать свой дом Базовое понимание платы было бы дополнительнымпреимуществом.

1

Начало работы с ESP8266

Невозможно не слышать об Интернете вещей (IoT), который начинает проникать в наши дома и нашу жизнь вместе с необходимостью ежедневно потреблять и контролировать огромное количество данных. У всех нас есть смартфон с подключением к Интернету, и мы можем мгновенно находить людей по всему миру и связываться с ними.

Если мы можем общаться и обсуждать с людьми по всему миру, почему бы не контролировать наши дома, наши машины и наши офисы с нашего смартфона? Именно здесь на сцену выходит IoT, который позволяет нам подключать к Интернету практически любой объект.

В настоящее время на рынке есть несколько микросхем, способных подключаться к Интернету, но один маленький человечек привлек внимание разработчиков своими характеристиками и ценой.

Это ESP8266, недорогой микроконтроллер МК с поддержкой TCP / IP и Wi-Fi, разработанный EspressIf Systems, компанией, расположенной в Шанхае, и мы узнаем о нем больше в этой книге. В этой главе мы рассмотрим следующие темы:

- Установка Arduino IDE
- Настройка IDE Arduino для ESP8266
- Открытие ESP8266
- Подключение вашего ESP к сети Wi-Fi

Знакомство с микросхемой ESP8266

ESP8266 - это недорогой 32-битный микроконтроллер RISC с возможностью подключения к Wi-Fi, способный работать на частотах 80 или 160 МГц. Он имеет 64 КБайт ОЗУ для инструкций и 96 КБайтОЗУ данных.

Для прошивки и другого хранилища данных к нему подключена внешняя флеш-память QSPI, размер которой может варьироваться от 512 КБ до 4 МБ. Сам чип имеет 16 контактов ввода / вывода общего назначения (GPIO), но некоторые из них используются для подключения флэш-памяти QSPI. Остальные контакты поддерживают последовательный периферийный интерфейс (SPI), I^2C, I^2S, универсальный асинхронный приемник / передатчик (UART) и один 10-битный АЦП.

Возможности Wi-Fi соответствуют IEEE 80.11 b / g / n и обеспечивают аутентификацию WPA / WPA2 иWEP, но также могут подключаться к открытым сетям.

Для этой главы вам понадобится только выбранный вами модуль ESP8266, поскольку в настоящее время доступно множество производителей и типов плат.

Теперь форм-фактор вашей платы зависит от ограничений вашего проекта, но для начала работы с этим чипом мы можем использовать одну из следующих плат:

- Плата Witty ESP12-F
- NodeMCU v1.0
- WeeMos D1 mini

Любая плата будет работать нормально, но если вы новичок, я бы порекомендовал вам начать с платы Witty ESP12-F, потому что в ней уже есть:

- LDR (фоторезистор) подключенный к аналоговому входу A0
- Светодиод RGB, подключенный к GPIO 15, GPIO 12 и GPIO13
- Кнопка, подключенная к GPIO 4

Позже, когда мы добавим другие датчики к ESP8266, этот модуль можно будет заменить любым другим модулей ESP8266.

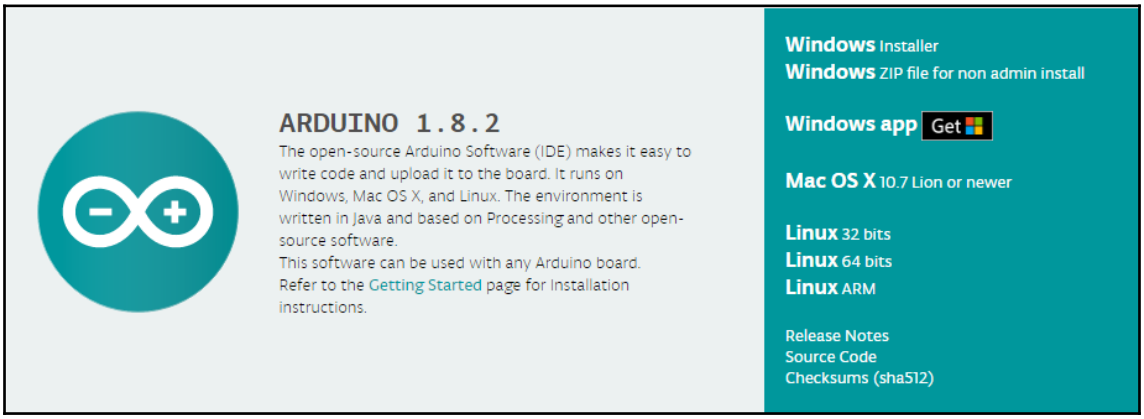
Установка Arduino IDE

ESP8266 можно использовать с официальными SDK EspressIf, которые содержат SDK NonOS и FreeRTOS для написания кода на C / C ++, но другие компании добавляют к нему другие языки программирования, такие как Lua, Javascript или MicroPhyton.

В этой книге будет использоваться NonOS SDK, а в качестве IDE разработки - Arduino IDE. В этой части вы загрузите программу Arduino (IDE), настроите его и установите ESP8266 SDK

Загрузка программы IDE Arduino

Чтобы загрузить IDE Arduino, перейдите по адресу https://www.arduino.cc/en/Main/Software и загрузите последнюю версию:

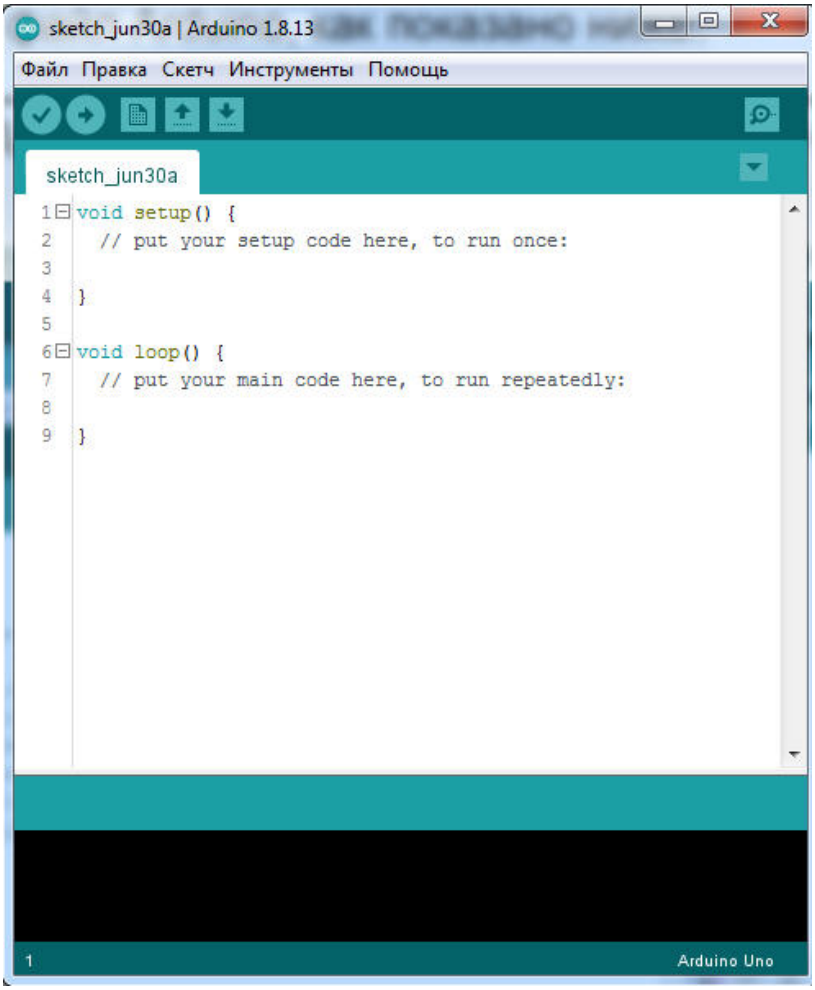


Теперь вы можете загрузить нужную версию в зависимости от вашей операционной системы. Существуют версии для Windows с правами администратора и без них, macOS X и Linux для 32-битных, 64-битных или процессоров ARM, так что вы можете установить и работать даже на Raspberry Pi.

После того, как вы загрузили IDE Arduino, вам необходимо установить ее на свой локальный компьютер.

Если вы работаете в Linux, вам нужно использовать команду xz и tar, чтобы открыть Arduino IDEarchive, затем вы можете войти в Arduino-1.8.2 и запустить его с помощью sudo ./arduino.

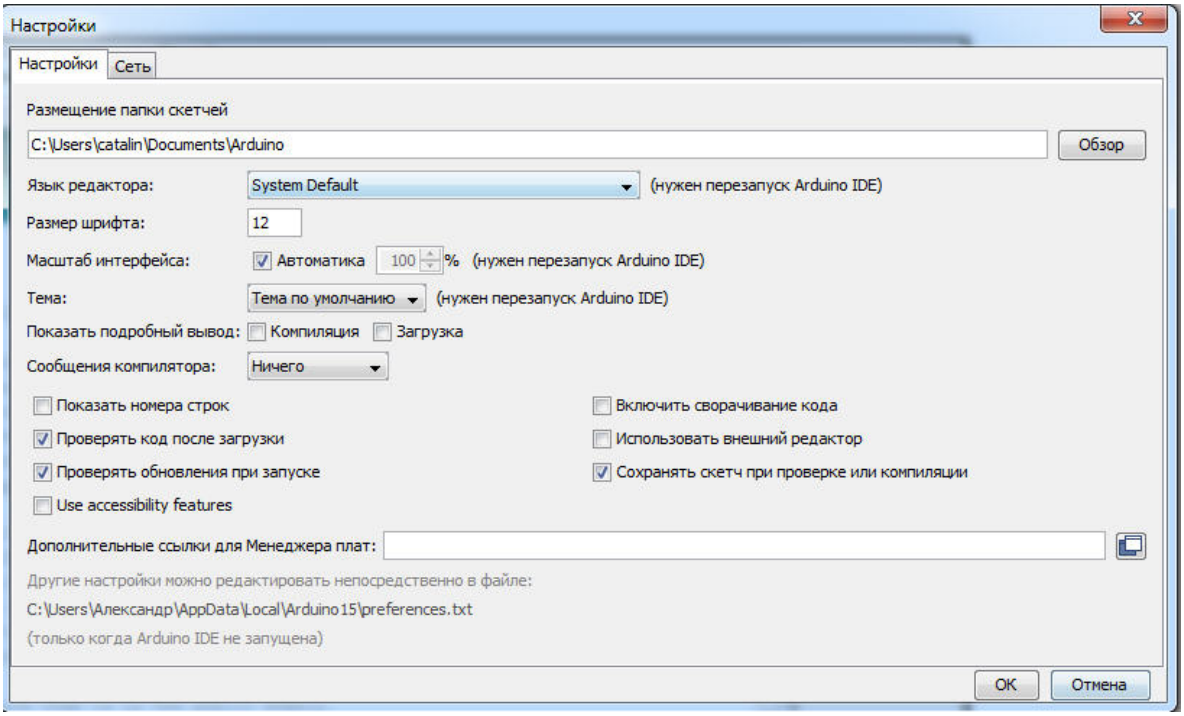
В Windows просто запустите исполняемый файл Arduino, как показано ниже:



Поздравляю! Вы установили IDE Arduino, и теперь пора настроить ее для ESP8266.

Настройка Arduino IDE

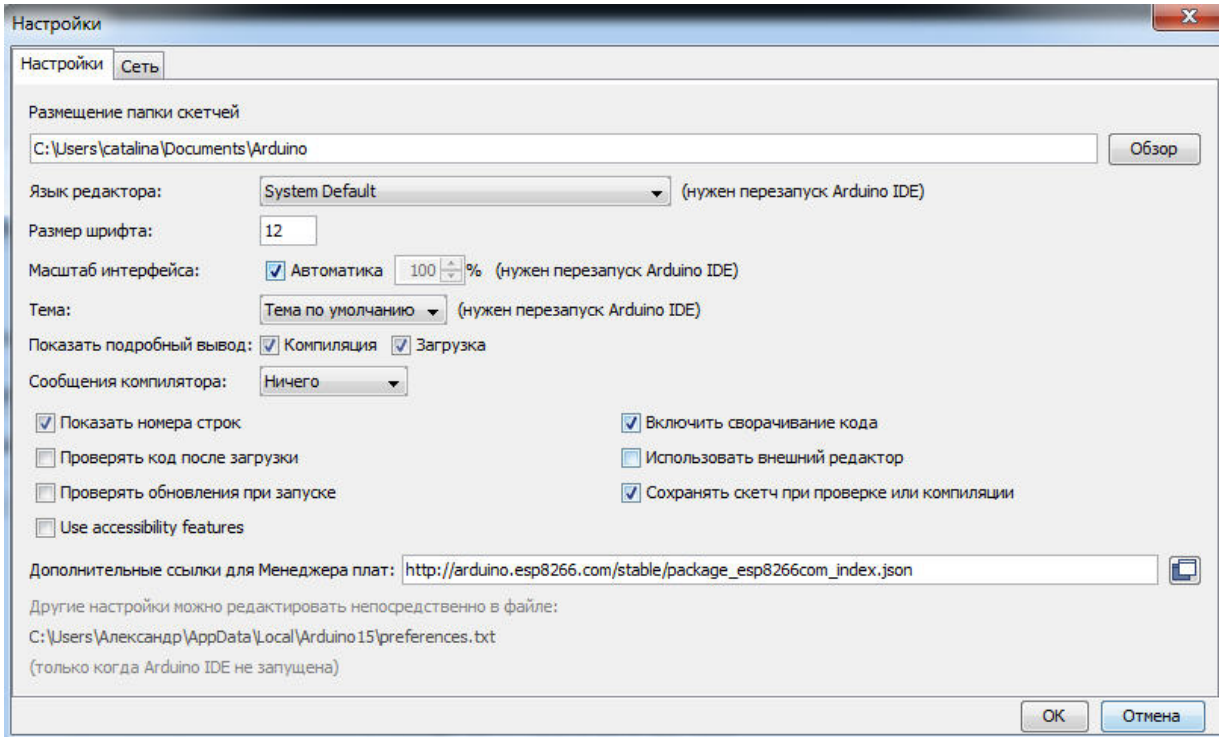
Чтобы настроить IDE Arduino для ESP8266, вам нужно перейти в Файл | Настройки. Начальный экран выглядит следующим образом:



На этом экране вы можете настроить некоторые поля следующим образом:

- **Размещение папки скетчей**: здесь вы можете выбрать, где будут храниться файлы вашего проекта.
- **Язык редактора**: выбор языка; но после этого вам нужно перезапустить IDE.
- **Размер шрифта**: это размер шрифта, используемый в среде IDE.
- **Показать подробный вывод** во время: **компиляции** и **загрузки**: я предлагаю вам проверить оба из них, чтобы получить подробный вывод во время компиляции, где вы можете увидеть файлы и их пути, а также загрузку.
- **Показать номера строк**: в этом поле удобно видеть номер строки в правой части номера IDE.

- **Включить сворачивание кода**: это поле даст вам больше места на экране.
- **Дополнительные ссылки для менеджера плат**: вот поле, которое позволит нам получить и установить компилятор ESP8266 Xtensagcc, необходимые инструменты для прошивки во флеш-память ESP8266 вместе с другими типами плат. В это поле нужно добавить http://arduino.esp8266.com/stable/package_esp8266com_index.json и экран настроек будет выглядеть следующим образом:



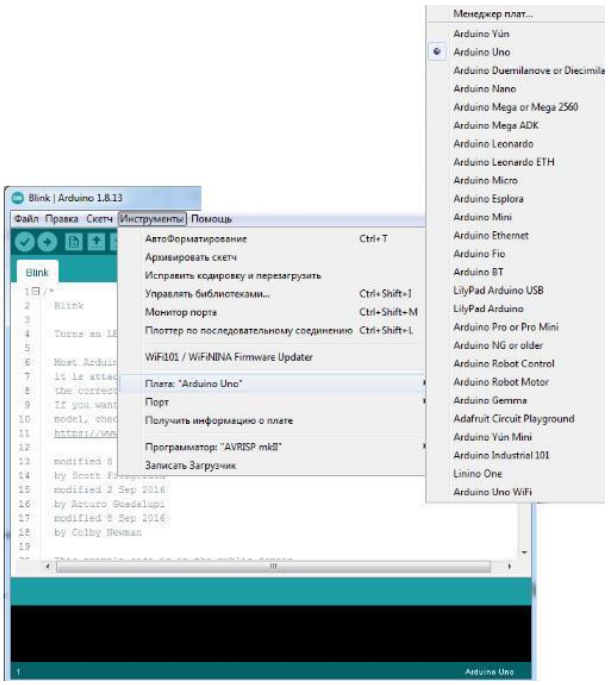
После настройки вы можете нажать кнопку **OK**. Не забывайте, что вы находитесь на прокси-сервере; вам необходимо заполнить необходимые данные на вкладке «**Сеть**».

Установка ESP8266 SDK

После настройки пришло время фактически получить ESP8266 SDK и инструменты.

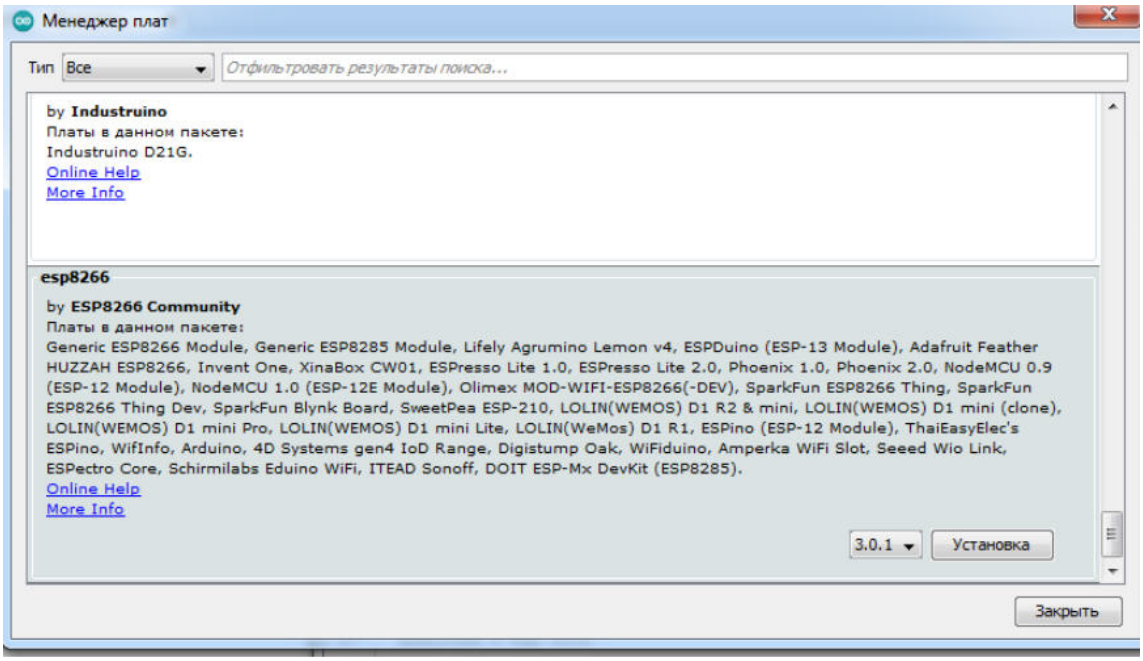
Для этого вам необходимо выполнить следующие действия:

1. Перейдите в **Инструменты** | **Плата**: «Arduino Uno» | **Менеджер плат** ...:



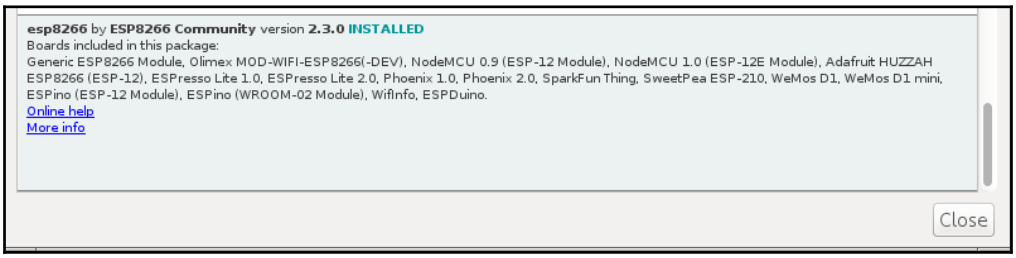
3. Перейдите в конец списка, выберите плату ESP8266 и нажмите «Установить»:

3. Перейдите в конец списка, выберите плату ESP8266 и нажмите «**Установка**»:

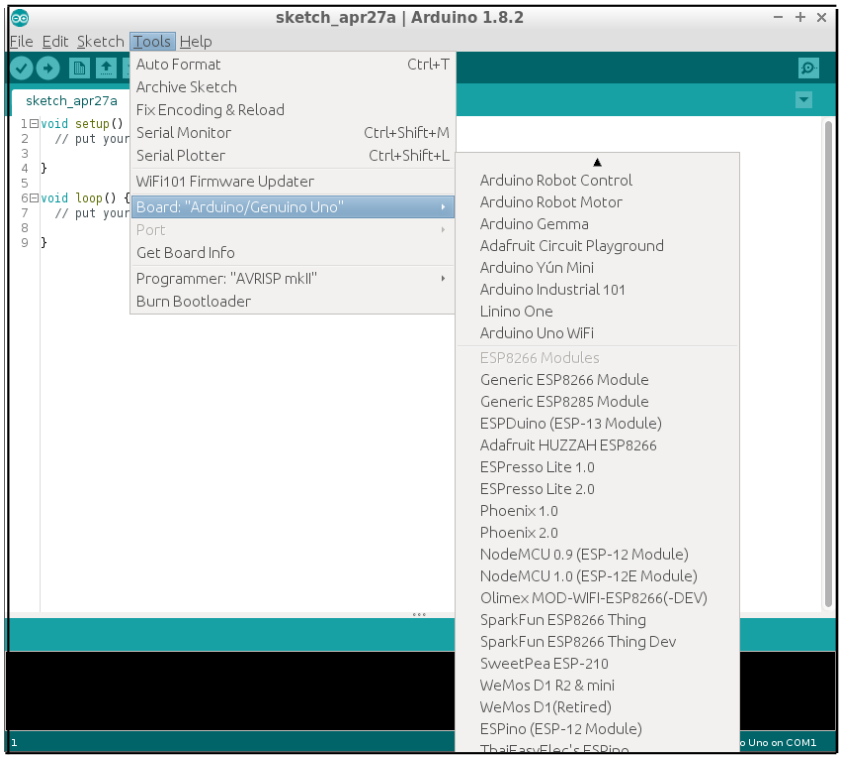


4. Теперь, в зависимости от вашего интернет-соединения, это может занять некоторое время. Сядьте поудобнее и расслабьтесь, вы собираетесь войти в мир Интернета вещей.

5. Найдите сообщение INSTALLED - **УСТАНОВЛЕНО**, как на этой картинке:



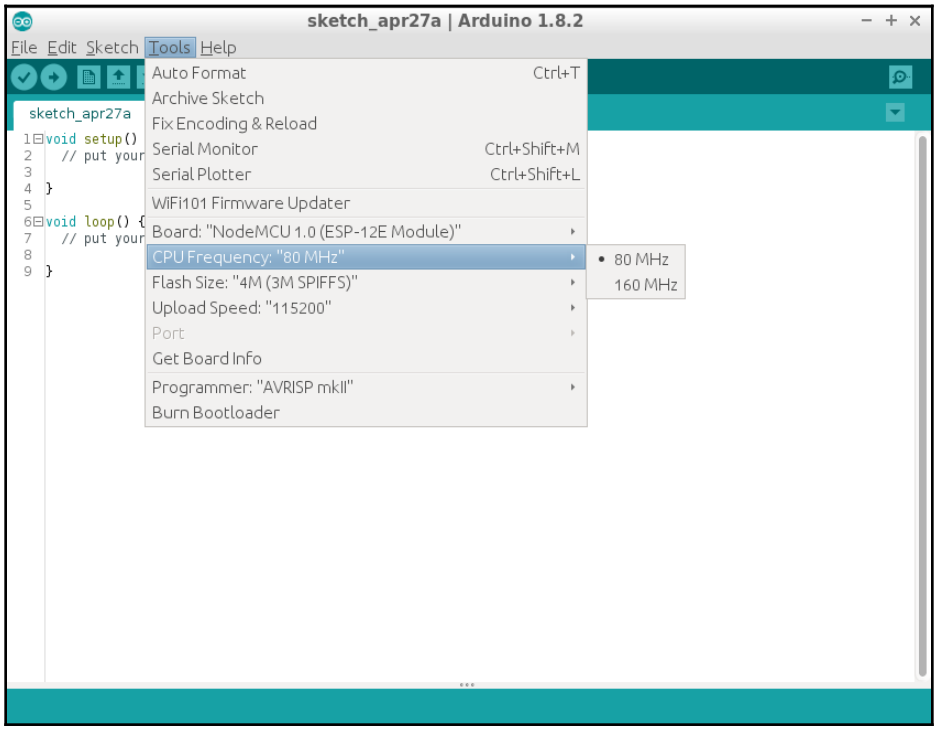
6. Теперь вернитесь в **Инструменты** | **Плата**: «Arduino / Uno», и вы должны увидеть множество плат на базе ESP8266 в нижней части:



7. Выберите NodeMCU 1.0 (ESP-12E Module) и вернитесь в меню «Инструменты», где вы увидите другие конфигурации, которые можно выполнить для ESP8266:

- **Частота процессора**: «80 **МГц**»: может быть изменена с 80 **МГц** на 160 **МГц**.
- Объем памяти вашего модуля и размер SPIFFS (1 или 3 MB)

- Скорость загрузки интерфейса UART между вашим компьютером и модулем ESP8266.
  Выберите **скорость загрузки**: «115200», чтобы двоичный файл прошивался в 8 раз быстрее:



8. Если у вас есть модуль, вы можете подключить его к компьютеру. Вернувшись снова в меню «**Инструменты**», вы увидите это в меню «**Порт**» и сможете выбрать свой последовательный интерфейс, подключенный к модулю ESP8266. Для Linux вы можете выбрать / dev / ttyUSB0, а для Windows - один из ваших COM-портов.

Перед запуском любой программы давайте рассмотрим кнопки IDE:

Начнем слева направо:

А Проверить: позволяет скомпилировать и проверить код на наличие ошибок.

Б Загрузка: выполняет то же действие, что и Проверить, плюс загружает сгенерированную прошивку во флэш-память ESP8266.

В Новый : открывает новое окно для создания новой программы

Г Открыть: открывает существующую программу с вашего локального диска

Д Сохранить: ваши файлы будут сохранены на диске

Е Serial Monitor: откроется окно, в котором вы сможете увидеть и отладить то, что вы добавляете в свою программу.

Как установить библиотеку

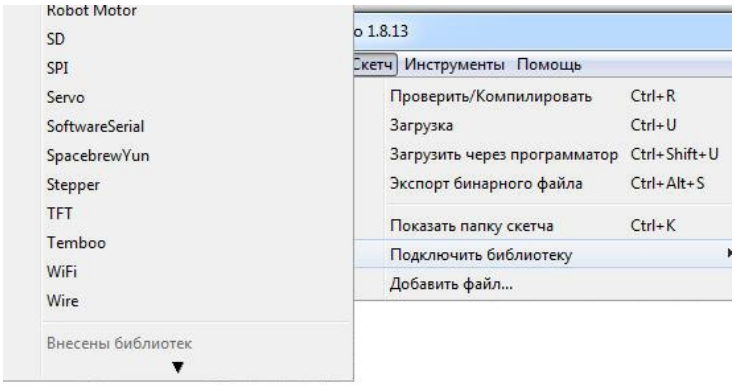
В некоторых главах этой книги нужны разные библиотеки; итак, давайте посмотрим, как можно установить библиотеку.

В установленном SDK есть несколько базовых библиотек, но вам наверняка понадобится больше библиотек для чтения различных датчиков, для анализа данных JSON или для отправки сообщений MQTT. Для библиотек, на которые есть ссылки в репозитории SDK, вам просто нужно установить их, а для тех, которые нет, вам нужно установить их вручную.

Библиотеки из репозитория Ардуино

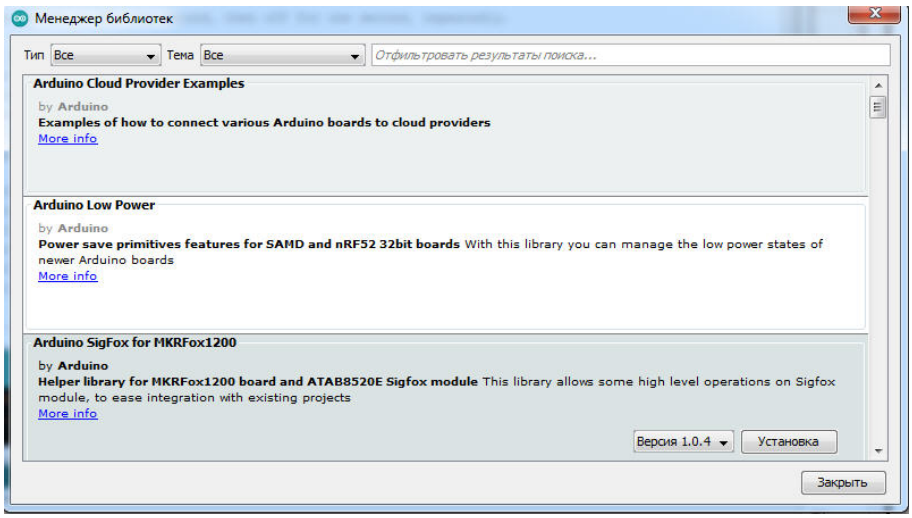
Некоторые библиотеки существуют в официальном репозитории, и вы можете установить их, выполнив следующие действия:

1. Если он существует, просто зайдите в скетч | Подключить библиотеку | Управлять библиотеками:

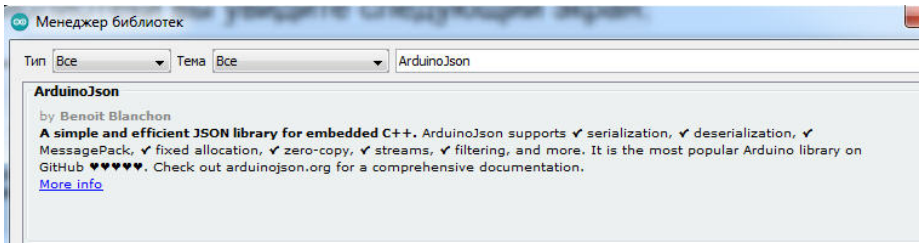


2. В новом окне удалите «отфильтровать результаты поиска ...» и напишите Arduino Json.

Затем IDE Arduino выполнит поиск в этой библиотеке для вас, и если она ее обнаружит, вы сможете установить ее, щелкнув по ней. Вы также можете использовать это меню для обновления ранее установленной библиотеки или для переключения между версиями библиотеки:



3. После установки библиотеки вы увидите следующий экран:



В качестве упражнения проделайте то же самое с библиотеками WiFiManager иPubSubClient.

Библиотеки нет в репозитории

Иногда нужная библиотека может не существовать в официальном репозитории, но вы можете найти ее на http://github.com в виде ZIP-архива.

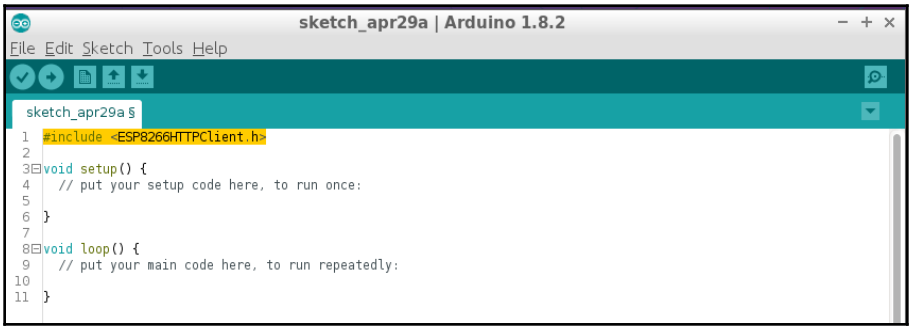
Для установки библиотеки в этом случае необходимо выполнить следующие действия:

1. Загрузите ZIP-файл и установите его вручную. Для этого перейдите в **Скетч | Подключить библиотеку | Добавить** .ZIP **библиотеку** ..., выберите загруженную библиотеку с вашего диска и нажмите Open.

2. Включите существующую библиотеку.

3. Чтобы включить существующую библиотеку, перейдите в **Скетч | подключить библиотеку** и выберите библиотеку, которую хотите включить в свой эскиз.

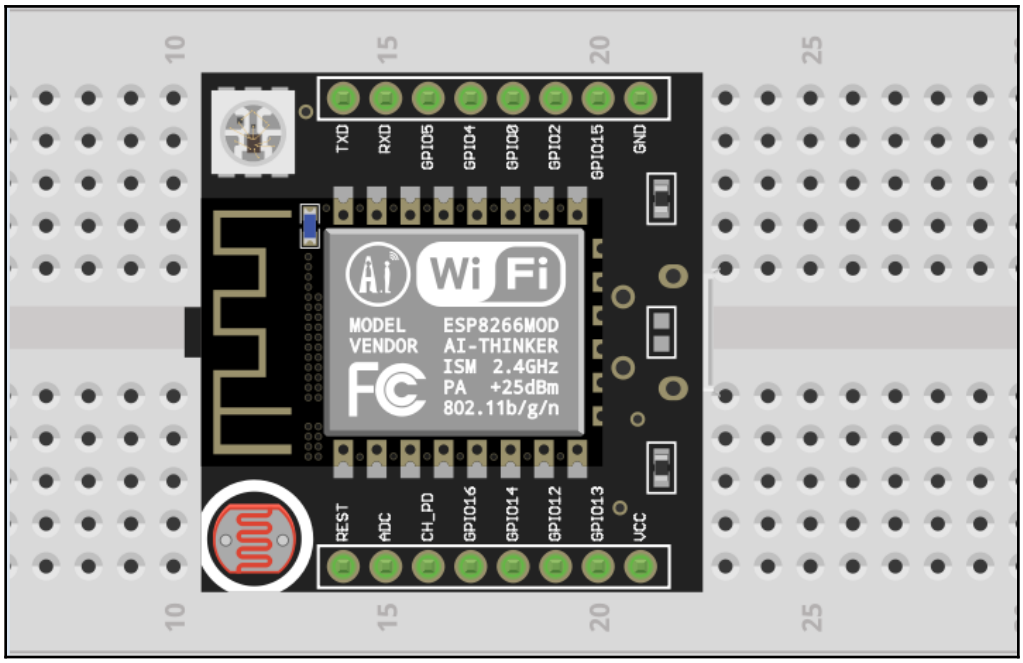
3. Файл .h будет добавлен в ваш скетч, и теперь у вас есть доступ к функции в этой библиотеке, чтобы использовать их в своей собственной программе:



Пришло время для вашей первой программы

Для начала давайте оценим основные входные и выходные данные модуля Witty ESP8266.

Определение выводов - это светозависимый резистор (LDR) на Witty, он подключен к A0 (аналоговый вход), кнопка подключена к GPIO 4, а светодиоды подключены к GPIO 12, GPIO 13 и GPIO 15:



Удалите все, что есть в вашей Arduino IDE, и замените его следующим кодом:

```
#define LDR     A0
#define BUTTON  4
#define RED     15
#define GREEN   12
#define BLUE    13
```

Раздел настройки будет запущен только один раз после перезагрузки модуля или включения питания. Последовательный UART запускается со скоростью 115200 бит /с, поэтому сообщения можно увидеть в окне Serial Monitor, где вам также необходимо установить такую же скорость в правом нижнем углу окна;в противном случае будут видны странные персонажи.

Все пины определяются как INPUT - ВХОД или OUTPUT - ВЫХОД в зависимостиот их использования. Пины с подключенными кнопкой и LDR настраиваются на вход, а подключенный к светодиоду пин - выход.

```
void setup()
{
Serial.begin(115200);

pinMode(LDR, INPUT);
pinMode(BUTTON, INPUT);
pinMode(RED, OUTPUT);
pinMode(GREEN, OUTPUT);
pinMode(BLUE, OUTPUT);
}
```

Функция loop () постоянно выполняется после setup () и в ней:

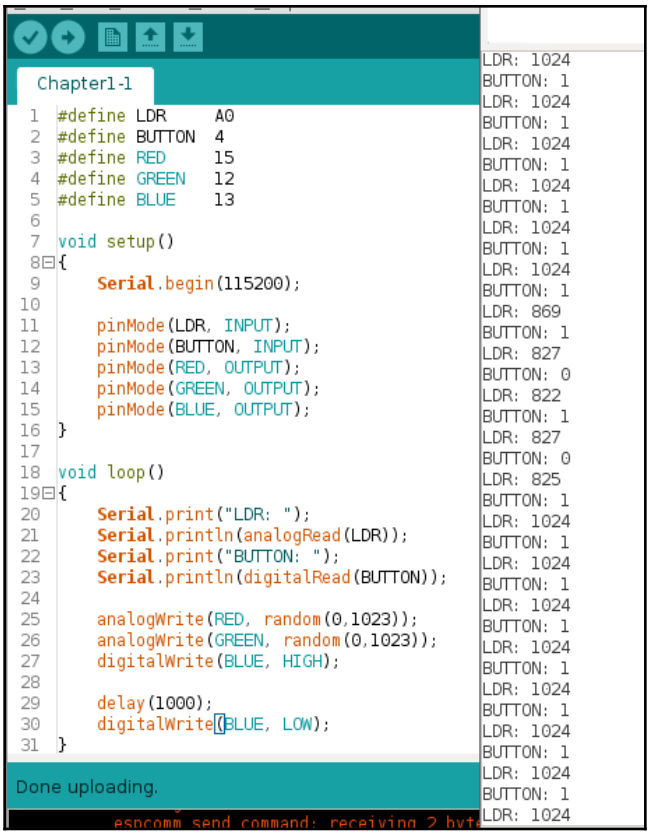
1. Функция analogRead считывает значение окружающего света, обеспечиваемое LDR в пределах 0–1 В.
2. Функция digitalRead считывает значение GPIO 4, которое может быть 0 В при нажатии кнопки, либо V_{CC} = 3,3 В, если кнопка не нажата.
3. Отобразите данные на Serial Monitor с помощью функции Serial.print. Serial.println просто добавляет новую строку.
4. Запишите случайное значение от 0 до 1023 в GPIO 15 и GPIO 12, которое будет управлять интенсивностью красного и зеленого цвета светодиода. Это **широтно-импульсная модуляция (ШИМ)**.
5. Включите синий светодиод, подключенный к GPIO 13.
6. Подождите 1000 мс (одну секунду).
7. Выключите синий светодиод и продолжайте с шага 1:

```
void loop()
{
Serial.print("LDR: ");
Serial.println(analogRead(LDR));
Serial.print("BUTTON: ");
Serial.println(digitalRead(BUTTON));

analogWrite(RED,   random(0,1023));
analogWrite(GREEN, random(0,1023));
digitalWrite(BLUE, HIGH);
delay(1000);
digitalWrite(BLUE, LOW);
}
```

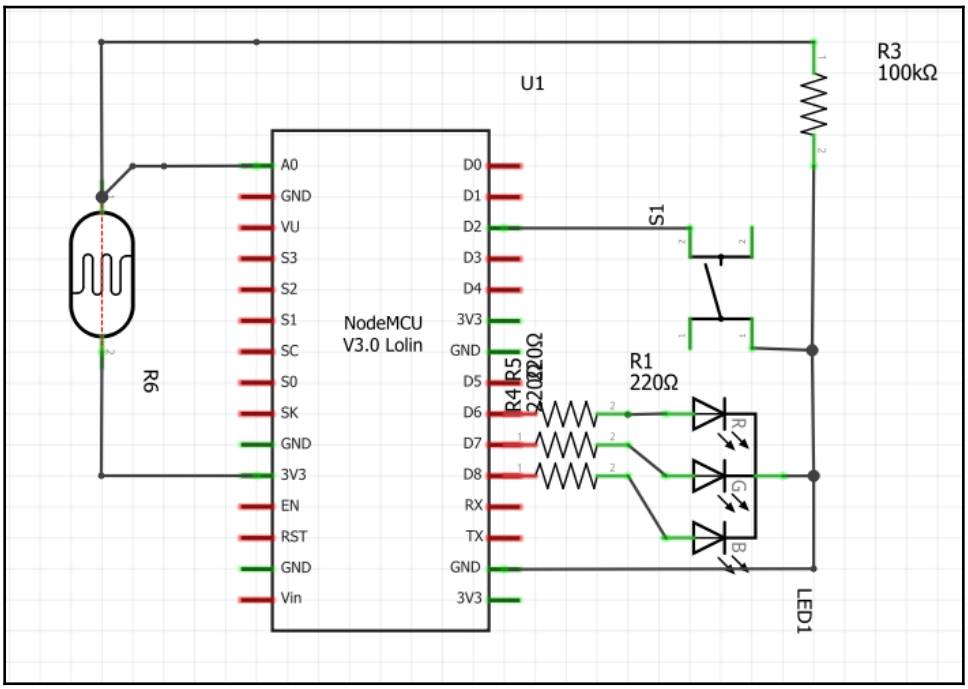
Чтобы скомпилировать и сбросить двоичный файл в микросхему ESP8266, вам необходимо нажать кнопку «Upload» (Загрузить).

Видя результат

В выводе Serial Monitor, как показано на следующем изображении, мы можем видеть значения внешнего освещения и состояние кнопки, где 0 означает нажатие, а1 - не нажатие:



Если у вас нет модуля Witty, вам понадобятся некоторые дополнительные детали, такие как резисторы, светодиоды, кнопки и датчики LDR (фоторезисторы), согласно следующей схеме:



Давайте теперь рассмотрим функции, которые позволяют вам управлять пинами GPIO, и функцию, которая будет печатать значения в последовательном мониторе:

- analogRead (pin): считывает значение на пине A0.
- digitalRead (pin): считывает значение для указанного пина, LOW или HIGH
- digitalWrite (pin, value): записывает LOW или HIGH значение на цифровой пин.
- Serial.println (val): выводит данные на последовательный порт в виде удобочитаемых символов ASCII, оканчивающихся на \ r, и символа новой строки \ n.

При использовании analogWrite (val), где val может находиться в интервале от 0 до 1023, на пине цифрового выхода ШИМ будет напряжение от 0 до 3,3 В с шагом 1023.

Подключение ESP8266 к Wi-Fi

Вы установили и настроили Arduino IDE для ESP8266 и узнали, как управлять светодиодом, считывать данные с аналогового входа и постепенно гасить светодиод.

Пришло время подключить ESP8266 к Wi-Fi. Включите библиотеку Wi-Fi ESP8266 и настройте имя SSID и пароль Wi-Fi:

```
#include <ESP8266WiFi.h>
const char* ssid     = "your_wifi_name";
const char* password = "your_wifi_password";
```

В разделе настройки Serial запускается и настраивается для отправки данных со скоростью 115200 бит / с; добавлена задержка 10 мс, чтобы позволить последовательному завершению, и GPIO от 12 до 15 настроены как выход, и их значение установлено на LOW:

```
void setup() {
Serial.begin(115200);
delay(10);
pinMode(12, OUTPUT);
pinMode(13, OUTPUT);
pinMode(14, OUTPUT);
pinMode(15, OUTPUT);

digitalWrite(12,LOW);
digitalWrite(13,LOW);
digitalWrite(14,LOW);
digitalWrite(15,LOW);
```

Начнем с подключения к сети Wi-Fi:

```
Serial.println();
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.begin(ssid, password);
```

Ждем, пока в статусе не будет указано, что ESP8266 подключен к сети Wi-Fi. После этого отображается сообщение о подключении Wi-Fi вместе с IP-адресом, назначенным ему роутером. Ваш роутер должен поддерживать DHCP и иметь включенную функцию DHCP:

```
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
  }

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}
```

В разделе цикла код проверяет, подключен ли чип к Wi-Fi, и если это правда, на модуле Witty загорится зеленый светодиод:

```
void loop()
{
if(WiFi.status() == WL_CONNECTED)
digitalWrite(12, HIGH);
}
```

В качестве упражнения вы можете зажечь КРАСНЫЙ светодиод, если нет подключения к вашему роутеру, и зеленый светодиод в противном случае.

Serial Monitor покажет IP-адрес, назначенный роутером, как показано ниже:

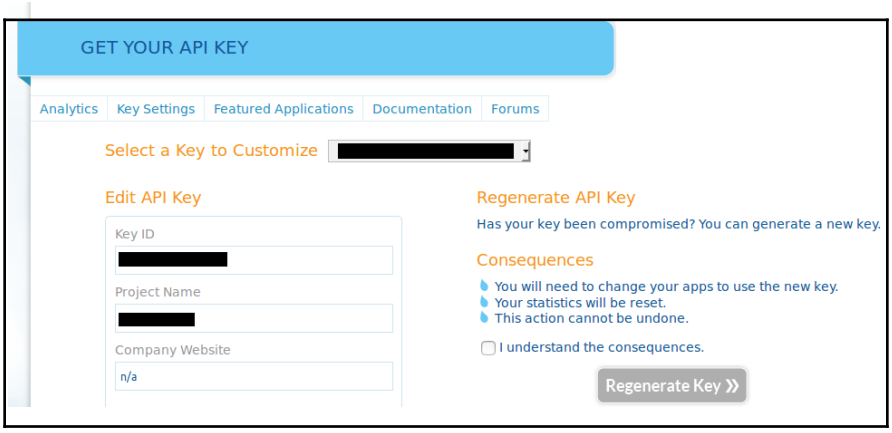
Получение данных из Интернета

Теперь, когда мы подключили ESP8266 к сети Wi-Fi, мы можем получать и отправлять данные через Интернет. Более того, мы можем считывать данные со входа или с датчиков, прикрепленных к плате, и отправлять их значения в Интернет.

Для начала давайте прочитаем некоторые данные, а что может быть интереснее текущих данных о погоде? Создайте учетную запись на http://www.wunderground.com, а затем перейдите на https://www.wunderground.com/weather/api/d/pricing.htm, где вы купите ключ за 0 долларов, как показано на следующем изображении. После заполнения данных о проекте у вас будет ключ:

Как видите, с ключом разработчика у вас есть 10 ограниченных вызовов в минуту, что означает, что вы можете получать данные каждые 6 секунд. Позже в коде мы будем получать данные каждые 10 секунд.

Чтобы проверить свой API_KEY, используйте его в браузере и убедитесь, что вы получаете какие-либо данные.
Замените APY_KEY своим собственным ключом:



После этого, если вы перейдете по этой ссылке в своем браузере, http://api.wunderground.com/api/APY_KEY/conditions/q/NL/Eindhoven.json; вы получите следующий ответ в формате JSON от сервера wunderground.com:

Включите библиотеку ESP8266WiFi и библиотеку ESP8266HTTPClient, которая позволит вам выполнить действие HTTP GET, чтобы получить такое же сообщение в формате JSON, как и в браузере:

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
```

Объявите SSID и пароль своей сети Wi-Fi:

```
const char* ssid     = "Your_WiFi_Name";
const char* password = "Your_WiFi_Password";

const String WUNDERGROUND_API_KEY = "YOUR_Wunderground_API_KEY";
const String WUNDERGROUND_COUNTRY = "NL";
const String WUNDERGROUND_CITY = "Eindhoven";
```

Создайте URL-адрес, который будет использоваться для получения данных:

```
const String dataURL =
"http://api.wunderground.com/api/"+WUNDERGROUND_API_KEY+"/conditions/q/"+WU
NDERGROUND_COUNTRY+"/"+WUNDERGROUND_CITY+".json";
```

Как обычно, в разделе настройки подключимся к сети Wi-Fi:

```
void setup() {
Serial.begin(115200);
delay(10);
Serial.println();
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
  }

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());

}
```

В цикле, если статус Wi-Fi подключен, вы создадите экземпляр объекта HTTPClient с именем http и начнете получать данные каждые 10 секунд из ранее созданной ссылки. В переменной полезной нагрузки у вас будет полный ответ от сервера:

```
void loop()
{
if(WiFi.status() == WL_CONNECTED)
  {
HTTPClient http;
http.begin(dataURL);
inthttpCode = http.GET();

if(httpCode> 0) {
// HTTP-заголовок был отправлен, и заголовок ответа сервера обработан

Serial.printf("[HTTP] GET... code: %d\n", httpCode);

            // файл найден на сервере
```

```
if(httpCode == HTTP_CODE_OK) {
                String payload = http.getString();
Serial.println(payload);
            }
        }
  }
delay(10000);
}
```

Если получение данных каждые 10 секунд - это слишком часто, давайте изменим его на раз в минуту, заменив вызов delay (10000), который блокирует выполнение другого кода.

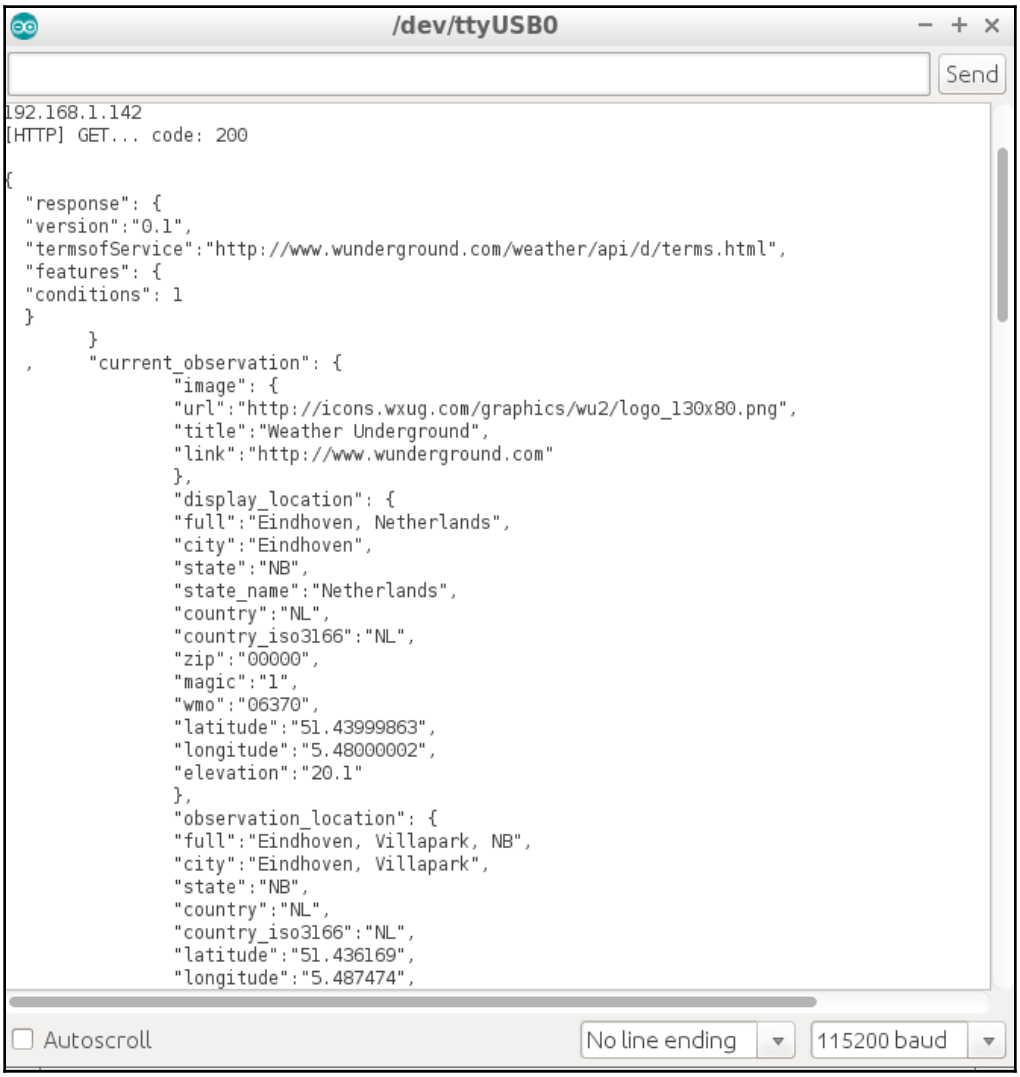
Итак, после const String WUNDERGROUND_CITY = "Eindhoven"; добавьте две строки кода:

```
const long interval = 60 * 1000;
unsigned long previousMillis = 0;
```

Теперь функция loop изменится следующим образом:

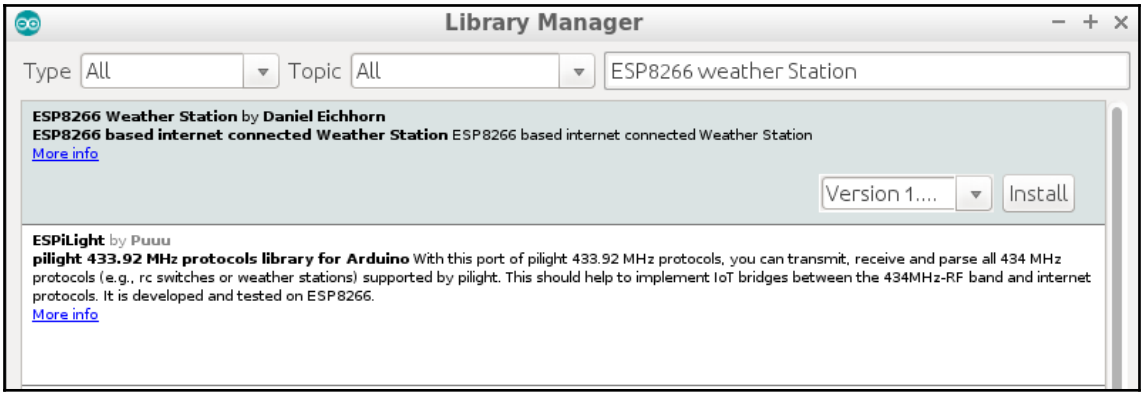
```
void loop()
{
unsigned long currentMillis = millis();
if(currentMillis - previousMillis >= interval)
  {
previousMillis = currentMillis;
if(WiFi.status() == WL_CONNECTED)
    {
HTTPClient http;
http.begin(dataURL);
inthttpCode = http.GET();

if(httpCode > 0) {
   // HTTP-заголовок был отправлен, и заголовок ответа сервера обработан

Serial.printf("[HTTP] GET... code: %d\n", httpCode);

            // файл найден на сервере
if(httpCode == HTTP_CODE_OK) {
                String payload = http.getString();
Serial.println(payload);
            }
        }
    }
  }
}
```

Теперь Serial Monitor каждую минуту будет отображать огромный JSON со всей информацией о погоде, от температуры до влажности, скорости ветра, точки росы и многого другого:

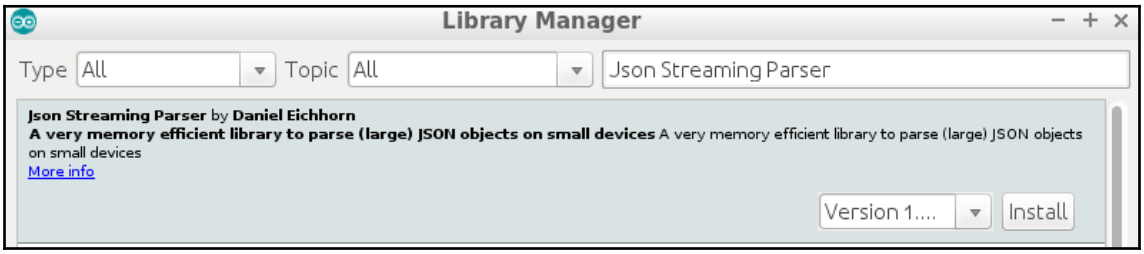


Но что, если вы хотите получить из этого JSON только определенные данные? К счастью, для этого есть библиотека Wunderground. Чтобы установить ее, перейдитев Sketch | Включить библиотеку | ManageLibraries и найдите метеостанцию ESP8266. После установки этой библиотеки вы также должны установить библиотеку JsonStraming Parser, которая будет анализировать полученный JSON. Вы можете выполнить следующие действия:

1. Установите библиотеку метеостанции - Weather Station - ESP8266:



2. Также установите библиотеку JSON Streaming Parser:



Теперь давайте получим те же данные, поэтому будет использоваться тот же API_KEY, но данные анализируются библиотечными функциями:

1. Включите файлы заголовков для ESP8266 Wi-Fi.h, JSONListener.h и WundergroundClient:

```
#include <ESP8266WiFi.h>
#include <JsonListener.h>
#include "WundergroundClient.h"
```

2. Определите API_KEY и установите метрическую логическую переменную:

```
const String  WUNDERGRROUND_API_KEY = "YOUR_API_KEY";
constboolean IS_METRIC = true;
```

3. Инициализируйте WundergoundClient для метрической системы:

```
WundergroundClientweather_data(IS_METRIC);
```

4. Также инициализируйте настройки и константы Wi-Fi, используемые для получения данных о погоде: 4.

```
const char* WIFI_SSID     = "YOUR_WIFI_SSID";
const char* WIFI_PASSWORD = "YOUR_WIFI_PASSWORD";
const String WUNDERGROUND_LANGUAGE = "EN";
const String WUNDERGROUND_COUNTRY = "NL";
const String WUNDERGROUND_CITY = "Eindhoven";
WiFiClientwifiClient;
```

5. Инициализируйте функцию настройки для подключения к сети Wi-Fi:

```
void setup() {
Serial.begin(115200);
delay(10);

WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
delay(20);
Serial.print("Connecting to ");
Serial.println(WIFI_SSID);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
  }
Serial.println("");
Serial.println("WiFi connected!");
Serial.println();
}
```

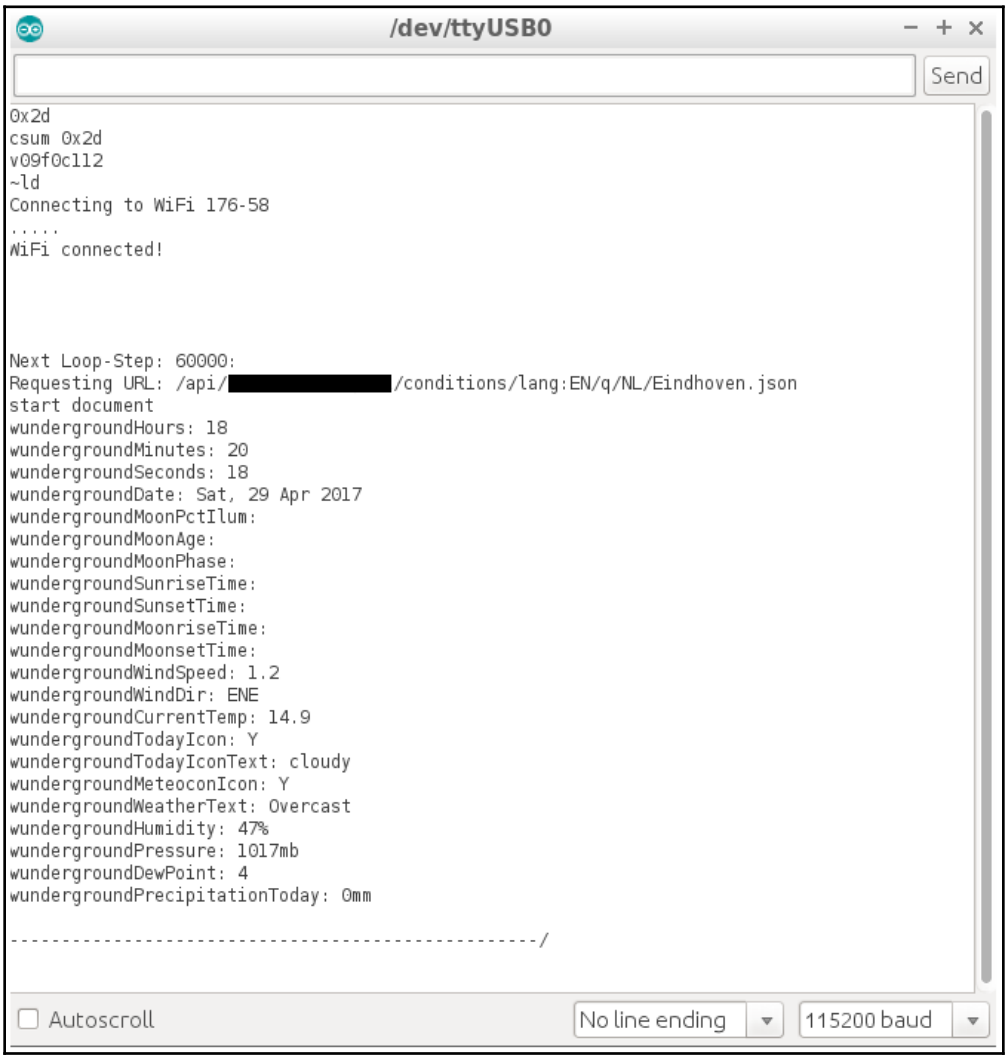
6. В функции цикла получать данные с сайта wunderground.com каждую минуту и отображать их в окне Serial Monitor:

```
void loop() {

if ((millis() % (60 * 1000)) == 0) {
Serial.println();
Serial.println("\n\nNext Loop-Step: " + String(millis()) + ":");

weather_data.updateConditions(WUNDERGRROUND_API_KEY,
WUNDERGROUND_LANGUAGE, WUNDERGROUND_COUNTRY, WUNDERGROUND_CITY);
```

```
Serial.println("wundergroundHours: " + weather_data.getHours());
Serial.println("wundergroundMinutes: " +
weather_data.getMinutes());
Serial.println("wundergroundSeconds: " +
weather_data.getSeconds());
Serial.println("wundergroundDate: " + weather_data.getDate());

Serial.println("wundergroundMoonPctIlum: " +
weather_data.getMoonPctIlum());
Serial.println("wundergroundMoonAge: " +
weather_data.getMoonAge());
Serial.println("wundergroundMoonPhase: " +
weather_data.getMoonPhase());
Serial.println("wundergroundSunriseTime: " +
weather_data.getSunriseTime());
Serial.println("wundergroundSunsetTime: " +
weather_data.getSunsetTime());
Serial.println("wundergroundMoonriseTime: " +
weather_data.getMoonriseTime());
Serial.println("wundergroundMoonsetTime: " +
weather_data.getMoonsetTime());
Serial.println("wundergroundWindSpeed: " +
weather_data.getWindSpeed());
Serial.println("wundergroundWindDir: " +
weather_data.getWindDir());

Serial.println("wundergroundCurrentTemp: " +
weather_data.getCurrentTemp());
Serial.println("wundergroundTodayIcon: " +
weather_data.getTodayIcon());
Serial.println("wundergroundTodayIconText: " +
weather_data.getTodayIconText());
Serial.println("wundergroundMeteoconIcon: " +
weather_data.getMeteoconIcon(weather_data.getTodayIconText()));
Serial.println("wundergroundWeatherText: " +
weather_data.getWeatherText());
Serial.println("wundergroundHumidity: " +
weather_data.getHumidity());
Serial.println("wundergroundPressure: " +
weather_data.getPressure());
Serial.println("wundergroundDewPoint: " +
weather_data.getDewPoint());
Serial.println("wundergroundPrecipitationToday: " +
weather_data.getPrecipitationToday());

Serial.println();
Serial.println("-------------------------------------------------
/\n");
```

```
        }
    }
```

7. Выходные данные для последовательного монитора выглядят следующим образом:



Теперь в качестве упражнения вы можете считывать температуру и включать или выключать светодиод, если есть условия обледенения или влажности, а температура на улице слишком высока.

Отправка данных в Интернет

Теперь давайте отправим те же данные в Интернет. Первое, что нужно сделать, это создать учетную запись на http://thingspeak.com и создать канал. Каждый канал имеет восемь полей, которые вы можете использовать для хранения данных, передаваемых ESP8266.

В качестве бесплатной учетной записи вам не нужно отправлять данные чаще трех раз в минуту. Преимущество в том, что ваши данные хранятся на их сервере, и вы можете увидеть их на красивом графике или встроить их как IFRAME на другой веб-сервер.

В Channel Settings (настройках канала) создайте одно поле и назовите его Light, затем перейдите на вкладку API-ключ и получите Write API KEY. Здесь вы также можете определить APY KEY для чтения, если у вас есть приложение, которое хочет читать данные, записанные другими модулями. Это элементарный способ обмена данными между модулями.

Поскольку модуль Witty имеет LDR, давайте будем использовать его для записи данных каждую минуту на api.thingspeak.com:

```
#include <ESP8266WiFi.h>

const char* WIFI_SSID     = "YOUR_WIFI_SSID";
const char* WIFI_PASSWORD = "YOUR WIFI_PASSWORD";
const char* host = "api.thingspeak.com";
const char* writeAPIKey = "YOUR_WRITE_API_KEY";
#define LDR     A0
```

В функции setup (), которая выполняется один раз, вывод LDR устанавливается как вывод INPUT и будет подключать ESP8266 к сети Wi-Fi с помощью функции WiFi.begin (WIFI_SSID, WIFI_PASSWORD):

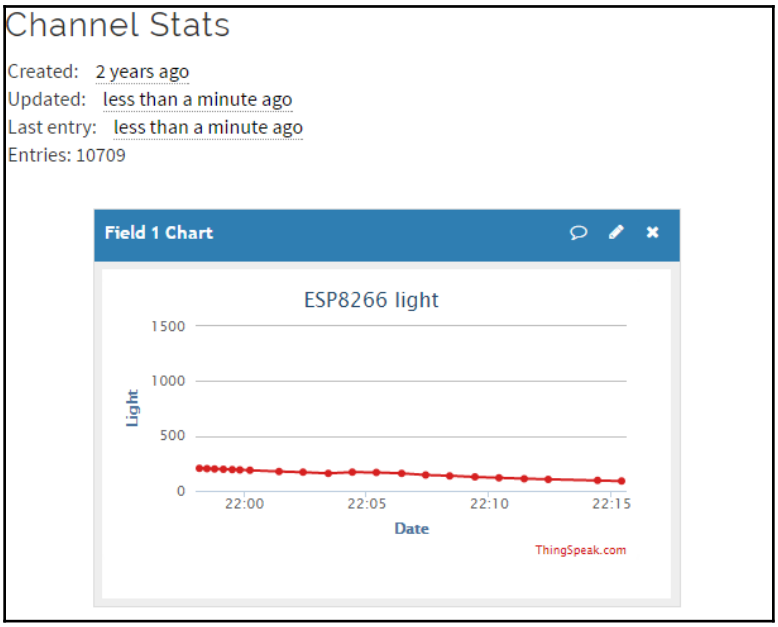
```
void setup()
{
Serial.begin(115200);
delay(10);
pinMode(LDR, INPUT);
WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
delay(20);
Serial.print("Connecting to ");
```

```
Serial.println(WIFI_SSID);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
  }
Serial.println("");
Serial.println("WiFi connected!");
Serial.println();
}
```

В функции loop () каждую минуту интенсивность света будет считываться с датчика LDR и размещаться в поле Light на канале https: / / thingspeak. Com /:

```
void loop()
{
if ((millis() % (60 * 1000)) == 0) {
// устанавливаем TCP-соединения
WiFiClient client;
const  int httpPort = 80;
if (!client.connect(host, httpPort)) {
return;
  }

  String url = "/update?key=";
url+=writeAPIKey;
url+="&field1=";
url+=String(analogRead(LDR));
url+="\r\n";
Serial.println(url);
  // Запрос к серверу
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  }

}
```

Посмотрим, как выглядят данные через несколько минут:



Теперь давайте объединим скетчи, которые считывают погоду с wunderground.com, и тот, который отправляет данные на thingspeak.com. Он будет измерять температуру, влажность, точку росы и количество осадков и сохранять их на thinkspeak.com, чтобы позже мы могли их импортировать.
По сути, это будет регистратор погоды:

```
#include <ESP8266WiFi.h>#include
<JsonListener.h>#include "
WundergroundClient.h"
```

Ниже приведены настройки Wunderground:

```
const String  WUNDERGRROUND_API_KEY = "58dfbeb30d02af26";
const Boolean IS_METRIC = true;
WundergroundClient weather_data(IS_METRIC);
const char* WIFI_SSID     = "YOUR_WIFI_SSID";
const char* WIFI_PASSWORD = "YOUR_WIFI_PASSWORD";
const String WUNDERGROUND_LANGUAGE = "EN";
const String WUNDERGROUND_COUNTRY = "NL";
const String WUNDERGROUND_CITY = "Eindhoven";
const char* host = "api.thingspeak.com";
const char* writeAPIKey = "YOUR_WRITE_API_KEY";
WiFiClient wifiClient
```

Ниже приведена функция setup () для подключения к сети Wi-Fi:

```
void setup() {

Serial.begin(115200);
delay(10);

WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
delay(20);
Serial.print("Connecting to ");
Serial.println(WIFI_SSID);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
  }
Serial.println("");
Serial.println("WiFi connected!");
Serial.println();
}
```

В функции loop () ежеминутные данные о погоде будут извлекаться с сайта wunderground.com и публиковаться на сайте thingspeak.com. Наряду с температурой, давлением, осадками и точкой росы в последовательном выводе будет напечатана дополнительная информация, например, фаза луны,восход или закат, информация, которую можно использовать, если вы планируете добавить модуль отображения для визуализации всех погодных условий:

```
void loop() {
if ((millis() % (60 * 1000)) == 0) {
Serial.println();
Serial.println("\n\nNext Loop-Step: " + String(millis()) + ":");

weather_data.updateConditions(WUNDERGRROUND_API_KEY, WUNDERGROUND_LANGUAGE,
WUNDERGROUND_COUNTRY, WUNDERGROUND_CITY);

Serial.println("wundergroundHours: " + weather_data.getHours());
Serial.println("wundergroundMinutes: " + weather_data.getMinutes());
Serial.println("wundergroundSeconds: " + weather_data.getSeconds());
Serial.println("wundergroundDate: " + weather_data.getDate());

Serial.println("wundergroundMoonPctIlum: " +
weather_data.getMoonPctIlum());
Serial.println("wundergroundMoonAge: " + weather_data.getMoonAge());
Serial.println("wundergroundMoonPhase: " + weather_data.getMoonPhase());
Serial.println("wundergroundSunriseTime: " +
weather_data.getSunriseTime());
Serial.println("wundergroundSunsetTime: " + weather_data.getSunsetTime());
Serial.println("wundergroundMoonriseTime: " +
```

```
weather_data.getMoonriseTime());
Serial.println("wundergroundMoonsetTime: " +
weather_data.getMoonsetTime());
Serial.println("wundergroundWindSpeed: " + weather_data.getWindSpeed());
Serial.println("wundergroundWindDir: " + weather_data.getWindDir());

Serial.println("wundergroundCurrentTemp: " +
weather_data.getCurrentTemp());
Serial.println("wundergroundTodayIcon: " + weather_data.getTodayIcon());
Serial.println("wundergroundTodayIconText: " +
weather_data.getTodayIconText());
Serial.println("wundergroundMeteoconIcon: " +
weather_data.getMeteoconIcon(weather_data.getTodayIconText()));
Serial.println("wundergroundWeatherText: " +
weather_data.getWeatherText());
Serial.println("wundergroundHumidity: " + weather_data.getHumidity());
Serial.println("wundergroundPressure: " + weather_data.getPressure());
Serial.println("wundergroundDewPoint: " + weather_data.getDewPoint());
Serial.println("wundergroundPrecipitationToday: " +
weather_data.getPrecipitationToday());

WiFiClient client;
const int httpPort = 80;
if (!client.connect(host, httpPort)) {
return;
    }

    String url = "/update?key=";
url+=writeAPIKey;
url+="&field1=";
url+=String(weather_data.getCurrentTemp());
url+="&field2=";
url+=String(weather_data.getPressure());
url+="&field3=";
url+=String(weather_data.getDewPoint());
url+="&field4=";
url+=String(weather_data.getPrecipitationToday());
url+="\r\n";
Serial.println(url);
    // Request to the server
client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");

Serial.println("-------------------------------------------------/\n");
  }
}
```

Через несколько минут вы можете увидеть значения, полученные ESP8266 с wunderground.com и опубликованные на сайте thingspeak.com, отображаемые на четырех графиках:

Резюме

Сделан первый шаг в разработке приложений Интернета вещей. Теперь у вас есть знания по установке и настройке Arduino IDE для разработки ESP8266, а также о том, как передавать и получать данные в Интернет и из Интернета.

Следующим шагом будет заставить модули ESP8266 взаимодействовать друг с другом, независимо от того, где они расположены.

2

Создание и настройка собственного сервера MQTT

Теперь, когда вы узнали, как использовать аналоговые и цифровые выходы, отправлять и получать данные через Wi-Fi, пора перейти к следующей части, где вы узнаете о протоколе MQTT и о том, как его использовать.

В предыдущей главе ESP8266 обменивался данными только с серверами: в этой главе мы увидим, как модули ESP8266 могут взаимодействовать друг с другом через транспортную телеметрию очереди сообщений - **Message Queue Telemetry Transport (MQTT).**

Message Queue Telemetry Transport (MQTT) - **Транспорт телеметрии очереди сообщений**

Обычно конечные устройства имеют ограниченную память и мощность процессора, работают от батарей, поэтому для их подключения к серверам требуется легкий протокол. Рассмотрим протокол MQTT, изобретенный в 1999 году Энди Стэнфорд-Кларком из IBM и Арленом Ниппером из Arcom, протокол SCADA, изначально предназначенный для устройств с батарейным питанием для контроля нефтепроводов. Позже в 2010 году IBM выпустила его как бесплатный протокол. В 2014 году OASIS объявил, что MQTT v.3.1.1 стал стандартом OASIS, и было разработано множество клиентов MQTT для всех языков программирования.

Характеристики MQTT перечислены ниже:

- Независимость от данных: MQTT может передавать все виды данных, от данных датчика до изображений или обновлений по беспроводной сети.
- Легкость и эффективная пропускная способность: наименьший кадр имеет длину всего 2 байта

- Provide QoS: три уровня качества обслуживания (QoS)
- Работает поверх стека TCP / IP
- Простота разработки: клиенты существуют для всех операционных систем и языков программирования.
- Central broker -Центральный брокер: может подключать разные устройства, не беспокоясь о совместимости
- Осведомленность о сеансе: обеспечивает подход на основе идентификационных данных для подписок.
- Гибкие темы подписки

Качество обслуживания

Стандарт MQTT определяет три уровня QoS:

- QoS 0: не более одного раза, сообщение с QoS 0 будет отправлено клиентом один раз и не будет сохранено и на него не будет ответа. Используйте этот QoS, если ваше приложение может позволить себе время от времени терять пакет. Например, если вы отправляете информацию о температуре каждую секунду и один пакет теряется, проблем не будет, поскольку температура не изменилась за одну секунду.
- QoS 1: По крайней мере один раз. Если вам нужно подтверждение того, чтоваше сообщение прибыло в пункт назначения, отправьте его с QoS 1. В этом случае брокер хранит сообщение до тех пор, пока не получит ответ от другого клиента. Например, если ваше сообщение должно включить реле, которое запускает воздушный вентилятор или лампу, вы должны быть уверены, что сообщение прибыло в пункт назначения.
- QoS 2: Ровно один раз, если у вас есть приложение, которое должно получать сообщения только один раз и не позволяет дублировать, используйте QoS 2 для вашего сообщения.

Безопасность

MQTT - это протокол, работающий по TCP / IP, поэтому вы можете использовать его для шифрования данных с помощью TLS / SSL и для безопасного соединения между клиентами.

Сохранять сообщения и последнее завещание

Представим, что нам нужно создать модуль, который будет каждый час отправлять данные о температуре и влажности на сервер и работать от батарей. Поскольку данные отправляются не очень часто и работают от батарей, модуль между отправками будет переведен в режим глубокого сна. В определенный момент времени, если данные запрашиваются каждые пять минут, нет другого способа указать модулю изменить время обновления, кроме как с помощью сообщения о сохранении. Из веб-приложения будет отправлено сообщение о сохранении, и брокер отправит это сообщение при первой передаче модуля. Таким образом, когда модуль просыпается и подписывается на сервер по теме конфигурации, брокер предоставит новый интервал обновления. Сообщение анализируется модулем, и с этого момента оно будет просыпаться и передавать каждые пять минут, а не каждый час.

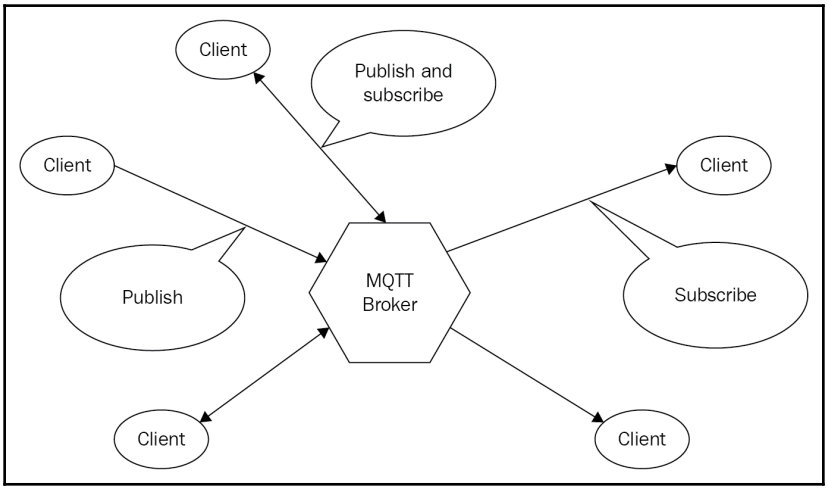
Когда у вас есть вариант использования для приложения, которое зависит от некоторых критических значений, отправляемых датчиком, вы можете определить, когда клиент прекратил передачу или теряет свою энергию, используя last will. Когда клиент подключается к брокеру, он также указывает последнюю тему желания и свое последнее сообщение. Например, если на клиенте произошел сбой сети или он не отвечает на сообщения Keep Alive, то брокер отправит всем клиентам, которые подписались на последнюю тему желания, сообщение от этого клиента. Обычно сообщение Last Will используется вместе с опцией Retain Message.

Основная терминология

До сих пор использовались такие термины, как центральный брокер, тема, публикация, подписка, поэтому пришло время объяснить их, используя аналогию с почтовым отделением и сообщения - это газеты или журналы:

- Брокер (Broker): это программное приложение (почтовое отделение), которое принимает сообщения (журналы) от клиентов (редакторов) и направляет сообщения в соответствии с запросами подписчика.
- Client (Клиент): это устройство, которое может публиковать сообщение (журнал) или получать сообщение (журнал), или и то, и другое.
- Topic (Тема): это строка (журнал), которая используется брокером для фильтрации сообщений для каждого подключенного клиента. Она отправляется клиентами брокеру в запросе на подписку, чтобы выразить желание получать сообщения, опубликованные другими клиентами. Она отправляется клиентами при публикации сообщений любому другому клиенту, который подписался на ту же тему.

- Publish (публиковать): действие по отправке сообщения (журнала) другому клиенту по определенной теме.
- Subscribe: действие по информированию брокера о заинтересованности в получении будущих сообщений, опубликованных другими клиентами по этой теме. Клиент может подписаться на несколько тем.
- Unsubscribe - Отказаться от подписки: действие клиента, которое сообщает брокеру не отправлять сообщения в указанную тему.



Архитектура MQTT

Поскольку одна из характеристик MQTT - гибкие темы подписки, давайте посмотрим, как формируется тема.
Темы состоят из иерархического уровня с использованием символа / в качестве разделителя уровней тем.

Вот примеры актуальных тем:

- `Europe/France/Paris/temperature`
- `62/livingroom/temperature`
- `456733-a55h56-667743/battery`

Темы чувствительны к регистру, поэтому Europe / France / Paris / temperature отличается от europe / France / Paris / temperature.

Подстановочные знаки в темах

При публикации данных другому клиенту через брокера необходимо указать полное имя темы: для получения сообщений клиенты могут подписаться с использованием подстановочного знака для тем.
Подстановочные знаки бывают одноуровневыми + и многоуровневыми #:

1. Одноуровневый +.
2. Использование + в уровне подписки для темы означает, что вместо + может быть любое значение.
3. Если вы хотите построить панель дисплея для отображения температуры в вашем доме, вы создаете модули, которые считывают температуру в каждой комнате и публикуете ее по таким темам как:

```
myHouse/groundFloor/livingroom/temperature
myHouse/groundFloor/kitchen/temperature
```

4. myHouse / firstFloor / bedroom / temperature и модуль дисплея будут подписываться на:

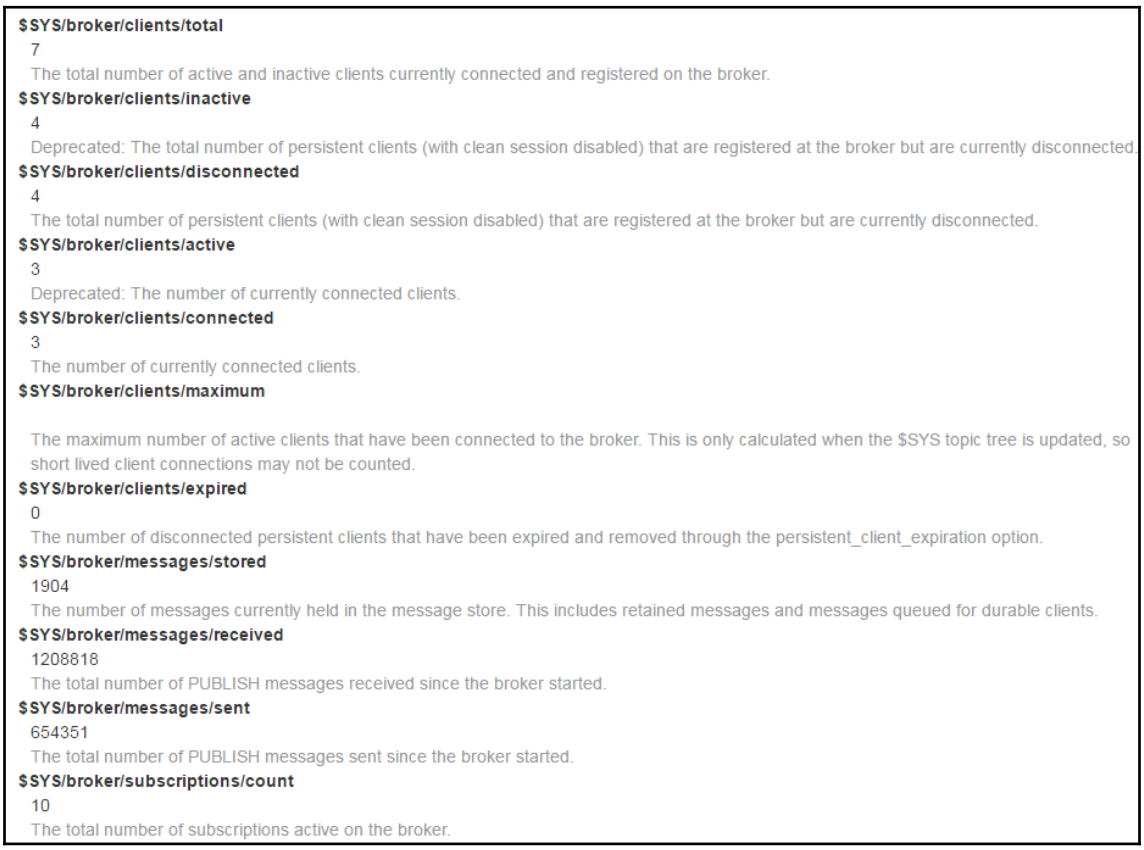
```
myHouse/groundFloor/+/temperature
```

5. Каждый раз, когда модуль температуры, расположенный на первом этаже, публикует какое-либо сообщение по своей теме, модуль дисплея будет получать его, поэтому нет необходимости подписываться по каждой теме, но он не будет получать данные из следующих тем:

```
myHouse/groundFloor/livingroom/humidity
myHouse/groundFloor/kitchen/light
```

6. Многоуровневый #.
7. Использование # в подписанном клиентском уровне темы будет получать все сообщения от этого.
8. Если панель отображения подписывается на тему.
9. myHouse / groundFloor / # означает, что он будет получать все сообщения, опубликованные по темам, которые начинаются с myHouse / groundFloor.
10. Если клиент подписывается на #, тема будет получать все сообщения, опубликованные вэтом брокере.
11. Специальные темы $.
12. Если вы хотите отслеживать внутреннюю статистику брокера, вам необходимо подписаться на темы $SYS.

13. Вот примеры того, что вы можете получить:



Пример вывода $SYS

Представляющий брокер Mosquitto

Eclipse MosquittoTM - это брокер MQTT с открытым исходным кодом, который реализует стандарты MQTT v3.1 и MQTT v.3.1.1 и предоставляет легкий метод передачи сообщений, позволяющий публиковать и подписываться на маломощные датчики, мобильные устройства, встроенные компьютеры и микроконтроллеры.

Вы можете установить Mosquitto на Raspberry Pi, на экземпляр AWS или на экземпляр VirtualBox Linux прямо из дистрибутива репозитория Linux; или вы можете получить исходный код и скомпилировать его самостоятельно, если вам нужна поддержка через веб-сокеты.

Установка из репозитория вашего дистрибутива Linux:

1. Первое обновление до последней версии:

```
sudo apt update && sudo apt upgrade
```

2. Затем установите mosquito:

```
sudo apt install mosquitto
```

Вы должны увидеть следующий экран:

```
catalin@plex:~/PROJECTS$ sudo apt install mosquitto
[sudo] password for catalin:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-4.4.0-31 linux-headers-4.4.0-31-generic linux-headers-4.4.0-72 linux-headers-4.4.0-72-generic linux-image-4.4.0-31-generic
  linux-image-4.4.0-72-generic linux-image-extra-4.4.0-31-generic linux-image-extra-4.4.0-72-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libev4 libwebsockets7
The following NEW packages will be installed:
  libev4 libwebsockets7 mosquitto
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 195 kB of archives.
After this operation, 540 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://de.archive.ubuntu.com/ubuntu xenial/universe amd64 libev4 amd64 1:4.22-1 [26.3 kB]
Get:2 http://de.archive.ubuntu.com/ubuntu xenial/universe amd64 libwebsockets7 amd64 1.7.1-1 [61.0 kB]
Get:3 http://de.archive.ubuntu.com/ubuntu xenial/universe amd64 mosquitto amd64 1.4.8-1build1 [108 kB]
Fetched 195 kB in 0s (262 kB/s)
Selecting previously unselected package libev4.
(Reading database ... 232626 files and directories currently installed.)
Preparing to unpack .../libev4_1%3a4.22-1_amd64.deb ...
Unpacking libev4 (1:4.22-1) ...
Selecting previously unselected package libwebsockets7:amd64.
Preparing to unpack .../libwebsockets7_1.7.1-1_amd64.deb ...
Unpacking libwebsockets7:amd64 (1.7.1-1) ...
Selecting previously unselected package mosquitto.
Preparing to unpack .../mosquitto_1.4.8-1build1_amd64.deb ...
Unpacking mosquitto (1.4.8-1build1) ...
Processing triggers for libc-bin (2.23-0ubuntu7) ...
Processing triggers for man-db (2.7.5-1) ...
Processing triggers for ureadahead (0.100.0-19) ...
Processing triggers for systemd (229-4ubuntu17) ...
Setting up libev4 (1:4.22-1) ...
Setting up libwebsockets7:amd64 (1.7.1-1) ...
Setting up mosquitto (1.4.8-1build1) ...
Processing triggers for libc-bin (2.23-0ubuntu7) ...
Processing triggers for systemd (229-4ubuntu17) ...
Processing triggers for ureadahead (0.100.0-19) ...
```

Установка Mosquitto

3. После установки брокера Mosquitto убедитесь, что он запущен, и установите mosquitto-client следующим образом:

```
catalin@plex:~/PROJECTS$ ps aux | grep mosquitto
mosquit+ 30970  0.0  0.1  42128  4736 ?        S    11:37   0:00 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
```

Убедитесь, что Mosquitto запущен

4. введите следующую команду:

```
sudo apt install mosquitto-clients
```

Вы получите следующий экран:

```
catalin@plex:~/PROJECTS$ sudo apt install mosquitto-clients
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-4.4.0-31 linux-headers-4.4.0-31-generic linux-headers-4.4.0-72 linux-headers-4.4.0-72-generic linux-image-4.4.0-31-generic
  linux-image-4.4.0-72-generic linux-image-extra-4.4.0-31-generic linux-image-extra-4.4.0-72-generic
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libc-ares2 libmosquitto1
The following NEW packages will be installed:
  libc-ares2 libmosquitto1 mosquitto-clients
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 96.2 kB of archives.
After this operation, 330 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://de.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libc-ares2 amd64 1.10.0-3ubuntu0.1 [34.1 kB]
Get:2 http://de.archive.ubuntu.com/ubuntu xenial/universe amd64 libmosquitto1 amd64 1.4.8-1build1 [31.1 kB]
Get:3 http://de.archive.ubuntu.com/ubuntu xenial/universe amd64 mosquitto-clients amd64 1.4.8-1build1 [31.0 kB]
Fetched 96.2 kB in 0s (226 kB/s)
Selecting previously unselected package libc-ares2:amd64.
(Reading database ... 232666 files and directories currently installed.)
Preparing to unpack .../libc-ares2_1.10.0-3ubuntu0.1_amd64.deb ...
Unpacking libc-ares2:amd64 (1.10.0-3ubuntu0.1) ...
Selecting previously unselected package libmosquitto1:amd64.
Preparing to unpack .../libmosquitto1_1.4.8-1build1_amd64.deb ...
Unpacking libmosquitto1:amd64 (1.4.8-1build1) ...
Selecting previously unselected package mosquitto-clients.
Preparing to unpack .../mosquitto-clients_1.4.8-1build1_amd64.deb ...
Unpacking mosquitto-clients (1.4.8-1build1) ...
Processing triggers for libc-bin (2.23-0ubuntu7) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up libc-ares2:amd64 (1.10.0-3ubuntu0.1) ...
Setting up libmosquitto1:amd64 (1.4.8-1build1) ...
Setting up mosquitto-clients (1.4.8-1build1) ...
Processing triggers for libc-bin (2.23-0ubuntu7) ...
```

Установка Mosquitto-клиентов

Клиенты Mosquitto поставляются с тремя очень важными утилитами:

- mosquitto_sub: простой клиент mqtt, который подписывается на одну тему и распечатывает все полученные сообщения.
- mosquitto_pub: простой клиент mqtt, который опубликует сообщение по одной теме и завершит работу.
- mosquitto_passwd: инструмент для управления файлами паролей для Mosquitto.

Использование любого из них с параметрами `--help`, например `mosquitto_sub --help`, предоставит исчерпывающий список всех параметров, которые вы можете использовать для тестирования своего брокера.

После установки Mosquitto запускается как служба и прослушивает порт 1883. Чтобы протестировать эту первоначальную установку, давайте подпишемся на тему и опубликуем сообщение по этой теме.

Подписка на тему с использованием параметра -t показана в следующей команде:

```
mosquitto_sub-t livingroom/temperature
```

Выполните эту команду следующим образом:

```
catalin@plex:~/PROJECTS$ mosquitto_sub -t livingroom/temperature
```

Subscribing to a topic, Observation (Подписка на тему, наблюдение): эта команда предполагает, что брокер работает на локальном сервере. Когда вы развертываете Mosquitto на виртуальном частном сервере (VPS) в Интернете для подписки, добавьте -h, чтобы указать IP-адрес вашего экземпляра сервера (используйте ifconfig, чтобы найти его) а также порт -p 1884, если вы не используете порт 1883 по умолчанию.В этом случае команда будет такой:

```
mosquitto_sub –h 46.102.34.87 –t livingroom/temperature
```

В другом терминале опубликуем какое-нибудь сообщение в формате JSON на ту же тему livingroom/temperature (гостиная / температура). Используйте следующую команду несколько раз и просмотрите сообщения:

```
mosquitto_pub –t livingroom/temperature –m {"t":27.4}
```

Вы должны увидеть следующий результат:

```
catalin@plex:~/PROJECTS$ mosquitto_sub -t livingroom/temperature
{t:27.4}
{t:27.4}
{t:27.4}
{t:27.4}
```

Первые данные, полученные по теме

После отправки сообщения mosquitto_pub выйдет. Если вы хотите отправлять значение каждую секунду, используйте утилиту watch, где -n 1 - количество секунд между командами:

```
watch –n 1 mosquitto_pub –t livingroom/temperature –m {"t":27.4}
```

Теперь вы можете выполнять упражнения, используя +, # в теме подписки mosquitto_sub -t livingroom / # предоставит вам все данные, которые отправляются с командами:

```
mosquitto_pub -t livingroom/temperature -m {"t":27.4}
mosquitto_pub -t livingroom/humidity -m {"h":68}
```

Вы должны увидеть следующее:

```
catalin@plex:~/PROJECTS$ mosquitto_sub -t livingroom/#
{t:27.4}
{h:68}
```

Получение данных от разных датчиков

И при использовани + в темах, подпишитесь на все темы температуры:

```
mosquitto_sub - t myhouse/+/temperature
```

И отправка из другого терминала сообщений, таких как:

```
mosquitto_pub -t myhouse/living/temperature -m {"t":25.6}
mosquitto_pub -t myhouse/kitchen/temperature -m {"t":27.1}
```

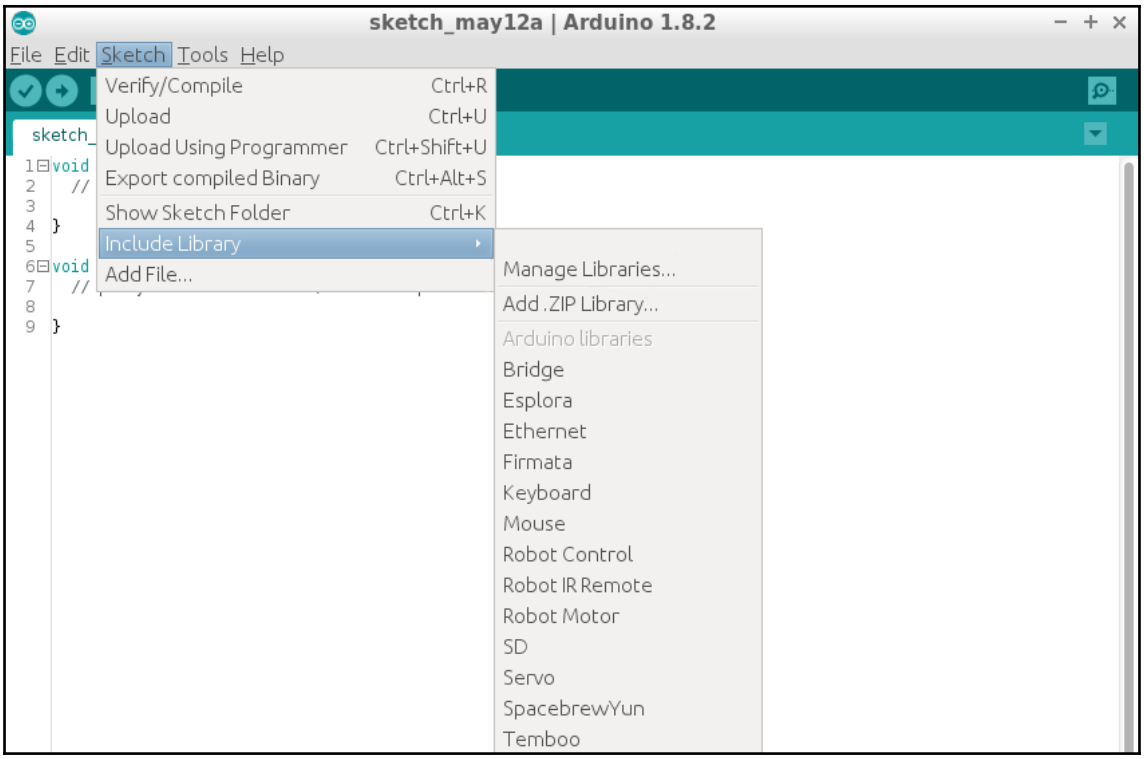
В терминале, на который вы подписались, вы увидите все следующие сообщения:

```
catalin@plex:~$ mosquitto_sub -t myhouse/+/temperature
{t:25.6}
{t:27.1}
{t:25.6}
{t:27.1}
```

ESP8266 и MQTT

Чтобы использовать ESP8266 в качестве клиента для отправки данных брокеру, вам понадобится библиотека, которая предлагает поддержку MQTT. Для этого вы можете использовать библиотеку PubSubClient, которую можно установить, как и другие библиотеки; см. главу 1, Начало работы с ESP8266:

1. Перейти к Sketch | Include Library | Manage Libraries ... следующим образом:



Manage libraries

2. Найдите библиотеку PubSubClient и нажмите «Install» (Установить), как показано на следующем снимке экрана:

Публикация данных из ESP8266

Начните новый скетч через Файл | Новый в среде Arduino IDE и вставьте следующий код. Включите библиотеки ESP8266WiFi и PubSubClient:

```
#include <ESP8266WiFi.h>#include
<PubSubClient.h>
```

Обновите их значениями, подходящими для вашей сети. Используйте ifconfig, чтобы получить IP-адрес сервера, на котором установлен брокер Mosquitto. Если у вашего сервера есть полное доменное имя, например myiotserver.com, и он зарегистрирован в DNS, то вместо IP-адреса в mqtt_server вы можете использовать полное доменное имя:

```
const char* wifi_network = "YOUR_WIFI_SSID";const
char* wifi_pass = "YOUR_WIFI_PASSWORD";const char*
mqtt_serv_address = "192.168.1.116";const int
mqtt_port_number = 1883;
```

Создайте экземпляр WiFiClient и передайте его PubSubClient:

```
WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;
```

Функция setup () начнет подключать ESP8266 к сети Wi-Fi, вызвав функцию setup_wifi () и настроив сервер MQTT и параметр, который будет использоваться через функцию client.Setserver ():

```
void setup() {
Serial.begin(115200);
setup_wifi();
client.setServer(mqtt_serv_address, mqtt_port_number);
}

void setup_wifi() {

delay(10);
  // Начнем с подключения к сети Wi-Fi
Serial.println();
Serial.print("Connecting to ");
Serial.println(wifi_network);

WiFi.begin(wifi_network, wifi_pass);

while (WiFi.status() != WL_CONNECTED) {
```

```
WiFi.begin(wifi_network, wifi_pass);

Serial.print(".");
delay(5000);
  }

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}
```

Если пакеты keep alive от сервера MQTT к модулю ESP8266 потеряны и связь прервана, функция reconnect () попытается снова подключиться к серверу MQTT. Эта функция повторного подключения также используется в качестве первого подключения к серверу.

После подключения к серверу MQTT ESP8266 опубликует сообщение «Hello world, I am ESP8266!» по теме fromEsp8266:

```
void reconnect() {
  // Цикл, пока мы снова не подключимся
while (!client.connected()) {
Serial.print("Attempting MQTT connection...");
    // Попытка подключиться
if (client.connect("ESP8266Client"))
    {
Serial.println("connected");
      // После подключения публикуем объявление ...
client.publish("fromEsp8266", "Hello world, I am ESP8266!");
    } else {
Serial.print("failed, rc=");
Serial.print(client.state());
Serial.println(" try again in 5 seconds");
      // Подождите 5 секунд перед повторной  попыткой
delay(5000);
    }
  }
}
```

Функция loop (цикл) будет проверять подключение к брокеру MQTT, повторно подключаться к нему, если есть проблема с подключением, и каждые две секунды будет публиковать сообщение в теме fromEsp8266:

```
void loop() {

if (!client.connected()) {
reconnect();
  }
client.loop();

long now = millis();
if (now - lastMsg> 2000) {
lastMsg = now;
    ++value;
snprintf (msg, 75, "Hello world #%ld", value);
Serial.print("Publish message: ");
Serial.println(msg);
client.publish("fromEsp8266", msg);
  }
}
```

После компиляции и загрузки кода в модуль ESP8266 в окне терминала подпишитесь на тему fromEsp8266. Сообщения, отправленные модулем ESP8266, будут отображаться в окне терминала:

```
catalin@plex:~$ mosquitto_sub -h 192.168.1.116 -t fromEsp8266
Hello world, I am ESP8266!
Hello world #1
Hello world #2
Hello world #3
Hello world #4
Hello world #5
Hello world #6
```

Сообщения, опубликованные ESP8266

Примечание: помните, что названия тем чувствительны к регистру. В качестве упражнения вы можете отправить статус вывода HIGH или LOW (ВЫСОКИЙ или НИЗКИЙ), как было описано в главе 1 «Начало работы с ESP8266».

Используйте digitalRead (PIN_NUMER) вместо value.

Получение сообщений MQTT в ESP8266

Теперь давайте опубликуем сообщение с помощью mosquitto_pub и получим его в ESP8266.

Для этого ESP8266 необходимо подписаться на ту же тему, в которой mosquitto_pub опубликует сообщение. Назовем тему - наружный / свет, иона будет публиковаться по значениям 0 или 1. Если ESP8266 получит значение 1,он включит светодиод, подключенный к GPIO 12 и если он получит 0, он выключит этот светодиод:

```
#include <ESP8266WiFi.h>
#include
<PubSubClient.h>
```

Обновите их значениями, подходящими для вашей сети:

```
const char* wifi_network= "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
const char* mqtt_serv_address = "YOUR_MQTT_SERVER_IP";
const int mqtt_port_number = 1883;
#define OUTDOOR_LIGHT  12

WiFiClient  espClient;
PubsubClient client(espClient);
long lastMsg; = 0;
```

Запустите подключение к сети Wi-Fi и задайте имя функции, которая будет вызываться при получении сообщения от брокера MQTT, как показано ниже:

```
void setup() {
pinMode(OUTDOOR_LIGHT, OUTPUT); // Инициализируем вывод BUILTIN_LED как выход

Serial.begin(115200);
setup_wifi();
client.setServer(mqtt_serv_address, mqtt_port_number);
client.setCallback(callback);
}
```

Подключитесь к сети Wi-Fi:

```
void setup_wifi() {
delay(10);
  // Начнем с подключения к сети Wi-Fi
Serial.println();
Serial.print("Connecting to ");
Serial.println(wifi_network);

WiFi.begin(wifi_network, password);
```

```
while (WiFi.status() != WL_CONNECTED) {
WiFi.begin(wifi_network, password);

Serial.print(".");
delay(5000);
  }

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}
```

Когда сообщение поступит в клиент MQTT ESP8266, будет вызвана функция callback () с параметрами topic, содержащими имя темы, по которой пришло сообщение, в случае, если этот модуль подписался на несколько тем, фактическое содержимое тема сообщения и длина сообщения:

```
void callback(char* topic, byte* payload, unsigned int msg_length) {
Serial.print("Message arrived [");
Serial.print(topic);
Serial.print("] ");
for( int i = 0; i < msg_length; i++) {
Serial.print((char)payload[i]);
  }
Serial.println();

  // Включаем светодиод, если в качестве первого символа была получена 1
if ((char)payload[0] == '0') {
digitalWrite(OUTDOOR_LIGHT, LOW);   // Выключаем светодиод
  } else {
digitalWrite(OUTDOOR_LIGHT, HIGH);  // Включаем светодиод
  }
}
```

В функции reconnect () он также подпишется на тему outdoor / light, из которой он будет получать сообщения, и если соединение с брокером будет потеряно, он будет пытаться подключиться к нему каждые пять секунд:

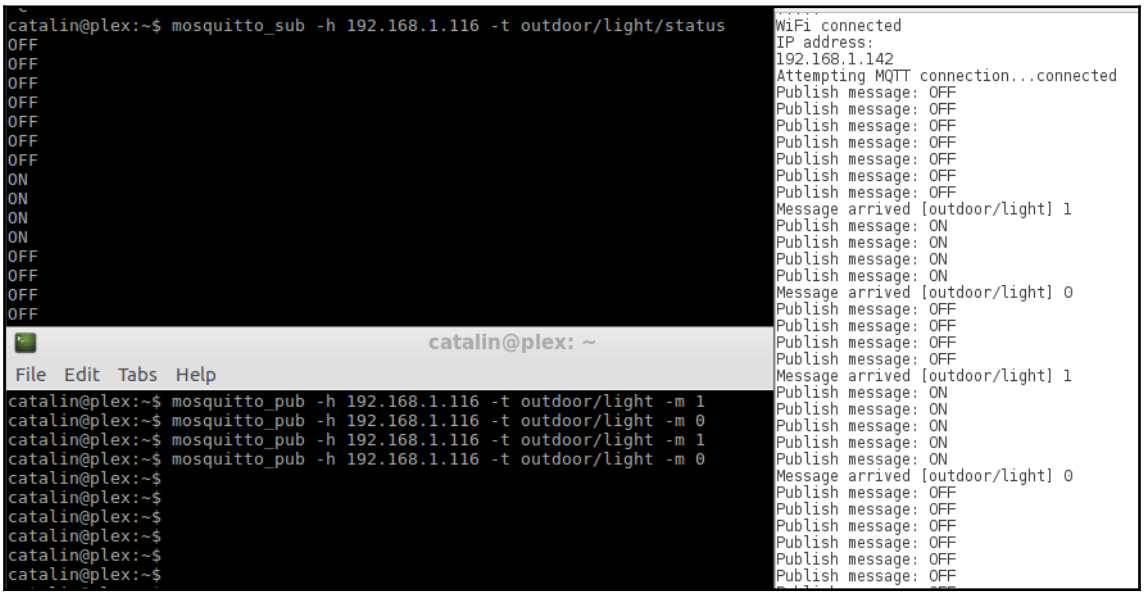
```
void reconnect() {
  // Цикл, пока мы снова не подключимся
while (!client.connected()) {
Serial.print("Attempting MQTT connection...");
    // Попытка подключиться
if (client.connect("ESP8266Client"))
    {
Serial.println("connected");
```

```
client.subscribe("outdoor/light");
    } else {
Serial.print("failed, rc=");
Serial.print(client.state());
Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
delay(5000);
    }
  }
}
```

Функция loop () опубликует значение GPIO 12, которое фактически является состоянием наружного освещения:

```
void loop() {

if (!client.connected()) {
reconnect();
  }
client.loop();

long now = millis();
if (now - lastMsg> 2000) {
lastMsg = now;
    String light_state;

if(digitalRead(OUTDOOR_LIGHT) == HIGH)
light_state = "ON";
else
light_state = "OFF";


Serial.print("Publish message: ");
Serial.println(light_state);
client.publish("outdoor/light/status", light_state.c_str());
  }
}
```

В окне Terminal вы увидите ON (ВКЛ) или OFF (ВЫКЛ) в зависимости от значения последнего сообщения, переданного по теме «outdoor / light -Наружный / свет». Обратите внимание, что модуль подписывается на тему outdoor / light и публикует статус GPIO в теме outdoor / light / status:



Отправка и получение сообщений

Обеспечение Mosquitto

Если ваш брокер Mosquitto MQTT находится в облаке, рекомендуется защитить его, по крайней мере, с помощью пользователя и пароля.

Mosquitto предлагает утилиту mosquitto_passwd, которая позволяет нам создать пользователя и пароль. Вам будет предложено ввести пароль и подтвердить его:

```
sudo mosquito_passwd -c /etc/mosquito/passwd   joe
```

В каталоге / etc / mosquitto будет создан файл с именем passwd, в котором будет находиться пользователь с именем joe и его закодированный пароль, как показано на следующем снимке экрана:

Теперь давайте добавим файл passwd в mosquitto.conf. Используйте свой любимый текстовый редактор и измените файл /etc/mosquitto.conf, чтобы Mosquitto читал и использовал файл passwd.

Содержимое файла будет:

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

allow_anonymous false
password_file /etc/mosquitto/passwd

include_dir /etc/mosquitto/conf.d
```

Расположение файла с паролем

- allow_anonymous: это логическое значение, которое определяет, разрешено ли подключение клиентам, которые подключаются без указания имени пользователя. Если установлено значение false, то другое означает, что необходимо создать соединение для управления доступом аутентифицированного клиента.

- password_file: Устанавливает путь к файлу паролей. Если определено, содержимое файла используется для управления доступом клиента к брокеру.
  Если для параметра allow_anonymous установлено значение false, только пользователи, указанные в этом файле, смогут подключиться.

После остановки и перезапуска сервиса Mosquitto с помощью команд:

```
sudo service mosquitto stop && sudo service mosquitto start
```

Если окно терминала попытается подключиться, как вы это делали раньше, мы получим сообщение об ошибке, как на следующем снимке экрана:

```
catalin@plex:/etc/mosquitto$ mosquitto_sub -t living/temperature
Connection Refused: not authorised.
Connection Refused: not authorised.
Connection Refused: not authorised.
Connection Refused: not authorised.
Connection Refused: not authorised.
```

Но подписка и публикация с пользователем и паролем, созданным ранее, будут работать.
Для подписки используется следующая команда:

```
sudo mosquitto_sub –t living/temperature –u joe –P joe1234
```

А для публикации:

```
sudo mosquitto_pub –t living/temperature –u joe –P joe1234 –m {"t":24.7}
```

Убедитесь, что вы используете для пароля заглавную букву «P», а не «p», обозначающую порт.

Вы получите следующий результат:

```
catalin@plex:/etc/mosquitto$ mosquitto_sub -t living/temperature -u joe -P joe1234

{t:24.7}
{t:24.7}
{t:24.7}
{t:24.7}
{t:24.7}
```

Предоставление имени пользователя и пароля

Теперь давайте используем ESP8266 для отправки имени пользователя и пароля брокеру:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
```

Обновите их значениями, подходящими для вашей сети:

```
const char* wifi_network = "WiFi 176-58";
const char* password = "P6etRUzaRa";
const char* mqtt_serv_address = "192.168.1.116";
const char* mqtt_user = "joe";
const char* mqtt_passwd = "joe1234";
const int   mqtt_port_number = 1883;
#define OUTDOOR_LIGHT  12
```

Mqtt_user сохранит значение имени пользователя, а mqtt_passwd сохранит пароль пользователя. Мы будем использовать их при подключении и передавать на сервер:

```
WiFiClient  espClient;PubSubClient   client
(espClient);
long  lastMsg = 0;

void setup() {
pinMode(OUTDOOR_LIGHT, OUTPUT); // Инициализируем вывод BUILTIN_LED как выход

Serial.begin(115200);
setup_wifi();
client.setServer(mqtt_serv_address,
mqtt_port_number);client.setCallback(callback);
}

void setup_wifi() {

delay(10);
  // Начнем с подключения к сети Wi-Fi
Serial.println();
Serial.print("Connecting to ");
Serial.println(wifi_network);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
WiFi.begin(wifi_network, password);

Serial.print(".");
delay(5000);
  }

Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int msg_length) {
Serial.print("Message arrived [");
Serial.print(topic);
Serial.print("] ");
for (int i = 0; i< msg_length; i++) {
Serial.print((char)payload[i]);
  }
Serial.println();
```

```
  // Включаем светодиод, если в качестве первого символа была получена 1
if ((char)payload[0] == '0') {
digitalWrite(OUTDOOR_LIGHT, LOW);   // Выключаем светодиод
  } else {
digitalWrite(OUTDOOR_LIGHT, HIGH);  // Включаем светодиод
  }
}

void reconnect() {
  // Цикл, пока мы снова не подключимся
while (!client.connected()) {
Serial.print("Attempting MQTT connection...");
```

Попытайтесь подключиться к брокеру MQTT Mosquitto, используя имя пользователя и пароль от mqtt_user и mqtt_passwd:

```
if (client.connect("ESP8266Client", mqtt_user, mqtt_passwd))
{
  Serial.println("connected");
  client.subscribe("outdoor/light");
} else {
  Serial.print("failed, rc=");
  Serial.print(client.state());
  Serial.println(" try again in 5 seconds");
  // Подождите 5 секунд перед повторной попыткой
  delay(5000);
  }
 }
}

void loop() {

if (!client.connected()) {
    reconnect();
}
client.loop();

long now = millis();
if (now - lastMsg> 2000) {
    lastMsg = now;
    String light_state;

    if(digitalRead(OUTDOOR_LIGHT) == HIGH)
       light_state = "ON";
    else
       light_state = "OFF";
```

```
      Serial.print("Publish message: ");
      Serial.println(light_state);
      client.publish("outdoor/light/status", light_state.c_str());
    }
  }
```

Публикация сообщения с полезной нагрузкой 1 запустит GPIO 12 и включит подключенный светодиод безопасным способом. Сообщения ON OFF - это состояние GPIO, где ON будет иметь 3V3, а ON OFF 0V:

```
catalin@plex:/etc/mosquitto$ mosquitto_sub -t outdoor/light/status -u joe -P joe1234
OFF
OFF
OFF
ON
ON
ON
ON
OFF
OFF
OFF
```

Получение сообщений ВКЛ / ВЫКЛ

Теперь отправка этого сообщения из любой точки мира будет включать и выключать любое устройство в вашем доме, если вместо светодиода мы добавим плату реле:

> До сих пор использовались небольшие сообщения, по умолчанию максимальный размер, разрешенный PubSubClient, составляет 128 байт. Если размер ваших сообщений превышает128 байт, перейдите в файл PubSubClient.h и измените значение MQTT_MAX_PACKET_SIZE.

Резюме

Использование сервера MQTT, такого как Mosquitto, очень важно, потому что он позволяет вам связывать ПК с машиной M2M, и вы можете начать автоматизировать многие задачи, от включения света в доме, если это пасмурный день, до автоматического управления климатом вашего дома.

3

Создание домашнего термостата с ESP8266

В этой главе мы построим домашний термостат с ESP8266. Термостат будет выполнять следующие функции:

- Он будет считывать температуру с датчика температуры DHT22.
- Он сравнит температуру с желаемой; если она выше - выключит реле, а если она ниже - включит реле.

Но сначала давайте обсудим, как мы можем сохранять данные в ESP8266 и извлекать их. Воспользуемся SPIFFS.

SPIFFS

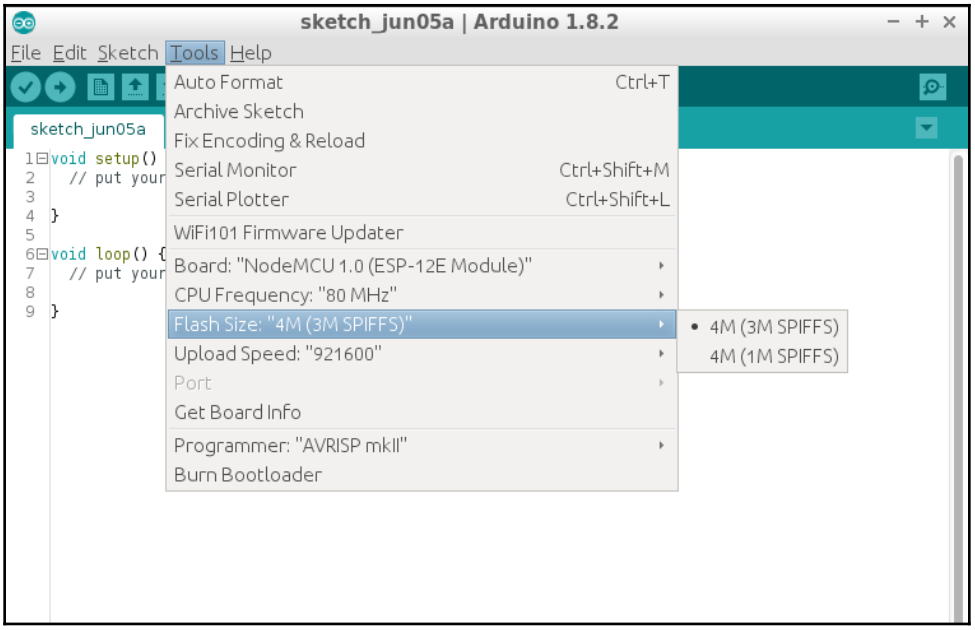
SPI Flash File System (SPIFFS) - это файловая система, созданная для небольших встроенных систем. SPIFFS имеет множество преимуществ, поскольку позволяет создавать файлы и моделировать каталоги.

Ниже перечислены особенности SPIFFS:

- Разработан для использования небольших ОЗУ на микроконтроллерах
- Использует буферы RAM статического размера
- Posix-подобный API: открытие, закрытие, чтение, запись, поиск, статистика и т. д.
- Он может работать на любой NOR флеш-памяти, а не только на флеш-памяти SPI. Несколько конфигураций SPIFFS могут работать для одной цели - и даже на одном флэш-устройстве SPI.

- Выполняет выравнивание статического износа
- Встроенные проверки целостности файловой системы
- Широкие возможности настройки и адаптация к различным типам флеш-памяти

Я настоятельно рекомендую вам использовать SPIFFS в ваших проектах для хранения данных во флэш-памяти NOR, поскольку их очень легко читать и записывать, это похоже на файловую систему * nix.Размер файловой системы зависит от размера флеш-чипа. В зависимости от платы,выбранной в IDE, вы можете выбрать разные размеры для SPIFFS. Например, если вы выбрали NodeMcu v1.0 в качестве типа платы, существуют два значения для SPIFFS, одно из 1M и 3M, как показано на следующем снимке экрана:



Размер флэш-памяти для NodeMcu

Несмотря на то, что файловая система хранится на той же микросхеме флэш-памяти, что и прикладная программа, обновление нового скетча не приведет к изменению содержимого файловой системы. Это позволяет нам использовать файловую систему для хранения данных, файлов конфигурации или контента для веб-серверов.

Давайте теперь посмотрим, какие функции доступны для управления файлами. Прежде всего, у них есть доступ к функции SPIFFS, и файл FS.h необходимо включить в скетч:

```
#include "FS.h"
```

После этого включения у нас есть доступ к трем объектам: SPIFFS, File и Dir. Подробнее об этих объектах мы узнаем в следующих разделах.

Объекты SPIFFS

Давайте посмотрим на несколько объектов SPIFFS:

- begin: Монтирует файловую систему. Должен быть вызван первым и возвращает true в случае успеха или false в противном случае:

```
if(SPIFFS.begin())
{
  Serial.println(F("File systestem mounted.")); //используйте
функцию F, чтобы сохранить строку во флэш-памяти, а не в оперативной
памяти. Это сэкономит много оперативной  памяти.
}
else
{
  Serial.println(F("Mounting file system failed."));
}
```

- info: возвращает информацию обо всей файловой системе, информацию, которая хранится в структуре FSInfo. Структура FSInfo состоит из следующих членов:

```
struct FSInfo {
    size_t totalBytes;
    size_t usedBytes;
    size_t blockSize;
    size_t pageSize;
    size_t maxOpenFiles;
    size_t maxPathLength;
};
```

  Объявление fs_info и заполнение его функцией info позволит нам получить доступ к информации о файловой системе:

```
FSInfo fs_info;
SPIFFS.info(fs_info);
Serial.print("Used bytes: ");
Serial.println(fs_info.usedBytes);
```

- exists: SPIFFS.exists (путь) возвращает true (истину) или false (ложь), если указанный путь (файл) существует в файловой системе:

```
if(SPIFFS.exist("/config.json"))
    Serial.println(F("File config.json exists."));
```

Путь должен быть абсолютным и начинаться с косой черты.

- format: форматирует всю файловую систему и возвращает истину или ложь. Format может быть вызван до или после функции begin, и это займет несколько десятков секунд в зависимости от размера вашей файловой системы:

```
SPIFFS.format();
```

- open: эта функция возвращает объект File и принимает в качестве параметров абсолютный путь к файлу и режим открытия файла. Возвращает истину в случае успеха или ложь в противном случае:

```
SPIFFS.open(path, mode);

File config_file = SPIFFS.open("/config.json", "w");
if (!config_file) {
    Serial.println(F("failed opening config.json file."));
}
```

В предыдущем коде выбранный режим для открытия файла был w (запись). Могут использоваться те же режимы, что и функция fopen ANSI C:

r Открыть текстовый файл для чтения. Поток располагается в начале файла.

r+ Открыт для чтения и записи. Поток располагается в начале файла.

w Обрезать файл до нулевой длины или создать текстовый файл для записи. Поток располагается вначале файла.

w+ Открыт для чтения и записи. Если файл не существует, он создается; в противном случае он усекается. Поток располагается в начале файла.

a Открыть для добавления (запись в конец файла). Если файл не существует, он создается. Поток располагается в конце файла.

a+ Открыть для чтения и добавления (запись в конце файла). Если файл не существует, он создается. Начальная позиция файла для чтения находится в начале файла, но вывод всегда добавляется в конец файла.

- remove: удалить файл из файловой системы. Он принимает в качестве параметра абсолютный путь и возвращает true - истину в случае успеха или false - ложь в противном случае:

```
if(SPIFFS.remove("/config.json"))
   Serial.println(F("File config.json was removed"));
```

- rename: переименовывает файл. Принимает два параметра: абсолютные пути для текущего имени и нового имени. Возвращает true в случае успеха или false в противном случае:

```
If( SPIFFS.rename("/old_file_name.json","/new_file_name.json")
    Serial.println(F("File renamed."));
```

Объект каталога

Если вам нужно перебрать все файлы из каталога, вы можете использовать объект **Dir** (Directory). Доступны три метода для итерации next (), получения имени следующего файла File Name () и открытия каталога openDir (mode):

SPIFFS не поддерживает каталоги. Фактически получается плоская структура.
Создание файла с путем /data/log.txt создаст файл с именем /data/log.txt вместо log.txt в данных каталога.

```
Dir dir = SPIFFS.openDir("/data");
while (dir.next()) {
    Serial.print(dir.fileName());
    File f = dir.openFile("r");
    Serial.println(f.size());
}
```

Из предыдущего кода мы узнаем, что:

- mode (режим) из openDir может иметь те же значения, что и функция open из объекта SPIFFS
- dir.next () возвращает true, если в каталоге данных есть файлы, и должен вызываться перед функциями fileName () и openFile (mode)

Объект File

Есть две функции, которые возвращают объект File, SPIFFS.open и dir.OpenFile, и этот объект File позволяет нам читать байты, заглядывать в файл и получать текущую позицию в файле, получать имя файла или его размер. Ниже перечислены различные функции, используемые объектом File:

- close: закрыть файл. Никакие другие операции не должны выполняться с объектом File после вызова функции close ():

```
file_name.close();
```

- name: возвращает имя текущего файла:

```
String name = my_file.name();
```

- position: возвращает текущую позицию в файле в байтах. Он используется, если вы храните данные того же размера и выполняете итерацию в файле.
- seek: позволяет перейти к файлу. Требуется два параметра; один -смещение в файле, а другой - режим. Возвращает true в случае успеха и false в противном случае:

```
my_file.seek(offset, mode);
```

- Параметр mode может иметь те же значения, что и в функции C fseek ():

  Если режим SeekSet, позиция устанавливается на смещение байтов от начала.

  Если режим SeekCur, текущая позиция перемещается на байты смещения.

  Если режим WeekEnd, позиция устанавливается на смещение в байтах от конца файла.

- size: получить текущий размер файла в байтах:

```
Serial.println(my_file.size());
```

В качестве примера давайте создадим файл, напишем в него что-нибудь и прочитаем то, что было написано; если GPIO 4 подключить к GND, он отформатирует флешку.

Если вы помните, для доступа к объекту SPIFFS необходимо включить FS.h:

```
#include "FS.h"
```

Установите для interruptPin значение 4 и создайте экземпляр interruptCounter равным нулю:

```
const byte interruptPin = 4;
volatile byte interruptCounter = 0;
```

Эта функция будет вызываться, если GPIO будет подключен к GND. В нем он будет увеличивать interruptCounter и проверять его значение в функции loop (). Старайтесь не выделять память на уровне прерываний; это рецепт катастрофы:

```
void handleInterrupt() {
  interruptCounter++;
}
```

В setup () мы установим GPIO 4 как входной и будем использовать функцию attacheInterrupt для FALLING (VCC to GND), чтобы запустить функцию handleInterrupt. Затем мы открываем my_file.txt и записываем в него сообщение. После закрытия файла он снова откроется, но теперь он находится в режиме чтения и будет читать его содержимое:

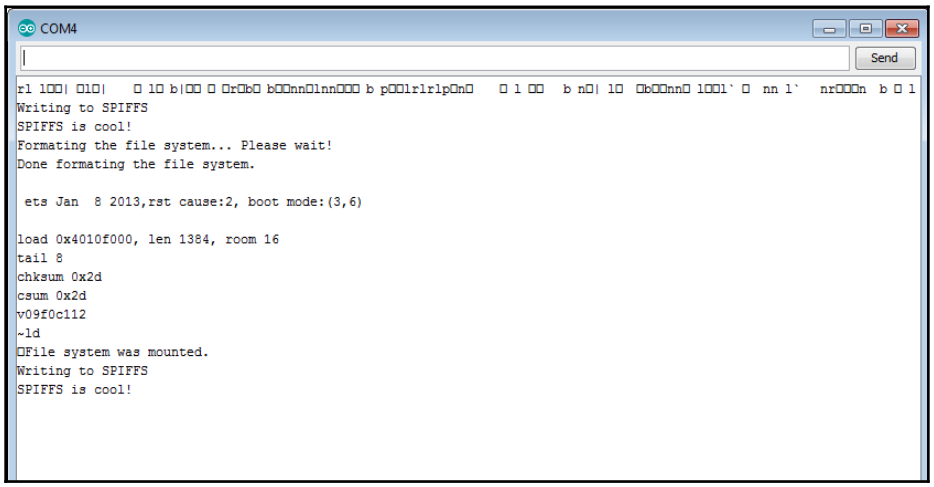
```
 void setup() {
  Serial.begin(115200); delay(10);
  //GPIO 4 format SPIFFS
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), handleInterrupt,
FALLING);

  if(SPIFFS.begin())
  {
    Serial.println(F("File system was mounted."));
    // open file for writing
    File my_file = SPIFFS.open("/my_file.txt", "w+");
    if (!my_file) {
      Serial.println("file open failed");
    }
    Serial.println(F("Writing to SPIFFS "));
    //print something to my_file.txt
    my_file.println("SPIFFS is cool!");
    //close now the file
    my_file.close();

    // open file for reading. Now I can use other File object
    File f = SPIFFS.open("/my_file.txt", "r");
    if (!f)
    {
       Serial.println(F("Failed to open my_file.txt"));
    }
    //now read the file content
    String s=f.readStringUntil('\n');
```

```
    Serial.println(s);
    //closing the file now
    f.close();
  }
  else
  {
    Serial.println(F("Failed to mount SPIFFS. Restart"));
    ESP.restart();
  }
}
```

В файле loop (цикла) мы проверяем значение intrruptCounter, и если оно больше нуля, он отформатирует файловую систему и перезапустит ESP:

```
void loop()
{
  if(interruptCounter>0)
  {
    interruptCounter--;
    Serial.println(F("Formating the file system... Please wait!"));
    SPIFFS.format();
    Serial.println(F("Done formating the file system."));
    ESP.restart();
  }
}
```

Последовательный вывод покажет, что было прочитано из файла, а также сообщения форматирования и новый перезапуск после этого, как показано на следующем снимке экрана:



Вывод для записи, чтения и форматирования

Теперь, когда вам удалось понять SPIFFS, я уверен, что вы будете использовать его во всех своих проектах, поскольку это очень удобный способ хранить данные конфигурации или всю информацию, такую как показания датчиков, особенно при батарейном питании. Вы можете сохранить все показания в файле в формате JSON и раз в день подключаться к Wi-Fi, читать файл и отправлять все данные сразу. После этого длинного, но полезного введения, давайте вернемся к нашему проекту - термостату, использующему ESP8266. Для этого помимо ESP8266 нам понадобятся:

- Датчик температуры
- Плата реле

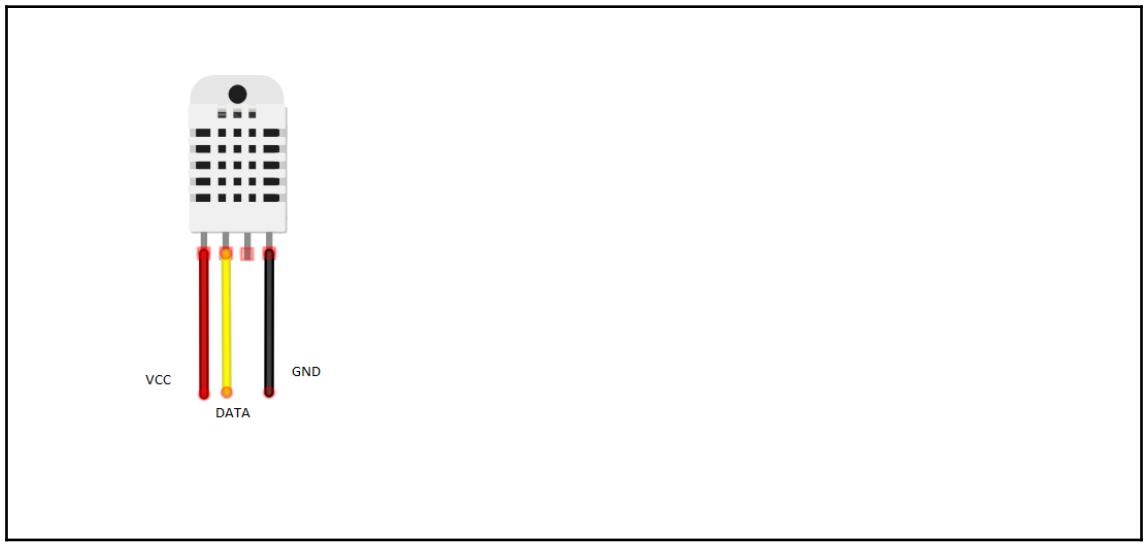
Датчик температуры

Для этого проекта мы будем использовать очень распространенный датчик температуры DTH22. Он может измерять температуру и влажность.

Ниже приведены характеристики DHT22:

- Бюджетный
- Питание от 3 до 5 В и ввод / вывод
- I_{max}=5 мА, используемый во время преобразования (при запросе данных)
- Измерение влажности 0-100% с точностью 2-5%
- Измеряет температуру от -40 до 125 ° C с точностью ± 0,5 ° C
- Частота дискретизации не более 0,5 Гц (один раз в две секунды)
- Размер корпуса 15,1 мм x 25 мм x 7,7 мм
- 4 вывода с шагом 0,1 дюйма (0,25 мм)

DHT22 можно найти как отдельный датчик так и датчик с начинкой. Предпочтительно покупать версию с начинкой,где начинка это подтягивающий резистор 4K7 и конденсатор. Если вы предпочитаете только датчик, вот распиновка:



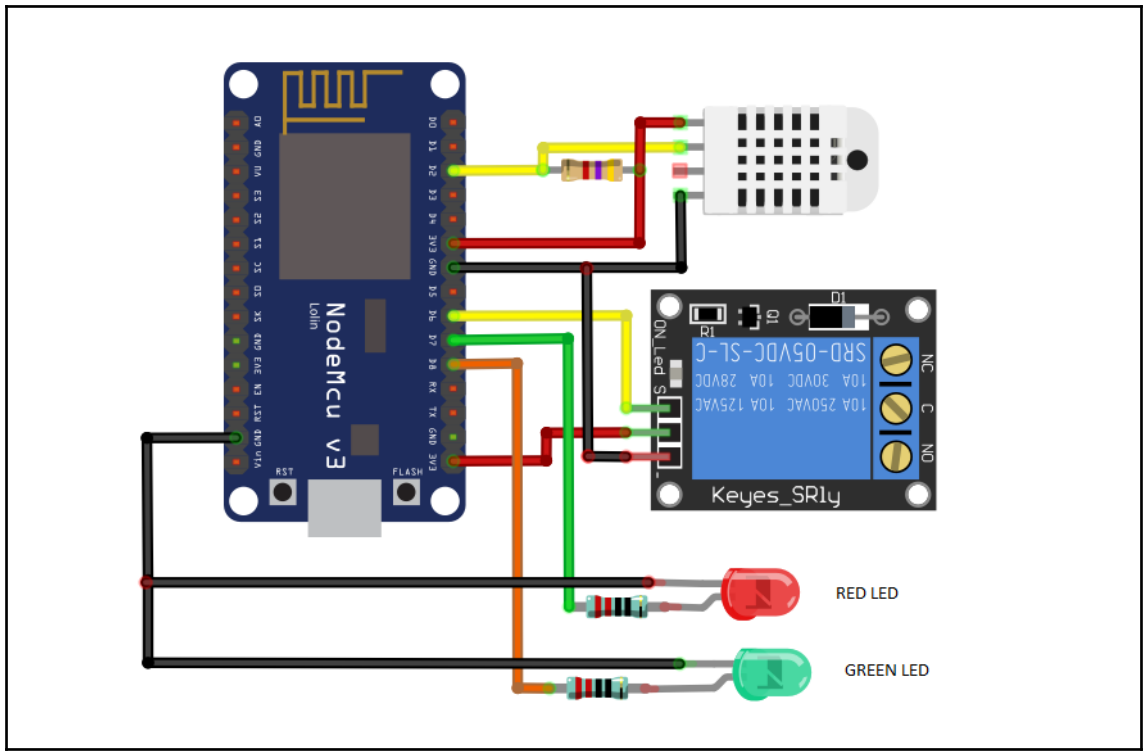
Распиновка DHT22

Здесь:

- **VCC**: может быть от 3V3 до 5V
- **GND**: это земля
- **DATA**: вывод данных

Не забудьте добавить подтягивающий резистор 4K7 между выводами DATA и VCC. Для подключения к газовой печи или другому нагревательному элементу на GPIO 12 будет добавлено реле. Убедитесь, что у вас хороший источник питания, поскольку реле потребляет большой ток.

В этом случае наши подключения будут такими:



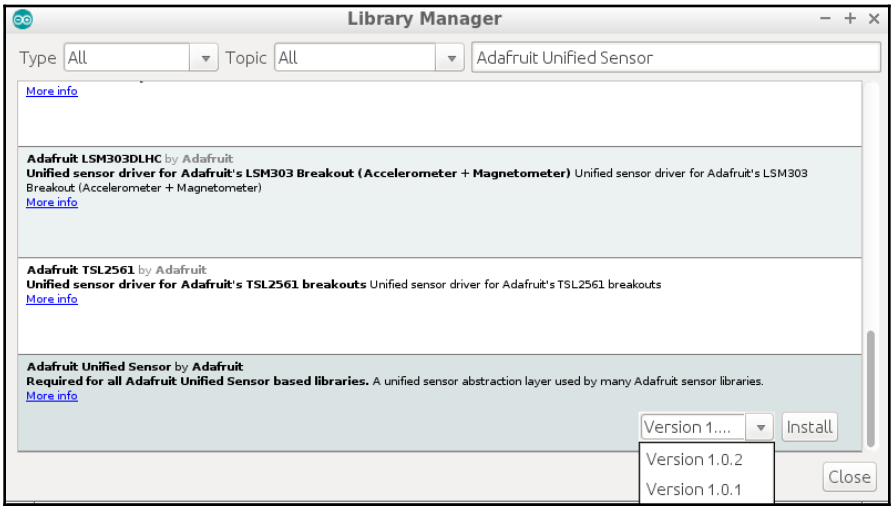
Конечная схема для термостата

Были добавлены два светодиода: один КРАСНЫЙ, чтобы показать, что идет нагрев, и один ЗЕЛЕНЫЙ, чтобы показать, что на систему подано питание. Зеленый светодиод загорается только тогда, когда красный не горит.
На плате реле соединение между C (общим) и NO (нормально разомкнутым) существует только тогда, когда GPIO 12 находится в состоянии ВЫСОКИЙ, и идет нагрев.

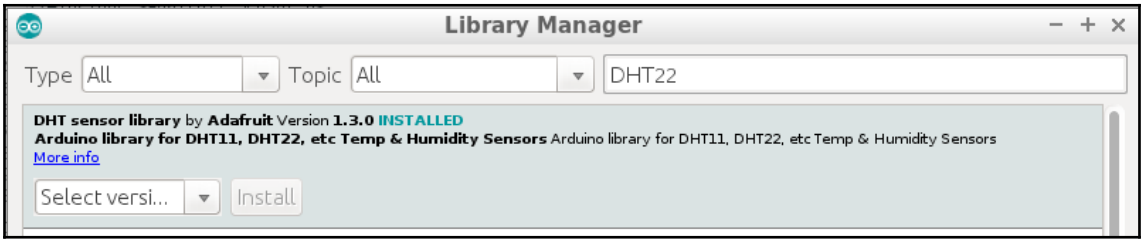
Будьте осторожны с электричеством 220 В и убедитесь, что вы выбрали реле, способное работать с подключенной нагрузкой

В основном термостат будет измерять температуру, и если она выше желаемой, он выключит реле, а если ниже, включит реле, чтобы начать нагрев.

Если мы сделаем нашу логику такой, что термостат будет очень часто включать и выключать реле, поэтому нам нужно добавить смещение (дельту) между началом и остановкой нагрева. На коммерческих термостатах это смещение может быть запрограммировано (с шагом 0,1ºC до 1ºC) или может быть зафиксировано как 0,5ºC. В нашей системе смещение будет установлено на 0,4ºC. Это означает, что если желаемая температура составляет 22,0 ° C, нагреватель запускается с 21,6 ° C и останавливается при 22,4 ° C.
Чтобы установить желаемую температуру на нашем термостате, мы отправим сообщение MQTT в тематический термостат / набор с содержанием желаемой температуры (например, 23.2); система сохранит значение в файл с помощью SPIFFS и сравнит его с текущей температурой, считанной с датчика DHT22. В случае сбоя питания желаемая температура будет считана из файла конфигурации, поэтому она не будет потеряна.
Когда термостат включается в первый раз, желаемая температура жестко запрограммирована на 22ºC. Периодически термостат будет показывать данные на термостате / получать текущую температуру, измеренную DHT22.
Чтобы использовать датчик DHT22, вам потребуются некоторые библиотеки для него. Установите унифицированный датчик Adafruit, а затем библиотеку датчиков DHT, следуя той же процедуре, что и в главе 1 «Начало работы с ESP8266».
Для поиска Adafruit Unified Sensor в Менеджере библиотек для Adafruit Unified Sensor:



Для поиска библиотеки DHT22 в диспетчере библиотек для DHT22:



После установки обеих библиотек у нас есть все необходимые включения:

```
#include <FS.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
```

Константы, которые будут использоваться позже в нашем коде, следующие. Убедитесь, что у вас правильные значения:

```
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
const char* mqtt_server = "YOUR_MQTT_SERVER";
const char* mqtt_user = "YOUR_MQTT_USER";
const char* mqtt_passwd = "YOUR_MQTT_PASSWORD";
const int mqtt_port = 1883; //YOUR_MQTT_PORT
```

Модуль реле подключен к GPIO 12, DHT22 к выводу 4, а светодиоды к GPIO 13 и GPIO 15:

```
#define RELAY_PIN  12
#define DHTTYPE DHT22
#define DHTPIN  4
#define GREEN_LED 15
#define RED_LED 13
```

Наши глобальные объекты и значения по умолчанию для смещения и желаемой температуры:

```
WiFiClient espClient;
PubSubClient client(espClient);
DHT dht(DHTPIN, DHTTYPE, 11);
long lastMsg = 0;
float offset_temp = 0.4;
float desired_temp = 22.0;
float humidity, temp_f;  // Значения считываются с датчика
```

gettempera () получает температуру и влажность от датчика DHT22 и сохраняет их в глобальных переменных humidity - влажности и temp_f. Если вам нужна температура в градусах Фаренгейта, вызовите функцию dht.readTemperature () с параметром true, например dht.readTemperature (true) ;:

```
void gettemperature()
{
  int runs=0;
  do {
       temp_f = dht.readTemperature(false);
       humidity = dht.readHumidity();

       if(runs > 0)
       {
           Serial.println("##Failed to read from DHT sensor! ###");
       }
//       Serial.println(String(temp_f ).c_str());
//       Serial.println(String(humidity ).c_str());
       runs++;
    }
    while(isnan(temp_f) && isnan(humidity));
}
```

В части настройки выводы реле и светодиодов устанавливаются как ВЫХОД, а зеленый светодиод горит по умолчанию при включении питания, как показано ниже:

```
void setup() {
  pinMode(RELAY_PIN, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(RED_LED, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);
  digitalWrite(GREEN_LED, HIGH);
  digitalWrite(RED_LED, LOW);

  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

  if(SPIFFS.begin())
  {
    Serial.println(F("File system was mounted."));
//проверяем, есть ли у нас желаемая температура, отличная от установленной по умолчанию
File f = SPIFFS.open("/config_temp.txt", "r");
    if (!f)
    {
      //теперь читаем содержимое файла
```

```
      String s=f.readStringUntil('\n');
      Serial.println(s);
      desired_temp = s.toFloat();
      //закрываем файл сейчас
      f.close();
    }
    else
      Serial.println(F("Failed to open my_file.txt"));
  }
}
```

Подключитесь к сети Wi-Fi с предоставленными учетными данными, как показано ниже:

```
void setup_wifi() {

  delay(10);
  // Начнем с подключения к сети Wi-Fi
  Serial.println();
  Serial.print(F("Connecting to "));
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED)
  {
    WiFi.begin(ssid, password);

    Serial.print(".");
    delay(5000);
  }
  Serial.println(F("WiFi connected"));
  Serial.println(F("IP address: "));
  Serial.println(WiFi.localIP());
}
```

Ниже приводится callback - функция обратного вызова, которая запускается, когда новое сообщение MQTT получено на подписанной теме thermostart / set:

```
void callback(char* topic, byte* payload, unsigned int length)
{
  Serial.print(F("Message arrived ["));
  Serial.print(topic);
  Serial.print(F("] "));

  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
```

```
  char rxj[20];
  int i;
  for(i=0;i<length;i++)
  {
    rxj[i] = payload[i];
  }

  File my_file = SPIFFS.open("/config_temp.txt", "w+");
  if (!my_file) {
      Serial.println("file open failed");
  }
  Serial.println(F("Writing to config_temp.txt "));
  //выводим что-нибудь в my_file.txt
  my_file.println(String(rxj).c_str());
  //сей час закрываем файл
  my_file.close();
  desired_temp = String(rxj).toFloat();
}
```

Повторно подключитесь к серверу MQTT в случае потери некоторых поддерживающих фреймов, как показано в следующем коде:

```
void reconnect() {
  // Цикл, пока мы снова не подключимся
  while (!client.connected()) {
    Serial.print(F("Attempting MQTT connection..."));
    if (client.connect("ESP8266Client", mqtt_user, mqtt_passwd))
    {
      Serial.println(F("connected"));
      client.subscribe("thermostat/set");
    } else {
      Serial.print(F("failed, rc="));
      Serial.print(client.state());
      Serial.println(F(" try again in 5 seconds"));
      // Подождите 5 секунд перед повторной попыткой
      delay(5000);
    }
  }
}
```

Функция цикла loop - это то место, где существует наша логика для срабатывания реле и светодиодов.Каждые две секунды она считывает температуру и сравнивает ее с желаемой как показано ниже:

```
void loop()
{

  gettemperature();
  if (!client.connected()) {
```

```
      reconnect();
    }
    client.loop();

    long now = millis();
    if (now - lastMsg > 2000) {
      lastMsg = now;

      if((float)desired_temp - offset_temp >= (float)temp_f)
      {
        //Serial.println(F("Start heating..."));
        digitalWrite(RELAY_PIN, HIGH);
        digitalWrite(GREEN_LED, LOW);
        digitalWrite(RED_LED, HIGH);
      }
      else if((float)desired_temp + offset_temp <= (float)temp_f)
      {
        //Serial.println(F("Stop heating..."));
        digitalWrite(RELAY_PIN, LOW);
        digitalWrite(GREEN_LED, HIGH);
        digitalWrite(RED_LED, LOW);
      }
      client.publish("thermostat/get", String(temp_f).c_str());
    }
  }
```

Чтобы установить температуру, вы можете использовать локальную консоль вашего брокера MQTT, как мы это делали в главе 2, Создание и настройка вашего собственного сервера MQTT:

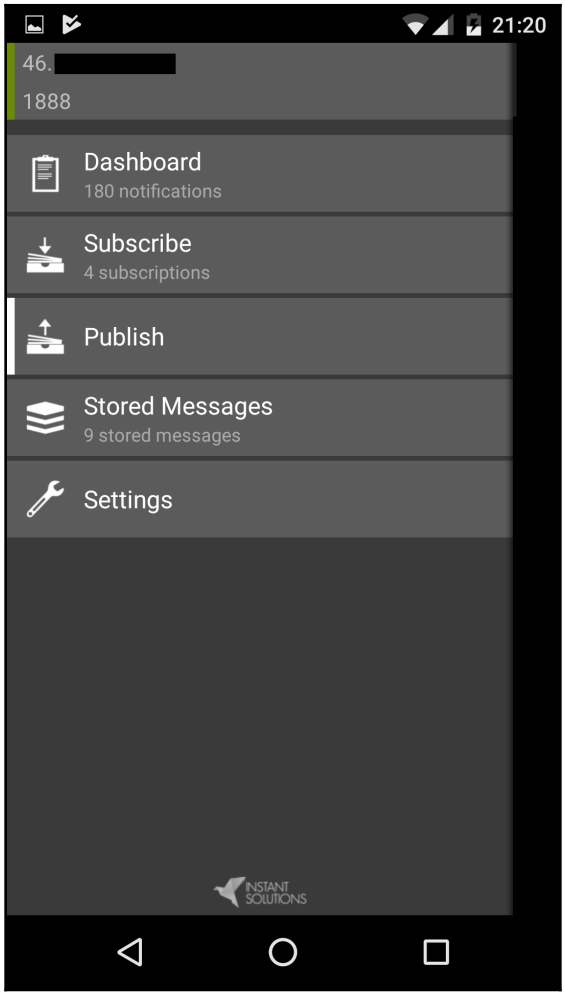
```
mosquitto_pub -t "thermostat/set" -m 28.2 -p 1883 -h YOUR_MQTT_SERVE_IP -u
YOUR_MQTT_USER -P YOUR_MQTT_PASSWORD
```

А если вы подпишетесь на тему thermostat/get - термостат / получение, вы будете получать текущую температуру в вашей комнате каждые две секунды, как показано на следующем снимке экрана:

```
27.30
27.30
27.30
27.30
27.30
27.30
27.60
27.60
27.60
27.60
27.50
27.50
27.50
27.50
27.50
```

Комнатная температура получена от брокера MQTT

Для телефонов Android есть приложение MyMQTT, и если ваш брокер доступен из Интернета, вы можете установить желаемую температуру, когда вас нет дома; вы также можете видеть текущую температуру в вашем доме.
Главное меню этого приложения MyMQTT выглядит следующим образом:

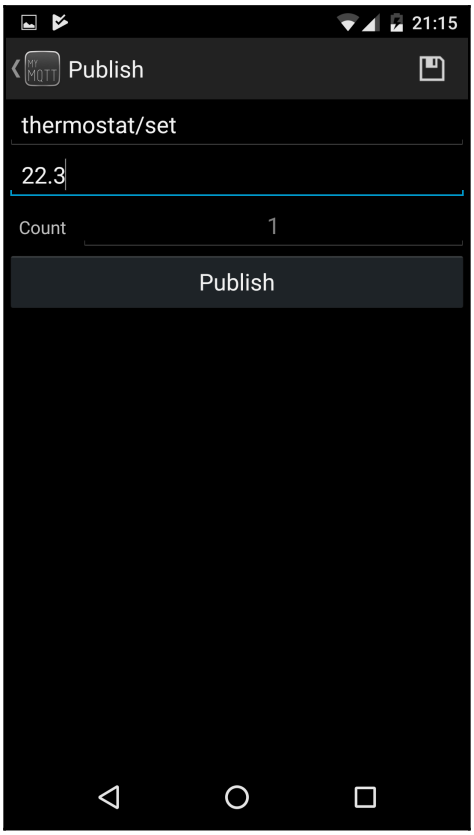


Главный экран MyMQTT

В меню « Settings -Настройки» настройте IP-адрес брокера MQTT, используемый порт MQTT, а также пользователя и пароль.
После сохранения конфигурации приложение подключится к вашему брокеру. Чтобы видеть сообщения от вашего брокера, вы можете подписаться на тему #; в этом случае вы увидите все сообщения, полученные брокером (как показано на следующем снимке экрана), или термостарт / получить, чтобы получать только температуру из вашего дома:

Получение температуры от термостата

Если вы в отъезде, вы можете дистанционно настроить термостат, опубликовав сообщение с желаемой температурой для домашнего термостата в теме термостат / установка, как показано на следующем снимке экрана:



Напечатайте желаемую температуру для управления вашим термостатом

Эта базовая функциональная система термостата научила вас использовать SPIFFS для хранения файлов и значений, отправлять и получать значения от брокеров MQTT и сохранять их в файле для считывания температуры и влажности с датчика. В качестве упражнения вы можете улучшить термостат со следующими функциями:

- Напечатать влажность
- Сохраните файл конфигурации как файл JSON

- Добавьте функцию хронотермостата, чтобы обогревать ваш дом только между запрограммированными часами
- Добавьте функцию открытия окна, чтобы прекратить обогрев дома, если температура упадет очень быстро
- Опубликуйте температуру, влажность и желаемую температуру в виде сообщения JSON через MQTT.
- Используйте обратную логику, чтобы охладить свой дом жарким летом

Резюме

В этой главе вы узнали, как сохранять данные в SPIFFS и как построить термостат и управлять им. До сих пор учетные данные для Wi-Fi были жестко закодированы в коде.

В следующей главе мы узнаем, как использовать библиотеку WiFiManager для открытия точки доступа. Получите учетные данные Wi-Fi вместе с другими данными и сохраните их на флэш-памяти, используя SPIFFS.

4

Устройства управления на ESP8266

Если вы хотите создать модуль IoT как коммерческий продукт и продавать его, вам нужно будет разрешить пользователю самостоятельно настраивать учетные данные Wi-Fi, при условии, что сервер MQTT, порт, имя пользователя и пароль будут использоваться пользователем. Вся дополнительная информация должна храниться в файловой системе SPIFFS, поэтому при запуске модуля он будет использоваться для подключения к Wi-Fi и для подключения к серверу MQTT.

В первой части этой главы мы обсудим, как создать веб-сервер для получения учетных данных сети Wi-Fi с помощью библиотеки WiFiManager, сохранить их в файле SPIFFS, а во второй части мы создадим модуль что позволяет управлять телевизором с помощью инфракрасного порта.

К концу этой главы вы получите знания, необходимые для запуска своего первого коммерческого продукта IoT.

Использование библиотеки WiFiManager

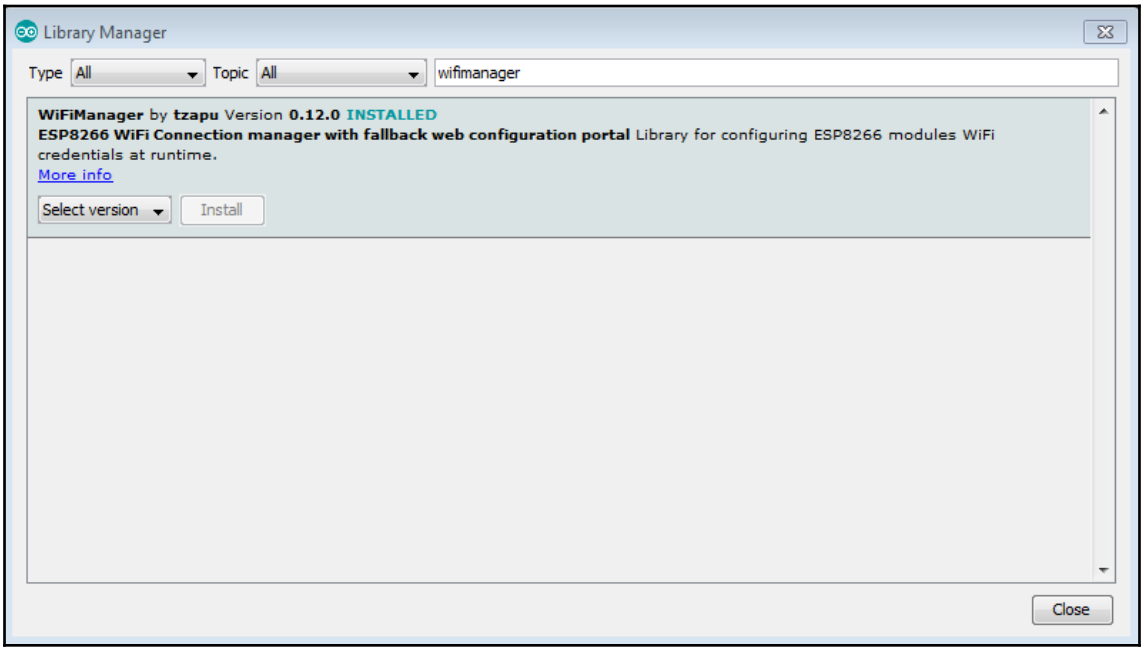
До сих пор SSID и пароль для подключения ESP8266 к сети Wi-Fi были жестко запрограммированы в скетче с использованием следующих строк:

```
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
```

Чтобы перестать использовать жестко запрограммированные значения, нам нужно сначала запустить ESP8266 в режиме AP и предоставить пользователю веб-интерфейс, обслуживаемый встроенным веб-сервером, размещенным внутри ESP8266.

Мы узнаем, как использовать библиотеку WiFiManager для открытия веб-страницы конфигурации; мы возьмем данные и сохраним их в SPIFFS, как мы делали в предыдущей главе, и будем использовать их для запуска нашего модуля в режиме станции и подключения к серверу MQTT.

Если вы не устанавливали библиотеку WiFiManager в главе 1 «Начало работы с ESP8266», вы можете сделать это сейчас, перейдя в Sketch | Включить библиотеку | Менеджер библиотек и ищите wifiManager, как показано на следующем снимке экрана:



Установка библиотеки WiFiManager

После установки библиотеки давайте посмотрим, как с ее помощью можно установить сеть Wi-Fi и пароль Wi-Fi.

Используйте следующий код, чтобы увидеть, как это выглядит и как использовать WiFiManager:

```
#include "FS.h"

#include <ESP8266WiFi.h>
#include <DNSServer.h>#include
<ESP8266WebServer.h>#include
<WiFiManager.h>
```

Если вам нужно очистить ранее сохраненные значения в SPIFFS, вы можете установить для init_esp значение true, чтобы очистить ранее сохраненные учетные данные Wi-Fi и отформатировать SPIFFS. Это может быть в том случае, если вы хотите переместить модуль ESP8266 в другую сеть Wi-Fi. После вы прошиваете код, функция setup () отформатирует SPIFFS. Затем измените значение обратно на false, чтобы вы не форматировали флэш каждый раз при запуске ESP8266:

```
boolean init_esp = false;
void setup()
{
  Serial.begin(115200); delay(10);

  if(init_esp)
  {
    SPIFFS.format();
    WiFi.disconnect();
  }
```

Создайте экземпляр объекта wifiManager. Это запустит ESP в режиме точки доступа и адаптивный портал, который перенаправит вас на веб-страницу конфигурации:

```
  WiFiManager wifiManager;
```

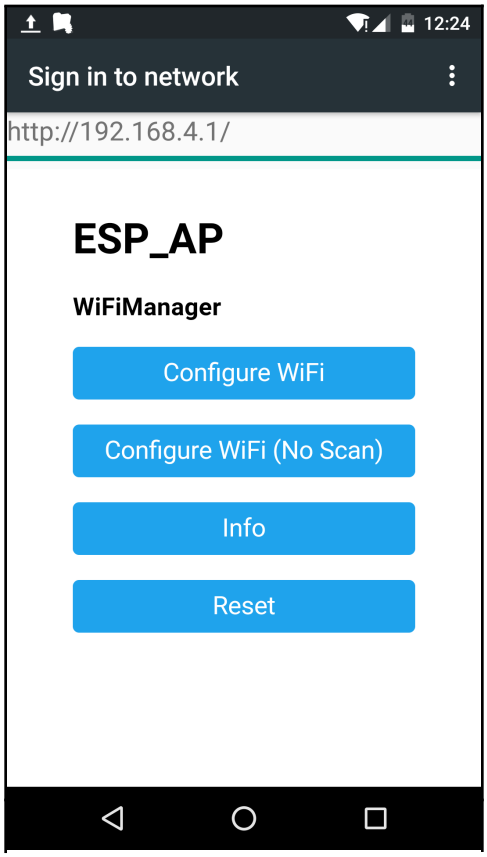
Установите время ожидания до 240 секунд:

```
  wifiManager.setConfigPortalTimeout(240);
  if (!wifiManager.autoConnect("ESP_AP", "changeit"))
  {
    Serial.println(F("Failed to connect. Reset and try again..."));
    delay(3000);
    //сбросить и повторить попытку
    ESP.reset();
    delay(5000);
  }
```

На этом этапе мы подключены к сети Wi-Fi после того, как мы выбрали сеть Wi-Fi и пароль в веб-браузере:

```
  //если вы попали сюда, значит, вы подключились к Wi-Fi
  Serial.println(F("Connected to Wifi."));
  Serial.print(F("My IP: "));
  Serial.println(WiFi.localIP());
}

void loop() {
    //добавляем ваш код для loop()
}
```

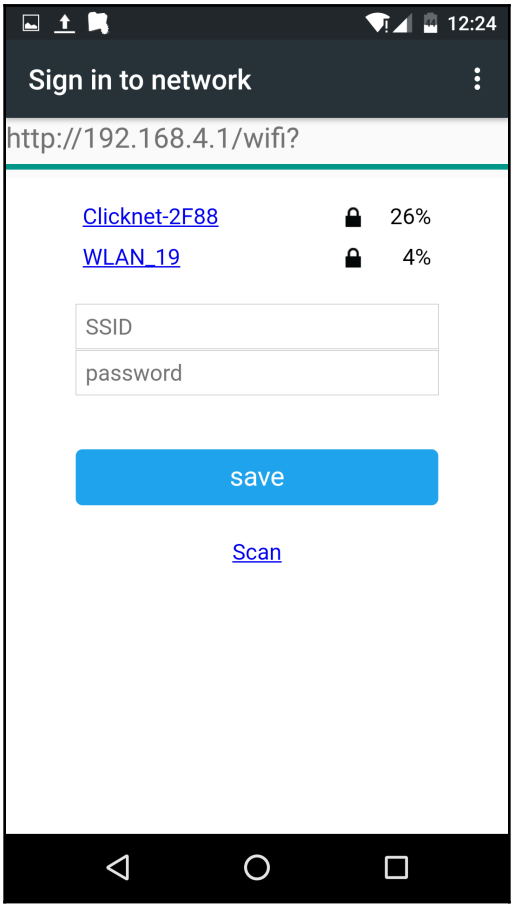
После того, как вы скомпилируете и запрограммируете ESP8266 с этим кодом, возьмите свой телефон и поищите окружающие сети Wi-Fi. Вы увидите сеть с именем ESP_AP. Подключитесь к ней, введите пароль для смены и нажмите«Войти в сеть». Если вы не видите это сообщение, перейдите в свой веб-браузер и попробуйте получить доступ к любой ссылке или введите 192.168.4.1 в адресной строке.
Вы будете перенаправлены на эту страницу:



Главный экран WiFi Manager

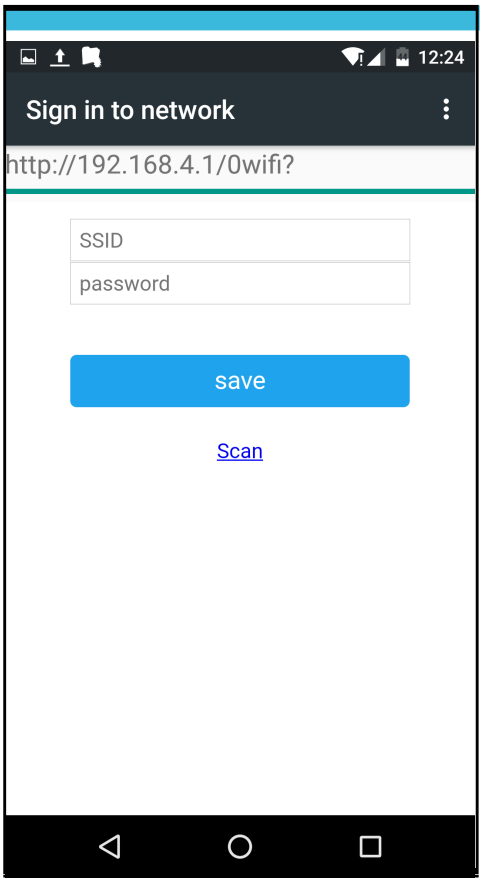
На этой странице вы увидите заданное имя точки доступа, ESP_AP и четыре кнопки, которые описаны в следующем списке:

- **Настройка Wi-Fi**: просканируйте окружающие сети Wi-Fi, и вы увидите их список вместе с мощностью их сигнала. Если у вас несколько сетей Wi-Fi, вы можете выбрать наиболее мощную, как показано на следующем снимке экрана:



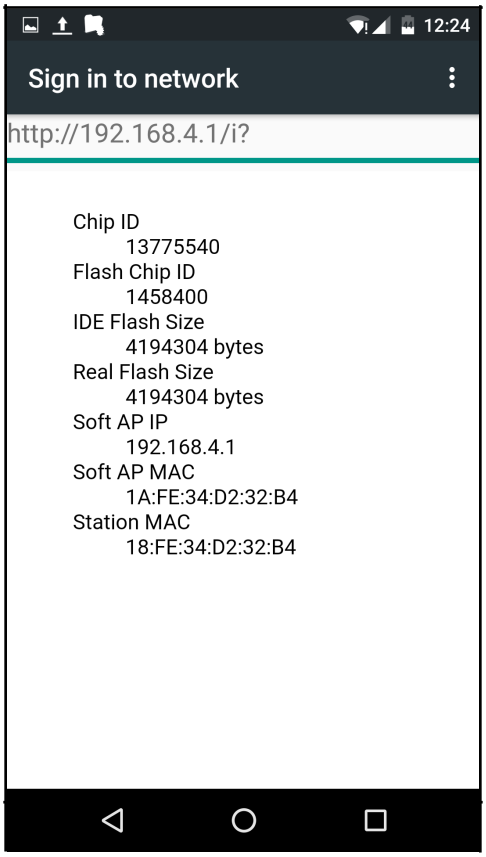
Конфигурация со сканированием

- **Configure WiFi (No Scan)**: если ваша сеть Wi-Fi имеет скрытый SSID или ваша сеть Wi-Fi не подключена к сети, вы можете ввести имя сети и пароль, пропуская функцию сканирования, как показано на следующем снимке экрана:



Конфигурация без сканирования

- **Info**: как показано на следующем снимке экрана, эта страница будет считывать идентификатор микросхемы, MAC-адрес для точки доступа или режим станции ESP8266, а также подробные сведения о флэш-памяти, такие как размер флэш-памяти и ее идентификатор:



Информация об ESP и пламени

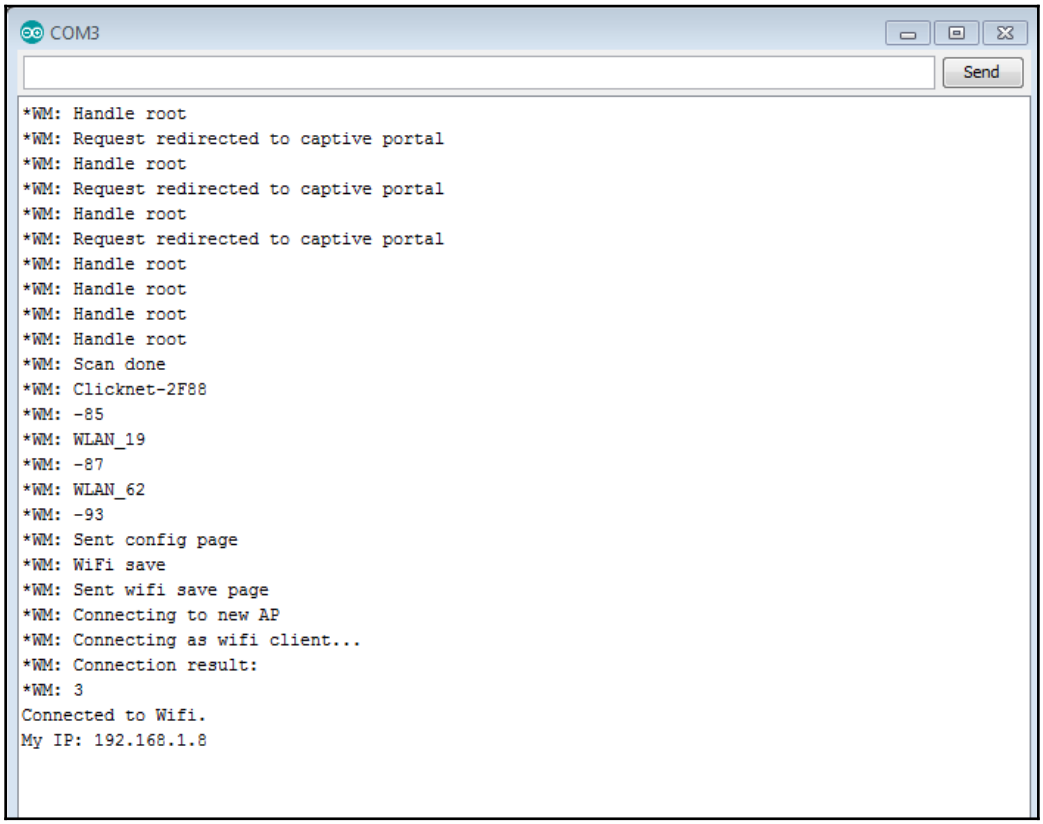
- **Reset**: нажатие кнопки **Reset** сбрасывает микросхему ESP8266. В последовательной консоли из Arduino IDE после запуска ESP вы увидите некоторую отладочную информацию о WiFiManager. Если вам нужна дополнительная информация, добавьте в свой скет следующую строку:

```
wifiManager.setDebugOutput(false);
```

- После строки:

```
WiFiManager wifiManager;
```

- После выбора сети Wi-Fi и ввода пароля на выходе в последовательной консоли будет показано, что ESP подключен к Wi-Fi; он также покажет IP-адрес, назначенный вашим маршрутизатором, как показано на следующем снимке экрана:



Вы можете добавить аппаратную кнопку, подключенную к GPIO, и при ее нажатии учетные данные Wi-Fi могут быть удалены, флеш-память может быть отформатирована или некоторые файлы из нее могут быть удалены; таким образом, вам не нужно прошивать скетч, если вы хотите изменить сеть Wi-Fi или пароль Wi-Fi.

Добавление параметров на страницу настройки WiFiManager и сохранение их в файл

Пока что единственными параметрами, сохраненными на странице WiFiManager, были SSID и пароль для вашей сети. Давайте добавим другие параметры, такие как сервер MQTT, порт, пользователь, пароль и тему, и сохраним их в файле конфигурации, расположенном в SPIFFS:

```
#include <FS.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>
#include <ArduinoJson.h>
```

Определите CLIENT_ID, который позже будет использоваться для подключения к брокеру MQTT :

```
#define CLIENT_ID         "ESP_%06X"
```

Определите здесь свои значения по умолчанию; если в config.json есть разные значения, они перезаписываются:

```
char mqtt_server[40];
char mqtt_port[6] = "1883";
char mqtt_user[32];
char mqtt_pass[32];
char mqtt_topic[64];

char dev_name[50];
```

Установите значение true, если вы хотите снова инициализировать ESP:

```
boolean clean_g = false;

WiFiClient espClient;
PubSubClient client(espClient);
```

Эта функция будет запущена, когда ESP8266 получит сообщение от брокера MQTT:

```
void mqtt_callback(char* topic, byte* payload, unsigned int length) {
  char rxj[512];
  for (int i = 0; i < length; i++) {
    rxj[i] = payload[i];
  }
  Serial.println(rxj);
}

//флаг для сохранения данных
bool saveConfig = false;

//обратный вызов, уведомляющий нас о необходимости сохранения конфигурации
void saveConfigToFileFn () {
  Serial.println("Should save config");
  saveConfig = true;
}
```

Эта функция настройки выполняет всю работу: запуск WiFi Manager, сохранение введенных данных в файл config.json на SPIFFS, чтение содержимого файла каждый раз при запуске ESP8266 и предоставление вам доступа ко всем сохраненным данным:

```
void setup() {
  // поместите сюда свой установочный код, чтобы запустить его один раз:
  Serial.begin(115200); delay(10);
  sprintf(dev_name, CLIENT_ID, ESP.getChipId());

  if(clean_g)
  {
    Serial.println(F("\n\nWait...I am formatting  the FLASH !!!"));
    SPIFFS.format();
    Serial.println(F("Done!"));
    WiFi.disconnect(true);
  }
```

Если файл config.json существует, прочтите его содержимое в формате JSON и установите для переменных mqtt соответствующие значения:

```
  if (SPIFFS.begin()) {
    Serial.println("mounted file system");
    if (SPIFFS.exists("/config.json")) {
      //файл существует, читается и загружается
      Serial.println("reading config file");
      File configFile = SPIFFS.open("/config.json", "r");
      if (configFile) {
        Serial.println("opened config file");
```

```
          size_t size = configFile.size();
          // Выделяем буфер для хранения содержимого файла.
          std::unique_ptr<char[]> buf(new char[size]);

          configFile.readBytes(buf.get(), size);
          DynamicJsonBuffer jsonBuffer;
          JsonObject& json = jsonBuffer.parseObject(buf.get());
          json.printTo(Serial);
          if (json.success()) {
            Serial.println("\nparsed json");

            strcpy(mqtt_server, json["mqtt_server"]);
            strcpy(mqtt_port, json["mqtt_port"]);
            strcpy(mqtt_user, json["mqtt_user"]);
            strcpy(mqtt_pass, json["mqtt_pass"]);
            strcpy(mqtt_topic, json["mqtt_topic"]);

          } else {
            Serial.println("failed to load json config");
          }
        }
      }
    } else {
      Serial.println("failed to mount FS");
    }
```

Добавьте настраиваемые параметры в WiFiManager, чтобы они были доступны, когда пользователь захочет установить их через веб-интерфейс:

```
    WiFiManagerParameter custom_mqtt_server("server", "mqtt server",
  mqtt_server, 40);
    WiFiManagerParameter custom_mqtt_port("port", "mqtt port", mqtt_port, 5);
    WiFiManagerParameter custom_mqtt_user("user", "mqtt user", mqtt_user,
  32);
    WiFiManagerParameter custom_mqtt_pass("pass", "mqtt pass", mqtt_pass,
  32);
    WiFiManagerParameter custom_mqtt_topic("topic", "mqtt topic", mqtt_topic,
  64);

    WiFiManager wifiManager;

    wifiManager.setSaveConfigCallback(saveConfigToFileFn);

    wifiManager.addParameter(&custom_mqtt_server);
    wifiManager.addParameter(&custom_mqtt_port);
    wifiManager.addParameter(&custom_mqtt_user);
    wifiManager.addParameter(&custom_mqtt_pass);
    wifiManager.addParameter(&custom_mqtt_topic);
```

```
Start the WiFiManager Access Point with the name ESP_AP.

  if (!wifiManager.autoConnect("ESP_AP"))
  {
    Serial.println(F("failed to connect and hit timeout"));
    delay(3000);
    //reset and try again, or maybe put it to deep sleep
    ESP.reset();
    delay(5000);
  }

  Serial.println(F("WiFi is connected now..."));
```

Прочтите обновленные параметры на веб-странице:

```
strcpy(mqtt_server, custom_mqtt_server.getValue());
strcpy(mqtt_port, custom_mqtt_port.getValue());
strcpy(mqtt_user, custom_mqtt_user.getValue());
strcpy(mqtt_pass, custom_mqtt_pass.getValue());
strcpy(mqtt_topic, custom_mqtt_topic.getValue());
```

Теперь, когда мы получили все необходимые параметры с веб-страницы, сохраните их в файле config.json. Этот файл читается при запуске ESP8266:

```
if (saveConfig) {
  Serial.println("saving config");
  DynamicJsonBuffer jsonBuffer;
  JsonObject& json = jsonBuffer.createObject();
  json["mqtt_server"] = mqtt_server;
  json["mqtt_port"]   = mqtt_port;
  json["mqtt_user"]   = mqtt_user;
  json["mqtt_pass"]   = mqtt_pass;
  json["mqtt_topic"]  = mqtt_topic;

  File configFile = SPIFFS.open("/config.json", "w");
  if (!configFile) {
    Serial.println(F("failed to open config file for writing"));
  }

  json.printTo(Serial);
  json.printTo(configFile);
  configFile.close();
  //конец сохранения
}
```

Теперь, когда ESP8266 подключен к сети Wi-Fi, осталось только подключиться к брокеру MQTT с учетными данными из файла config.json и подписаться на тему:

```
  Serial.println(F("My IP address: "));
  Serial.println(WiFi.localIP());
  //подключаемся к серверу mqtt
  client.setServer(mqtt_server, atoi(mqtt_port));
  client.setCallback(mqtt_callback);
  if (client.connect(dev_name , mqtt_user, mqtt_pass))
  {
       Serial.println(F("Connected to MQTT broker"));
Serial.println(F("Subscribe to your mqtt_topic now"));
  }

}

void loop() {
  // поместите свой основной код для запуска в цикле
client.loop();
}
```

В заключение, вот что вы можете делать с библиотекой WiFiManager; создайте захватывающий портал, который позволит вам настроить ESP8266 с учетными данными Wi-Fi и настраиваемыми параметрами. Вы можете использовать эту библиотеку в любом проекте, где вы хотели бы дать пользователям возможность настраивать свои ESP с различными значениями.

ESP8266 и инфракрасная связь

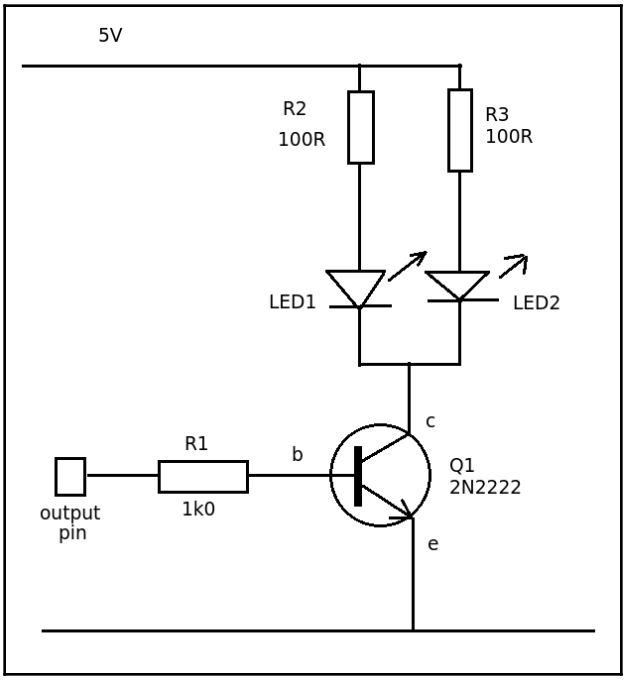
В предыдущих главах ESP866 управлял реле, считывал температуру и влажность, но не всеми приборами в доме можно управлять с помощью реле. Некоторыми можно управлять с помощью инфракрасного порта, например, телевизор или кондиционер.Теперь давайте посмотрим, как мы можем использовать ESP8266 для включения / выключения телевизора Panasonic. Это можно распространить на другие марки телевизоров, изменив адреса и значения.

Компоненты устройства

Для выполнения этого проекта вам понадобятся:

- ESP8266
- Инфракрасные светодиоды (может взять от старого пульта)
- Резисторы на 100 Ом (в зависимости от количества инфракрасных светодиодов, один на светодиод)
- Резистор 1 кОм
- Транзистор 2N2222 NPN

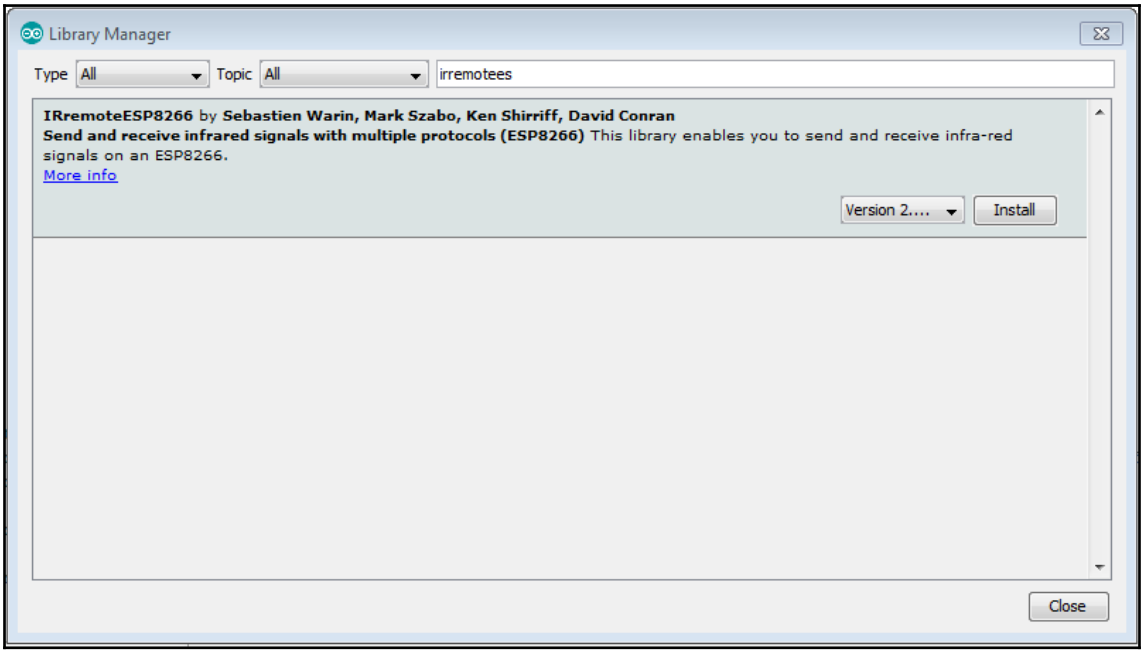
Используйте следующую схему устройства этого проекта:



Подключите output pin (см. схему) к GPIO 12 ESP8266, землю к контакту GND и 5V к 5V вашего ESP.

Программа и библиотеки для этого проекта

Сначала установим библиотеку IRremoteESP8266. Для этого перейдите в Sketch | IncludeLibrary | Менеджер библиотек ... и найдите библиотеку IRremoteESP8266, как показано на следующем снимке экрана:



После установки библиотеки и выполнения подключений к ESP давайте воспользуемся следующим кодом, чтобы отправить по теме MQTT команду на включение и выключение телевизора Panasonic. Поскольку для этого используется та же инфракрасная команда, вам просто нужно получить что-то в теме MQTT, чтобы отправить инфракрасную команду на телевизор:

```
#include <ArduinoJson.h>
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <PubSubClient.h>
#include <IRremoteESP8266.h>
#include <IRsend.h>

#define PanasonicAddress      0x4004
#define PanasonicPower        0x100BCBD  // Panasonic Power
button
```

Добавьте свои учетные данные Wi-Fi и MQTT в следующие переменные:

```
#define wifi_ssid "YOUR_WIFI_SSID"
#define wifi_password "YOUR_WIFI_PASS"

#define mqtt_server "YOUR_MQTT_SERVER"
#define mqtt_user "YOUR_MQTT_USER"
#define mqtt_password "YOUR_MQTT_PASSWORD"
#define mqtt_port 1883
```

Установите "инфракрасный" пин на GPIO12 или D6 на платах NodeMCU и WeMos и укажите тему, по которой будут приходить сообщения для включения / выключения телевизора:

```
#define IR_PIN 12
#define ir_topic "/62/ir/command"

void rx_mqtt_callback(char* topic, byte* payload, unsigned int length);

#define DEBUG false
#define Serial if(DEBUG)Serial
#define DEBUG_OUTPUT Serial

IRsend irsend(IR_PIN); // ИК-светодиод подключен к контакту
GPIOWiFiClient espClient;
PubSubClient client(mqtt_server, mqtt_port, rx_mqtt_callback,espClient);

StaticJsonBuffer<512> jsonDeviceStatus;
JsonObject& jsondeviceStatus = jsonDeviceStatus.createObject();

char dev_name[50];
char json_buffer_status[512];
char my_ip_s[16];
```

Запустите Wi-Fi и подключитесь к сети Wi-Fi:

```
void setup_wifi()
{
  delay(10);
  // Начнем с подключения к сети Wi-Fi
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(wifi_ssid);

  WiFi.mode(WIFI_STA);
  WiFi.begin(wifi_ssid, wifi_password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

Подключитесь и повторно подключитесь к брокеру MQTT. В случае ошибки пытайтесь повторно подключаться каждые 5 секунд:

```
void reconnect() {
   // Цикл, пока мы снова не подключимся
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect(dev_name, mqtt_user, mqtt_password)) {
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Подождите 5 секунд перед повторной
      попыткой
    } delay(5000);
  }
}
```

Это функция, которая будет вызываться при получении сообщения от брокера MQTT.

Получите сообщение и отправьте инфракрасные команды на телевизор:

```
void rx_mqtt_callback(char* topic, byte* payload, unsigned int length)
{
  //резервируем место для входящего сообщения
  StaticJsonBuffer<256> jsonRxMqttBuffer;
  int i = 0;
  char rxj[256];
  Serial.println(dev_name);
Serial.print(F("Topic:"));Serial.println(topic);
  for(i=0;i<length;i++)
  {
    rxj[i] = payload[i];
  }

  Serial.println(rxj);
  JsonObject& root = jsonRxMqttBuffer.parseObject(rxj);
  if (!root.success())
  {
    Serial.println(F("parseObject() failed"));
```

```
    return;
  }

  const char* device_name  = root["device_name"];
  const char* type         = root["type"];
  const char* value        = root["value"];

  Serial.println(device_name);
  Serial.println(type); Serial.println(value);

  sendIR();

  return;
}
```

Здесь мы отправляем код сообщения на телевизор. Если не получается с первого раза, попробуйте подойти ближе к телевизору. Это зависит от качества ваших ИК-светодиодов, количества инфракрасных светодиодов и окружающего освещения:

```
void sendIR()
{
    int i = 0;
    Serial.print("sendIR for 2 sec");
    for(i=0;i<20; i++)
    {
                    irsend.sendPanasonic(PanasonicAddress,PanasonicPower);
  // Оно должно включать и выключать ваш телевизор
 delay(100);
    }
}
void setup()
{
  delay(1000);
  irsend.begin();
  Serial.begin(115200);
  sprintf(dev_name, "ESP_%d", ESP.getChipId());
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  client.connect(dev_name, mqtt_user, mqtt_password);
  client.subscribe(ir_topic);
  if (!client.connected())
  {
    reconnect();
  }
}
void loop() {
  client.loop();
}
```

Теперь, используя приложение MyMQTT для Android, попробуйте отправить команду ir_topic для включения и выключения телевизора. Как вы, наверное, уже заметили, я не использовал WiFiManager для настройки ESP8266, но это очень хорошее упражнение для вас, чтобы завершить главу и проект. Изучите библиотеку LIRC из Linux и попробуйте найти другие устройства, которыми можно управлять с помощью инфракрасного порта и ESP8266.

Резюме

В этой главе вы узнали, как использовать WiFiManager для настройки Wi-Fi и добавления параметров в файл конфигурации, сохранения полученных значений в SPIFFS, считывания значений во время запуска и использования их для подключения к брокеру MQTT. Наконец, вы отправили инфракрасные команды для управления дополнительными устройствами, такими как кондиционер, телевизор и т. д.

5

Использование ESP8266 для создания системы безопасности

Мы все заинтересованы в том, чтобы знать, что происходит дома, когда нас нет. Либо речь идет о безопасности, либо о критических элементах, таких как газ, огонь или вода. Знание того момента, когда что-то происходит, жизненно важно для минимизации возможного ущерба. В этой главе мы сосредоточимся на датчике PIR, но те же принципы применимы и к другим датчикам, таким как датчики влажности, газа или дыма.

Пассивный инфракрасный датчик

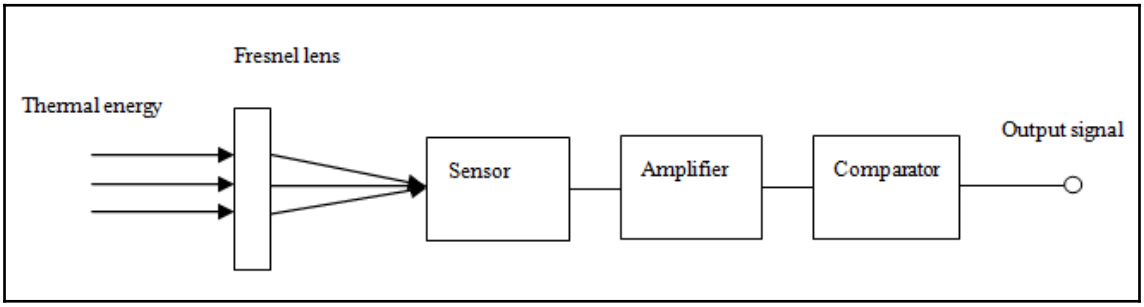
PIR - наиболее распространенный датчик, используемый в системах внутренней и наружной сигнализации. Он также используется в автоматических дверях и системах автоматического освещения.

Как работают PIR

Функциональность датчика PIR основана на излучении, исходящем от человеческого тела. Объекты выделяют тепло в виде инфракрасного излучения, и к таким объектам относятся животные и человеческое тело, излучение которых наиболее сильно на длине волны 9,4 мкм.

Когда человек проходит перед датчиком, температура в точке обзора датчика PIR изменяется с фонового значения на значение для человека. Датчик обнаруживает это изменение инфракрасного излучения и изменяет свое выходное напряжение, сигнализируя об обнаружении.

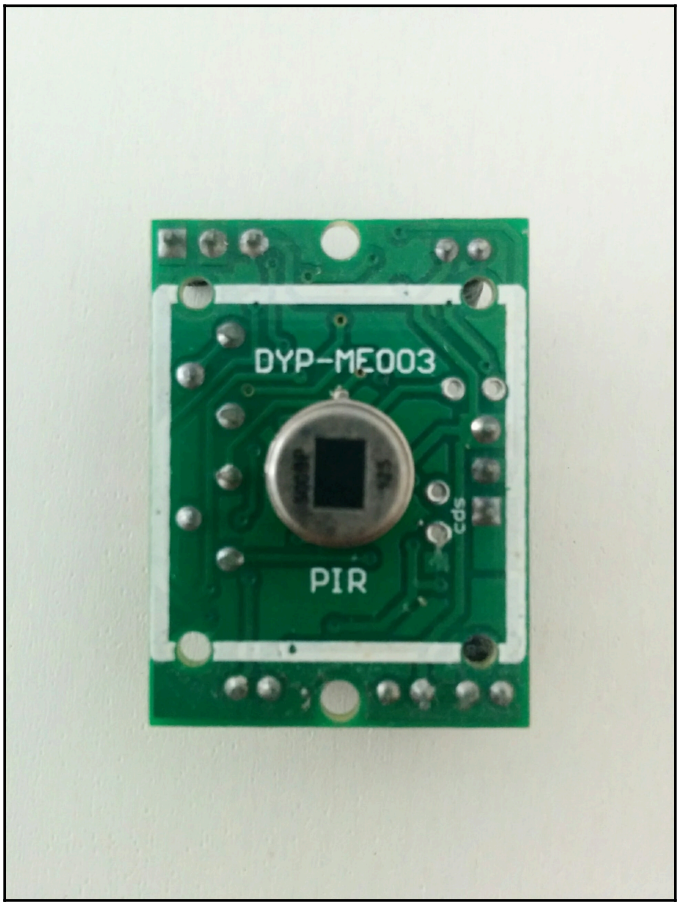
Для увеличения чувствительности ИК-сенсора перед ним установлена линза Френеля. Датчик на самом деле представляет собой FET транзистор с выводом истока, подключенным к земле. Мы можем видеть эту устройство на следующем рисунке:



На следующем изображении мы видим линзу Френеля из пластика:

Сам датчик расположен под купольной линзой Френеля, а перед ним расположен инфракрасный фильтр.

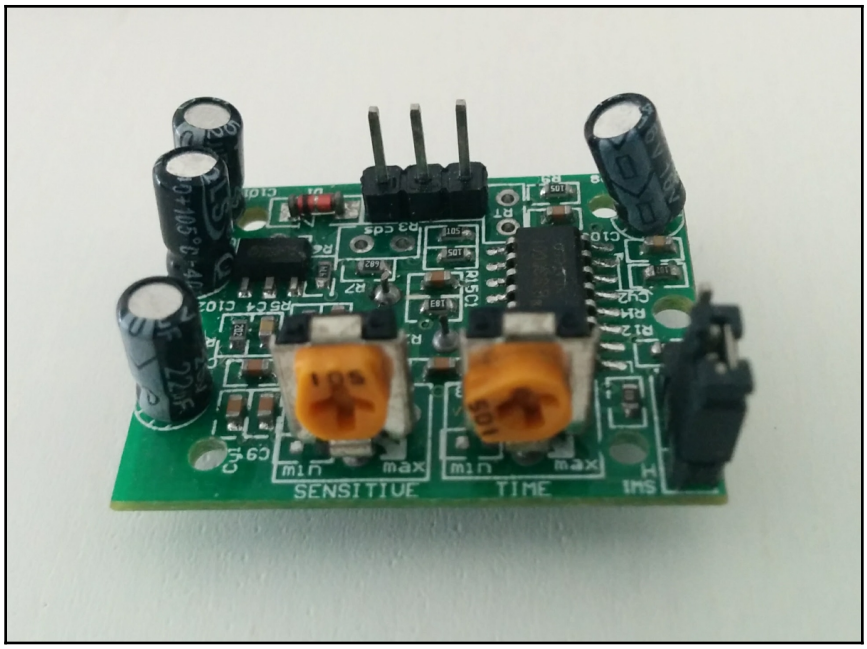
Здесь мы видим DYP-ME003:



Представленный здесь модуль DYP-ME003 стоит около одного доллара за модуль. Основные характеристики этого модуля:

- Датчик угла <100 градусов угла конуса
- Размеры платы 32 мм x 24 мм
- Рабочая температура от -17 до 70 градусов Цельсия

- Выходной уровень высокий 3,3 В / низкий 0 В
- Рабочее напряжение 4,5-20 В
- Регулируемое время задержки от 5 до 200 секунд
-



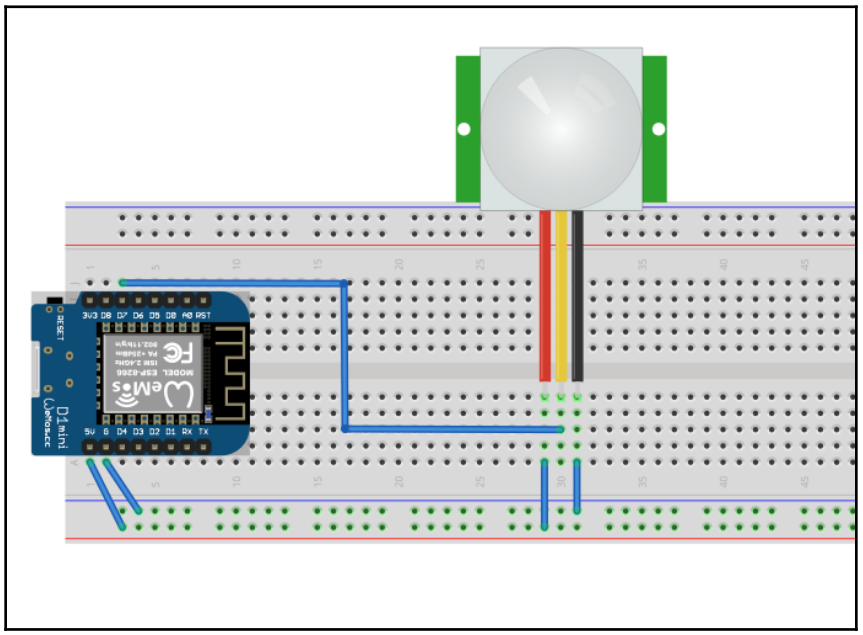
На модуле есть два переменных резистора, которые позволяют изменять два параметра:

- Sensitivity - Чувствительность: это расстояние, на котором датчик обнаруживает человека. Дальность обнаружения может быть установлена от 3 до 7 метров. Поворот движка подстроечника чувствительности по часовой стрелке снижает чувствительность датчика.
- Time - Время устанавливает, как долго вывод будет оставаться на высоком уровне после обнаружения движения. Время может составлять от 5 секунд до 5 минут. Поворот движка регулировки времени вывода по часовой стрелке увеличит задержку времени.

Тестирование модуля PIR

А пока давайте протестируем модуль PIR, чтобы убедиться, что он работает. Для этого вам понадобятся:

- Модуль ESP8266 (NodeMCU, WeMos или другая плата)
- Модуль PIR
- Макетная плата
- Перемычки



Для программной части используйте следующий скетч:

- Определите PIR, поскольку он будет подключен к выводу D7, установите для pirState значение LOW, предполагая, что PIR выключен, и установите текущее состояние на 0:

```
#define PIN_PIR  D7int
pirState = LOW;int
current_value = 0;
```

- Во время настройки мы объявим пин D7, к которому подключен вход PIR:

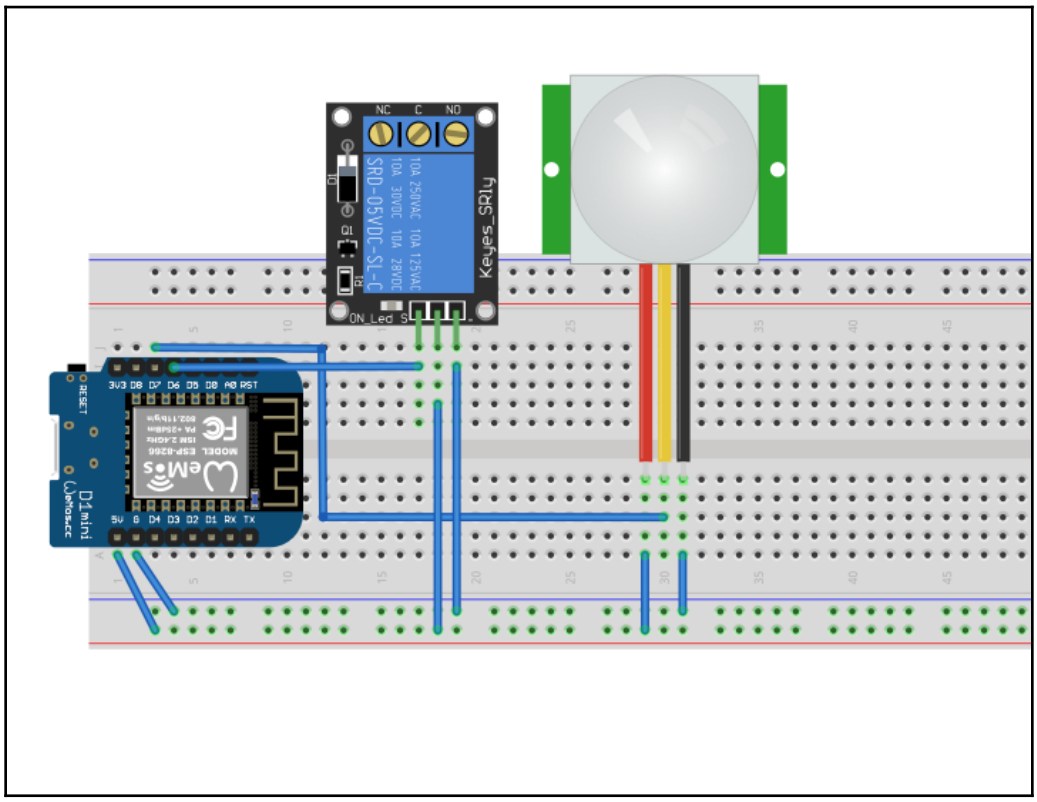
```
void setup() {
  pinMode(PIN_PIR, INPUT);
  Serial.begin(115200);
}
```

Теперь постоянно отслеживайте PIN_PIR и ожидайте изменения его состояния. Если произойдет изменение, распечатайте на последовательной консоли сообщение: датчик обнаружил движение! и когда оно истечет - сообщение: Движение закончилось ... и измените pirState на LOW:

```
void loop(){
  current_value = digitalRead(PIN_PIR);
  if (current_value == HIGH) {
    if (pirState == LOW) {
      Serial.println(F("Sensor detected motion!"));
      // Мы хотим печатать только изменение вывода, а не состояние
      pirState = HIGH;
    }
  }
  else
  {
    if (pirState == HIGH){
      Serial.println(F("Motion ended..."));
      pirState = LOW;
    }
  }
}
```

Как мы видим, этот скетч предназначен только для тестирования PIR, чтобы вы могли поиграться и настроить чувствительность и время, которые вы хотите для своего проекта. Чтобы использовать этот датчик только для включения или выключения света, вам необходимо добавить реле.

Если вы хотите добавить реле для включения или выключения лампы при входе в комнату, вам нужно добавить реле и немного изменить код:



Теперь код для добавления реле будет:

```
#define PIN_PIR  D7  //GPIO13
#define RELAY_PIN D6 //GPIO12
int pirState = LOW;
int current_value = 0;
```

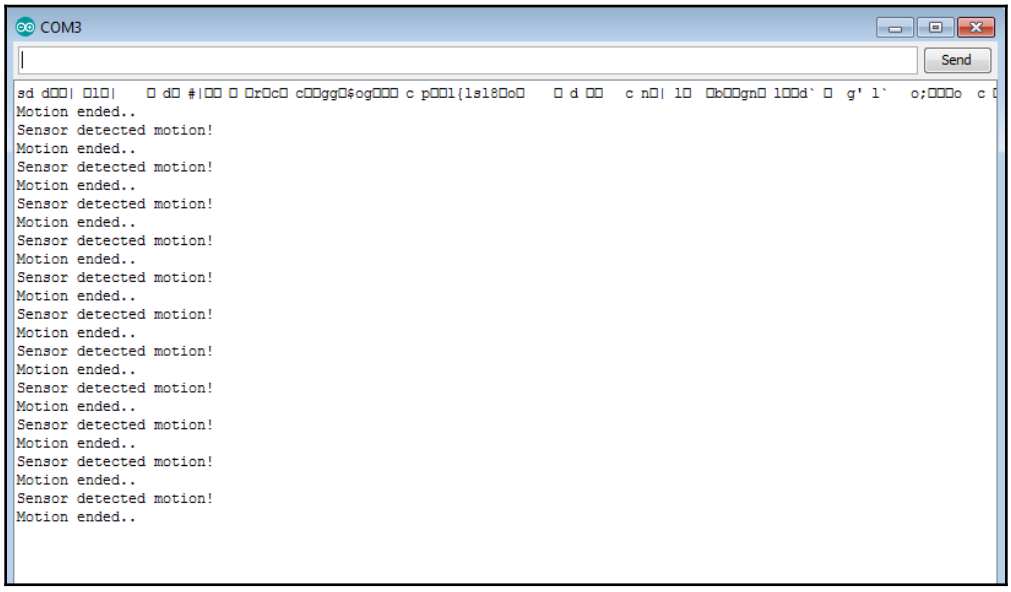
Просто установите RELAY_PIN как выходной контакт:

```
void setup() {
  pinMode(PIN_PIR, INPUT);
  pinMode(RELAY_PIN, OUTPUT);
  Serial.begin(115200);
}
```

Если движение обнаружено, активируйте реле и включите лампу. По окончании движения выключите лампу через реле:

```
void loop(){
  current_value = digitalRead(PIN_PIR);
  if (current_value == HIGH) {
    if (pirState == LOW) {
      Serial.println(F("Sensor detected motion!"));
      // Мы хотим печатать только изменение вывода, а не состояние
      pirState = HIGH;
      digitalWrite(RELAY_PIN,HIGH);
    }
  }
  else
  {
    if (pirState == HIGH){
      Serial.println(F("Motion ended.."));
      pirState = LOW;
      digitalWrite(RELAY_PIN,LOW);
    }
  }
}
```

После компиляции и прошивки софта попробуйте походить перед датчиком.
На последовательном мониторе вы увидите сообщения об обнаружении движения, а также услышите щелчки реле. Если вы подключили лампу к реле, лампа должна загореться или погаснуть:



Подключение модуля PIR к Интернету

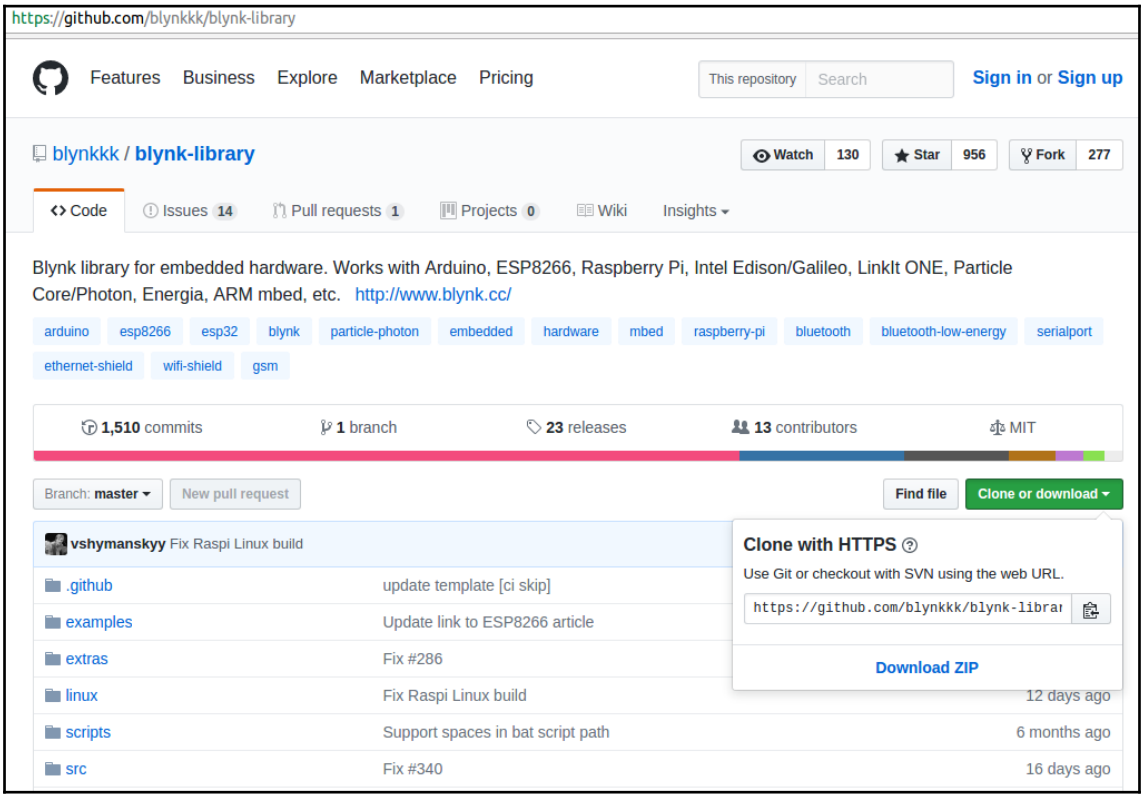
Все, что мы делали до сих пор, также может быть достигнуто с помощью Arduino, так в чем же польза от ESP8266? Вы получите уведомление по электронной почте и сигнал на свой телефон, когда в доме обнаружено движение. Кроме того, позже вы можете совместить обнаруженное движение с включением света, телевизора или включения сирены.

Для этого мы будем использовать библиотеку с именем blynk, которую можно скачать по следующей ссылке:

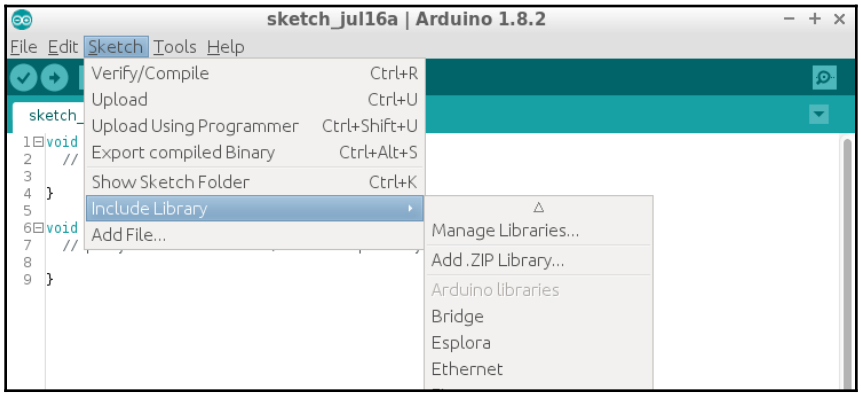
https://github.com/blynkkk/blynk-library или их мобильное приложение.

Теперь посмотрим, как его настроить:

1. Сначала загрузим и установим библиотеку:



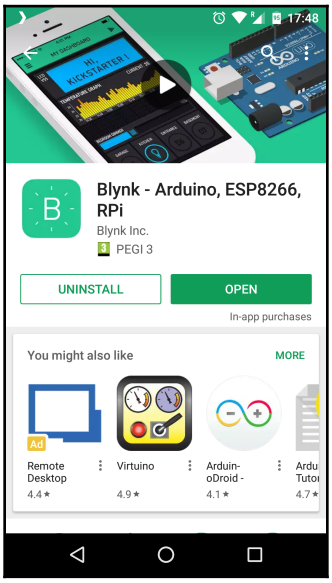
2. Нажмите кнопку "Загрузить ZIP". Затем перейдите в Sketch | Включить библиотеку | Добавить библиотеку .ZIP ...:



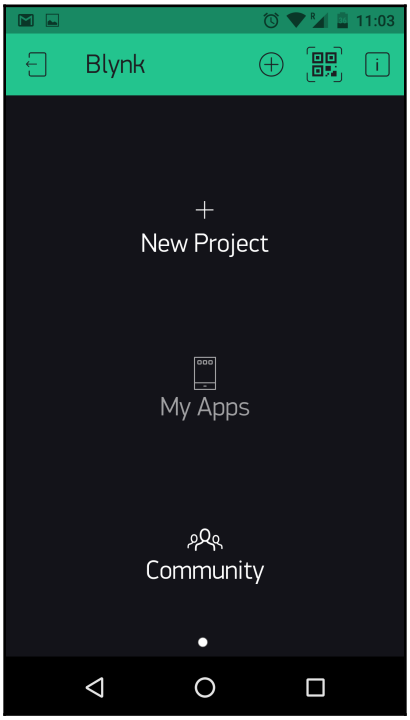
3. Найдите загруженный ZIP-файл и установите библиотеку.

Пришло время установить приложение для Android, чтобы мы могли использовать его позже:

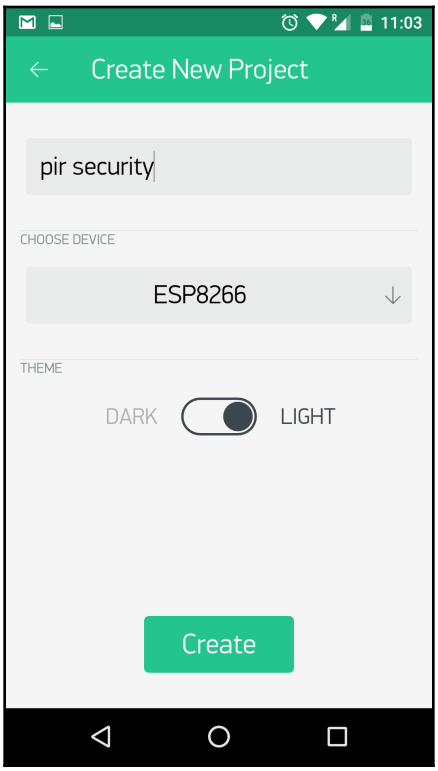
1. Перейдите в Google Play Store и найдите Blynk - Arduino, ESP8266, Rpi и установите приложение:



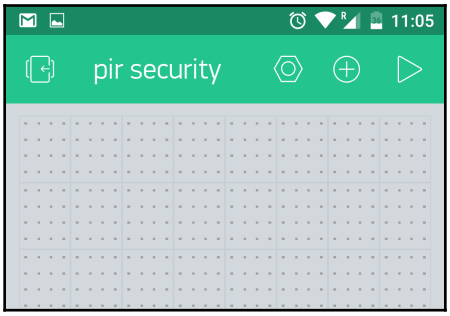
2. Давайте сначала сосредоточимся на приложении для Android. Откройте приложение и создайте учетную запись в приложении Blynk:



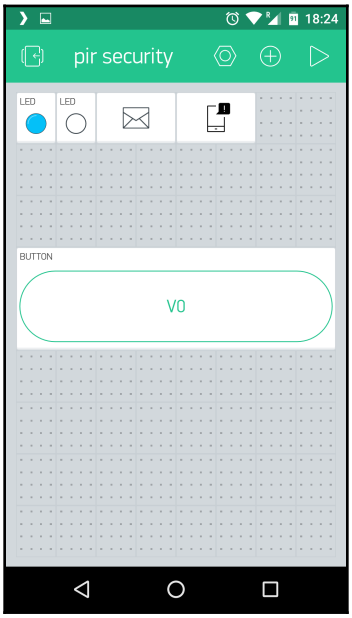
3. После этого нажмите « New Project - Новый проект», установите имя проекта, тип платы на ESP8266, тему вашего приложения - dark (темную) или light (светлую) и продолжите, нажав кнопку«Create - Создать». Проект создан, и на ваш адрес электронной почты, который использовался для регистрации, будет отправлено письмо с токеном. Кроме того, этот токен можно будет найти позже на вкладке Project Settings:



4. Теперь проект создан, токен находится в электронном письме, и мы находимся в приложении на главном экране, откуда мы можем добавлять к нему светодиоды и уведомления. Для этого нажмите значок +:

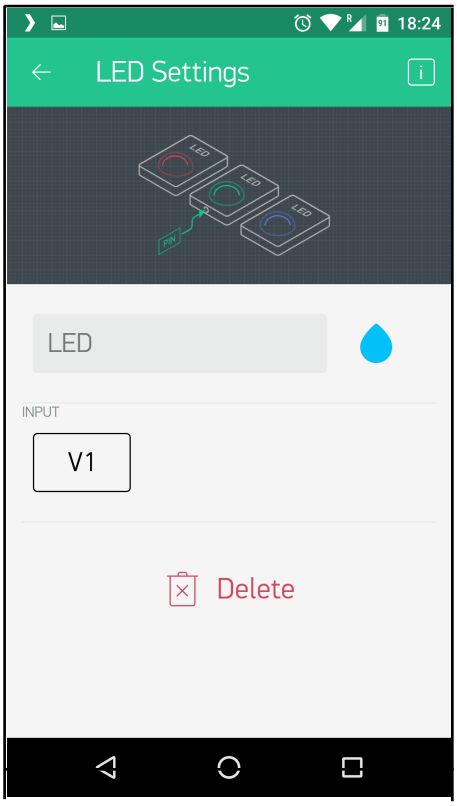


5. После того, как вы добавили все элементы, главный экран вашего приложения должен выглядеть так:

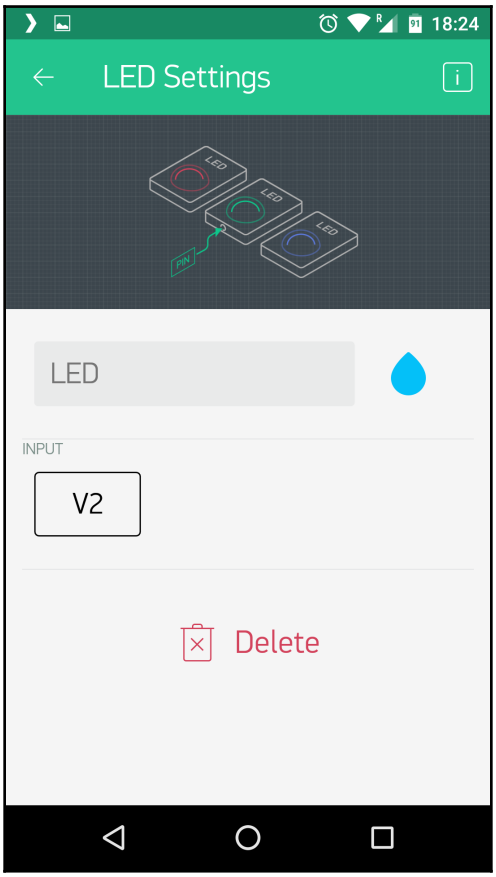


6. Теперь пришло время настроить светодиоды для некоторых виртуальных контактов (V1 иV2), добавить адрес электронной почты, на который будет отправлено уведомление, и настроить звук, который будет воспроизводиться вашим телефоном, если устройство переходит в автономный режим или новое уведомление отправляется облачным сервером Blynk:

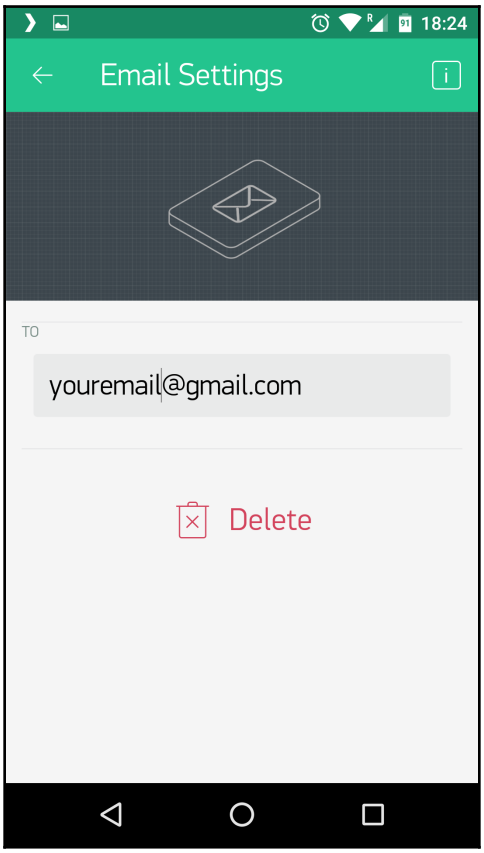
   Виртуальный LED 1:



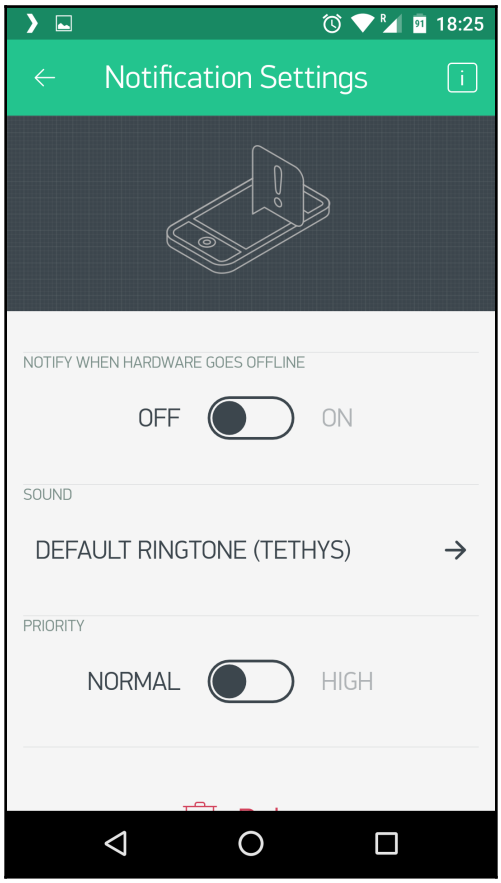
Виртуальный LED 2:



Настройки электронной почты:



Настройки уведомлений (Notification):



В этот момент приложение готово к запуску. Если вы хотите пропустить все эти шаги по созданию элементов, вы можете отсканировать следующий QR-код, который добавит и настроит все за вас. Вам просто нужно будет изменить адрес электронной почты в элементе уведомления по электронной почте.

Защитный код ESP8266 PIR

Чтобы модуль ESP8266 и PIR работал с определенным приложением Blynk, он включает заголовок Blynk, простой таймер и определяет BLYNK_PRINT как последовательный для целей отладки. Вам нужно запустить следующий код:

```
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#include <SimpleTimer.h>
```

Перейдите в мобильное приложение, а затем в настройки проекта и отправьте себе app_token по электронной почте:

```
char app_token[] = "TOKEN_FROM_EMAIL";
SimpleTimer timer;
char ssid[] = "YOUR_WIFI_NETWORK";
char pass[] = "YOUR_PASSWORD";
int state;
int counter=0;
int flag=1;
WidgetLED led1(V1);
WidgetLED pirLED(V2);
```

Это функция timer, которая будет вызываться каждую секунду для проверки состояния вывода PIR. Вся логика для приложения ESP8266 находится в этой функции:

```
void timer_ev()
{
  counter = counter+1;
  if(counter==5)
  {
    pirLED.off();
    counter=0;
    flag = 1;
  }
  int pirStatus = digitalRead(D7);
  if (pirStatus)
  {
    if (flag == 1)
    {
        Serial.println(F("Sensor detected motion!"));
        Blynk.email("bcatalin@gmail.com","Subject: Security alert!",
"Movement detected!");
        Blynk.notify("Security alert! Someone is in the house!");
        digitalWrite(D4, LOW);
        led1.on();
        pirLED.on();
        flag=2;
    }
  }
}
```

Настройте Blynk с токеном мобильного приложения, запустите соединение Wi-Fi с помощью ssid и pass для роутера. Вы можете изменить настройку Wi-Fi с помощью WiFiManager, чтобы не кодировать жестко значения в коде . Кроме того, вы можете добавить библиотеку MQTT PubSubClient и отправить Mosquitto сообщение JSON по определенной теме, чтобы другие устройства подписались на получение сообщения. Например, вы создаете сирену, которая будет подписываться на тему тревоги, где модуль будет публиковать тревожное событие. При получении сообщения о тревоге сирена может включить звуковой сигнал. Если система освещения подписывается на тему тревоги при получении сообщения тревоги, она может включить все огни в доме:

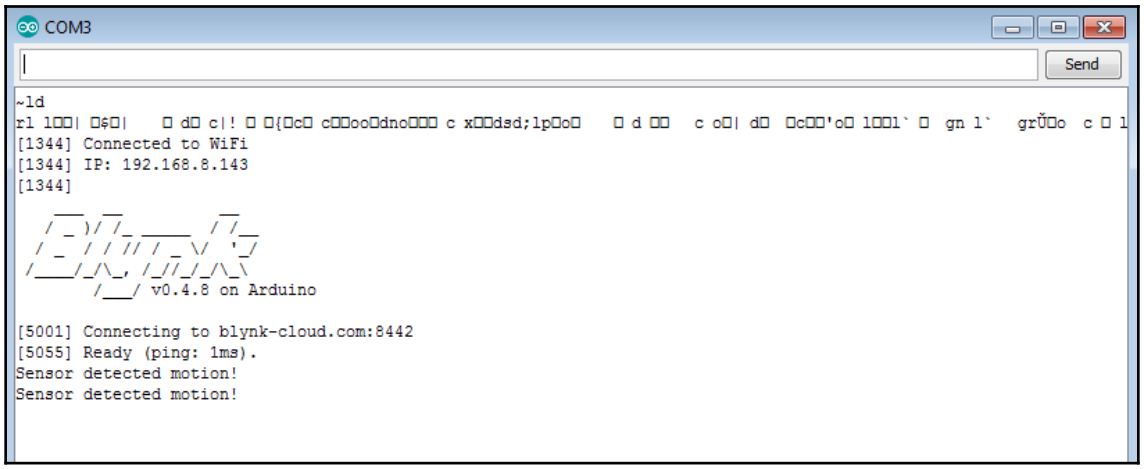
```
void setup()
{
  Serial.begin(115200);
  Blynk.begin(app_token, ssid,
  pass);pinMode(D4, OUTPUT);
  timer.setInterval(1000L, timer_ev);
}
BLYNK_WRITE(V0)
{
```

```
    state = param.asInt();
    if (state == 1){
      digitalWrite(D4, LOW);
      led1.on();
    }
    else {
      digitalWrite(D4, HIGH);
      led1.off();
    }
  }
```

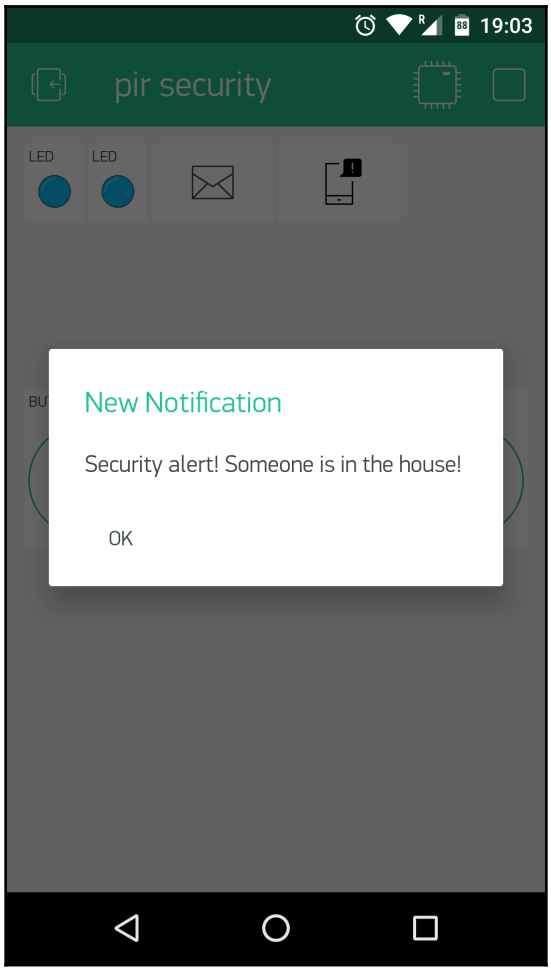
В функции loop (цикл) просто запустите timer и Blynk:

```
  void loop()
  {
    Blynk.run();
    timer.run();
  }
```

Теперь подойдите к датчику, и вы начнете получать сообщения от приложения Blynk на свой мобильный телефон. Также проверьте Serial Monitor на наличие сообщений:



И сообщение, полученное на телефон, будет:



Обратите внимание, что интервал отправки сообщений на Blynk составляет 15 секунд, а также максимальное количество электронных писем для Gmail составляет 500 в день.

Резюме

Теперь вы можете создать систему безопасности в кратчайшие сроки, но безопасность не ограничивается датчиками PIR, вы можете добавить к ней другие датчики, такие как CO 2 или газа, или датчики для обнаружения утечек воды или дыма, и вы можете создать зависимость между ними. Например, если у вас в доме прорвало трубу и датчик утечки воды обнаруживает воду, вы будете уведомлены в приложении Blynk, но также другой ESP8266, который подписывается на тему утечки, может закрыть главный клапан, поэтому ущерб будет ограничен временем, когда вы прибудете домой.

6

Защита ваших данных

В главе 2 «Создание и настройка собственного сервера MQTT» вы узнали о протоколе MQTT, о том, как создается тема, а также о том, как установить и настроить брокер mosquitto. В то время вы использовали локальный файл конфигурации, чтобы добавить пользователя и пароль, которые будут использоваться в качестве метода аутентификации для локального брокера. Как насчет времени, когда пакеты передаются от вашего модуля ESP8266 к экземпляру облачного mosquitto? Чтобы зашифровать пакеты, вам необходимо включить от mosquitto и отправлять зашифрованные пакеты с вашего ESP8266.

Включение шифрования на mosquitto

Чтобы включить шифрование на mosquitto, вам сначала нужно иметь сертификаты. Вы можете купить их у компании, которая выпускает сертификаты, или вы можете создать их самостоятельно в качестве сертификатов с собственной подписью.

Установка пакета openssl

Во-первых, убедитесь, что у вас установлен пакет openssl и у него более новая версия (1.0.2g), как показано на следующем снимке экрана:

```
catalin@lubuntuBIG:~$ openssl version
OpenSSL 1.0.2g  1 Mar 2016
catalin@lubuntuBIG:~$
catalin@lubuntuBIG:~$
catalin@lubuntuBIG:~$
```

Если у вас не установлен openssl, вам необходимо сначала установить его, используя следующую команду:

```
sudo apt install openssl on Ubuntu
```

Или используйте следующую команду:

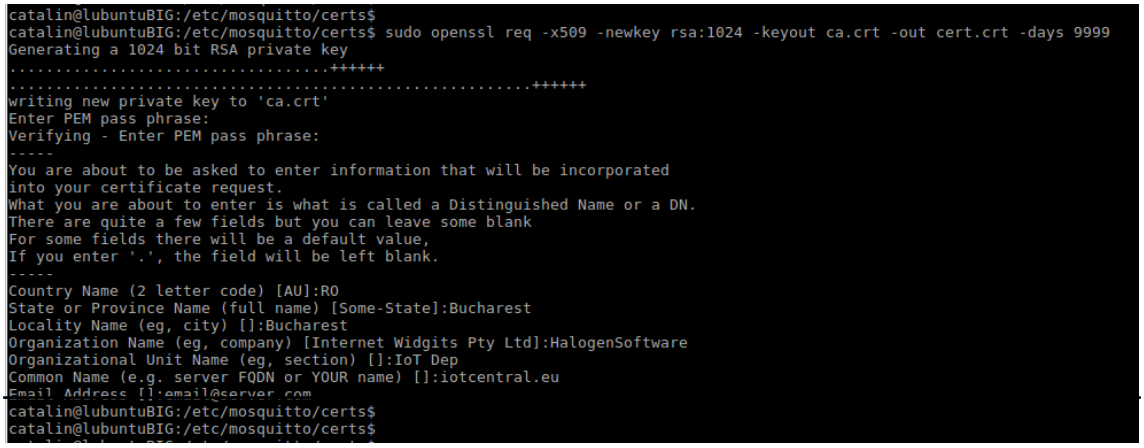
```
yum install openssl on CentOS/Redhat
```

Создание собственных сертификатов

Сначала перейдите в / etc / mosquitto / certs и введите следующую команду:

```
sudo openssl req -x509 -newkey rsa:1024 -keyout ca.crt -out cert.crt -days
9999
```

Затем вам будет предложено заполнить некоторые детали, как показано на следующем снимке экрана, о владельце сертификата, такие как страна проживания, штат, компания, город, адрес электронной почты, и наиболее важным из них является полное доменное имя. (Полное доменное имя). Это должен быть домен вашего сервера или IP-адрес вашего сервера, если у вас нет для него доменного имени:



Когда вас спросят о ключевой фразе, вам необходимо ввести пароль, длина которого превышает четыре символа. Позже,на следующем шаге мы удалим его из сертификата.

Результатом выполнения предыдущей команды должны быть два файла, расположенные в каталоге / etc / mosquitto / certs:

```
catalin@lubuntuBIG:/etc/mosquitto/certs$
catalin@lubuntuBIG:/etc/mosquitto/certs$ ll
total 16
drwxr-xr-x 2 root root 4096 okt  7 17:40 ./
drwxr-xr-x 5 root root 4096 okt  7 17:29 ../
-rw-r--r-- 1 root root 1041 okt  7 17:30 ca.crt
-rw-r--r-- 1 root root 1107 okt  7 17:30 cert.crt
catalin@lubuntuBIG:/etc/mosquitto/certs$
catalin@lubuntuBIG:/etc/mosquitto/certs$
catalin@lubuntuBIG:/etc/mosquitto/certs$
```

Следующая команда удаляет предыдущую парольную фразу и создает новый файл:

```
sudo openssl rsa –in ca.crt –out newca.pem
```

Входным файлом для этой команды является ca.crt, и он создаст имя файла newca.pem с удаленной кодовой фразой. Мы видим это на следующем скриншоте:

```
catalin@lubuntuBIG:/etc/mosquitto/certs$
catalin@lubuntuBIG:/etc/mosquitto/certs$
catalin@lubuntuBIG:/etc/mosquitto/certs$ sudo openssl rsa -in ca.crt -out newca.pem
Enter pass phrase for ca.crt:
writing RSA key
catalin@lubuntuBIG:/etc/mosquitto/certs$
catalin@lubuntuBIG:/etc/mosquitto/certs$
catalin@lubuntuBIG:/etc/mosquitto/certs$
```

В итоге у вас должно быть три файла в / etc / mosquitto / certs:

```
catalin@lubuntuBIG:/etc/mosquitto/certs$ ll
total 20
drwxr-xr-x 2 root root 4096 okt  7 17:43 ./
drwxr-xr-x 5 root root 4096 okt  7 17:29 ../
-rw-r--r-- 1 root root 1041 okt  7 17:30 ca.crt
-rw-r--r-- 1 root root 1107 okt  7 17:30 cert.crt
-rw-r--r-- 1 root root  887 okt  7 17:43 newca.pem
catalin@lubuntuBIG:/etc/mosquitto/certs$
catalin@lubuntuBIG:/etc/mosquitto/certs$
```

Примечание. В конфигурации mosquitto вам понадобятся только два из трех файлов cert.crt и newca.pem. Убедитесь, что ca.crt надежно хранится вне вашего сервера. Убедитесь, что к сертификатам имеет доступ только пользователь mosquitto. Брокер mosquitto в производственной среде также должен запускаться пользователем mosquitto, а не пользователем root.

Теперь, когда у вас есть нужные файлы с сертификатами, давайте настроим mosquitto для их учета. Если вы хотите, чтобы два порта были незащищенными, например 1883 и для соединения 9004 WebSockets, вы можете добавить, например, 8883 в качестве безопасного порта и 9883 в качестве порта WebSocket Secure (WSS).

Чтобы настроить mosquitto, вам необходимо отредактировать файл конфигурации mosquitto.conf, расположенный в / etc / mosquitto /. Новое содержимое файла должно быть:

```
catalin@lubuntuBIG:/etc/mosquitto$
catalin@lubuntuBIG:/etc/mosquitto$
catalin@lubuntuBIG:/etc/mosquitto$ more mosquitto.conf

listener 1883
protocol mqtt
listener 9004
protocol websockets
allow_anonymous true

# MQTT over TLS/SSL
listener 8883
cafile /etc/mosquitto/certs/cert.crt
certfile /etc/mosquitto/certs/cert.crt
keyfile /etc/mosquitto/certs/newca.pem
tls_version tlsv1

# WebSockets over TLS/SSL
listener 9883
protocol websockets
cafile /etc/mosquitto/certs/cert.crt
certfile /etc/mosquitto/certs/cert.crt
keyfile /etc/mosquitto/certs/newca.pem

pid_file /var/run/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

catalin@lubuntuBIG:/etc/mosquitto$
catalin@lubuntuBIG:/etc/mosquitto$
```

Перезапустите службу mosquitto, чтобы начать работу с новой конфигурацией, используя следующую команду:

```
sudo service mosquitto restart
```

Теперь вы можете проверить, правильно ли запущен mosquitto и нет ли ошибок в mosquitto.conf, просмотрев /var/log/mosquitto/mosquitto.log:

```
1507382754: mosquitto version 1.4.8 terminating
1507382756: mosquitto version 1.4.8 (build date Tue, 23 May 2017 22:14:40 +0100) starting
1507382756: Config loaded from /etc/mosquitto/mosquitto.conf.
1507382756: Opening ipv4 listen socket on port 1883.
1507382756: Opening ipv6 listen socket on port 1883.
1507382756: Opening websockets listen socket on port 9004.
1507382756: Opening ipv4 listen socket on port 8883.
1507382756: Opening ipv6 listen socket on port 8883.
1507382756: Opening websockets listen socket on port 9883.
```

Если ошибок нет, можно переходить к подключению клиентов к новым безопасным портам.

Обеспечение безопасности соединения между ESP8266 и брокером MQTT

Если вы не хотите иметь собственного брокера, но хотите безопасное соединение MQTT, вы можете использовать экземпляр облачного MQTT, например http://iotcentral.eu.

Сначала создайте учетную запись на iotcentral.eu и подтвердите свой адрес электронной почты. После этого вы можете авторизоваться на iotcental.eu и получить свою приватную назначенную тему. Это восьмизначный код, например c5c05211, и этот код должен предшествовать всем вашим темам следующим образом:

```
c5c05211/living/temperature
```

Независимо от того, публикуетесь ли вы или подписываетесь. Следующий код подключается к Wi-Fi, а затем устанавливает безопасное соединение с облачным MQTT iotcentral.eu через порт 8883. Каждое сообщение, отправленное брокеру iotcentral.eu, возвращается в виде обратной связи.Включенные файлы заголовков - это ESP8266WiFi и класс PubSubClient MQTT:

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
```

GPIO 12 будет использоваться позже для мигания светодиода каждый раз при получении сообщения, а GPIO 13 будет изменяться на HIGH каждый раз при получении сообщения с содержанием 1 и на LOW, если полезная нагрузка сообщения равна 0:

```
#define PIN_12 12
#define PIN_13 13
```

Определите значения для сети Wi-Fi, SSID и пароля, а также используемый сервер MQTT (в этом примере использовался iotcentral.eu, но вы можете использовать локальный), имя пользователя, пароль и базовую тему (нам нужно взять это с iotcentral.eu в своем аккаунте):

```
const char* ssid = "YOUR_WIFI_SSID";
const char* password = "YOUR_WIFI_PASSWORD";
const char* mqtt_server = "iotcentral.eu";
const char* mqtt_username = "email@email.com"; // адрес электронной почты,
                                                  используемый на iotcentral.eu

const char* mqtt_password = "*******"; // ваш пароль, используемый на
                                          iotcentral.eu
#define MQTT_CLIENT_ID          "ESP_%06X"
#define BASE_TOPIC "c5c05211" // получить с iotcentral.eu
char dev_name[11];
```

Если в других примерах книги класс WiFiClient использовался для незащищенного соединения, здесь класс WiFiClientSecure будет использоваться для создания объекта espClient:

```
WiFiClientSecure espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;
```

В функции настройки мы подключим ESP8266 к предоставленной сети Wi-Fi и запишем в объект клиента MQTT-сервер, который будет использоваться, и порт. В предыдущих примерах использовался порт 1883. Брокер прослушивает порт 8883 для обеспечения безопасного подключения к порту 8883:

```
void setup()
{
  pinMode(PIN_12, OUTPUT);
  digitalWrite(PIN_12, LOW);
  pinMode(PIN_13, OUTPUT);
  digitalWrite(PIN_13, LOW);
  sprintf(dev_name, MQTT_CLIENT_ID, ESP.getChipId());
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 8883);
  client.setCallback(callback);
}
```

Используйте следующий код, чтобы запустить соединение Wi-Fi:

```
void setup_wifi() {
  delay(10);
  // Сначала подключаемся к сети
  Wi-FiSerial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected. My IP address: ");
  Serial.println(WiFi.localIP());
}
```

Это функция обратного вызова, которая вызывается каждый раз при получении сообщения от брокера. Если к GPIO 12 подключен светодиод, он будет гореть 50 мс каждый раз при получении сообщения:

```
void callback(char* topic, byte* payload, unsigned int length) {
  digitalWrite(PIN_12,HIGH);
  delay(50);
  digitalWrite(PIN_12,LOW);
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println("");
  // Включаем светодиод, подключенный к PIN_13, если получено сообщение '1' по
любой теме
  if ((char)payload[0] == '1') {
    digitalWrite(PIN_13, HIGH);   // Turn the LED on
  }
  else if ( (char)payload[0] == '0')
  {
    digitalWrite(PIN_13, LOW);
  }
}
```

Если соединение с брокером потеряно, попробуйте восстановить соединение. Если вы хотите сделать это более надежным, вы также можете проверить соединение Wi-Fi и попытаться повторно подключиться к роутеру Wi-Fi, а затем к брокеру MQTT:

```
void reconnect() {
  // Цикл, пока мы снова не подключимся
  while (!client.connected()) {
    Serial.print("Start MQTT connection...");
    if (client.connect(dev_name, mqtt_username, mqtt_password)) {
      Serial.print("connected to MQTT broker");
      Serial.println(mqtt_server);
      client.subscribe(BASE_TOPIC"/#");
    } else {
      Serial.print("Failed to connect. Error code: ");
      Serial.println(client.state());
      Serial.println(" try again in 5 seconds");
      // Отключите и подождите 5 секунд перед повторной попыткой .
      client.disconnect();
      delay(5000);
    }
  }
}
```

Код основного цикла проверяет, подключен ли ESP8266 к брокеру iotcentral.eu MQTT. В случае сбоя он попытается подключиться заново. Каждую секунду увеличивайте значение и публикуйте это значение в сообщении брокеру:

```
void loop() {
  if (!client.connected()) {
    Serial.println("Reconnect to the broker....");
    reconnect();
  }
  client.loop();

  long now = millis();
  if (now - lastMsg > 1000) {
    lastMsg = now;
    ++value;
    snprintf (msg, 75, "Sending message #%ld", value);
    Serial.print("Send to MQTT broker message: ");
    Serial.println(msg);
    client.publish(BASE_TOPIC"/outTopic", msg);
  }
}
```

Потому что в функции повторного подключения мы подписались на:

```
client.subscribe(BASE_TOPIC"/#");
```

и мы публикуем:

```
client.publish(BASE_TOPIC"/outTopic", msg);
```

Каждый раз при отправке сообщения брокер отправляет его обратно. Откройте последовательный терминал, и вы увидите следующие сообщения:

```
Message arrived [c5c05211/outTopic] Sending message #2048
Send to MQTT broker message: Sending message #2049
Message arrived [c5c05211/outTopic] Sending message #2049
Send to MQTT broker message: Sending message #2050
Message arrived [c5c05211/outTopic] Sending message #2050
Send to MQTT broker message: Sending message #2051
Message arrived [c5c05211/outTopic] Sending message #2051
Send to MQTT broker message: Sending message #2052
Message arrived [c5c05211/outTopic] Sending message #2052
Send to MQTT broker message: Sending message #2053
Message arrived [c5c05211/outTopic] Sending message #2053
Send to MQTT broker message: Sending message #2054
Message arrived [c5c05211/outTopic] Sending message #2054
```

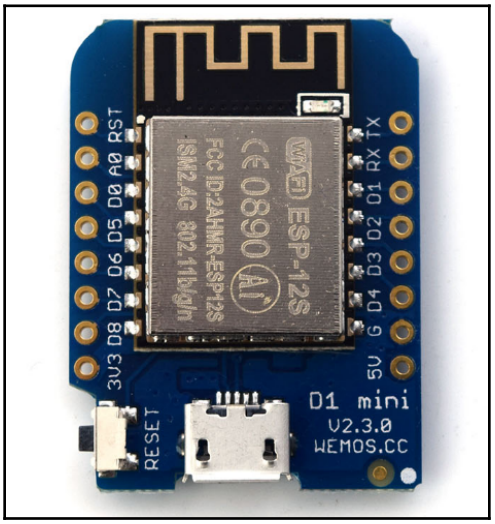
Работа в автономном режиме

Если ваши данные более конфиденциальны, и вы не хотите передавать их по сети Wi-Fi или у вас нет подключения к Wi-Fi, решением является хранение ваших данных на SD-карте.

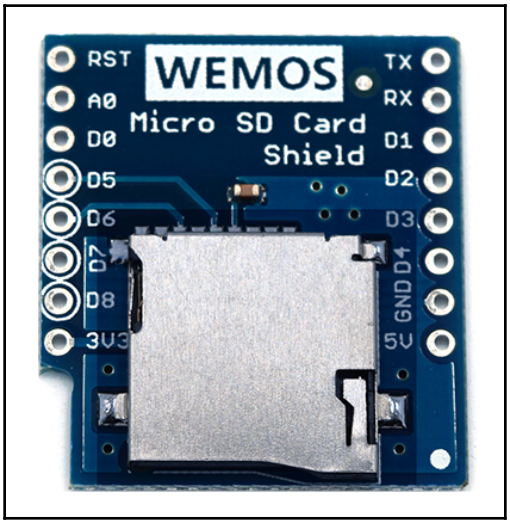
Давайте посмотрим, как данные могут быть сохранены на SD-карте.

Необходимые компоненты, которые будут использоваться:

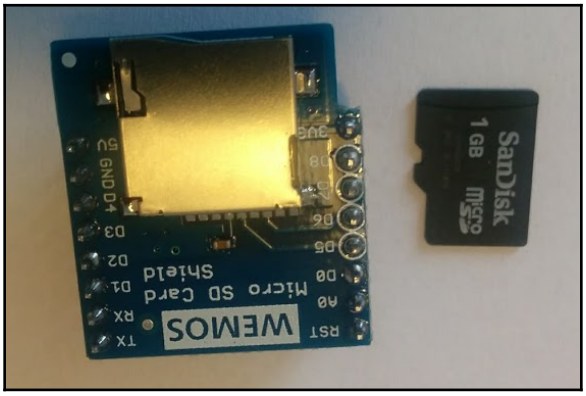
- Wemos D1 mini:



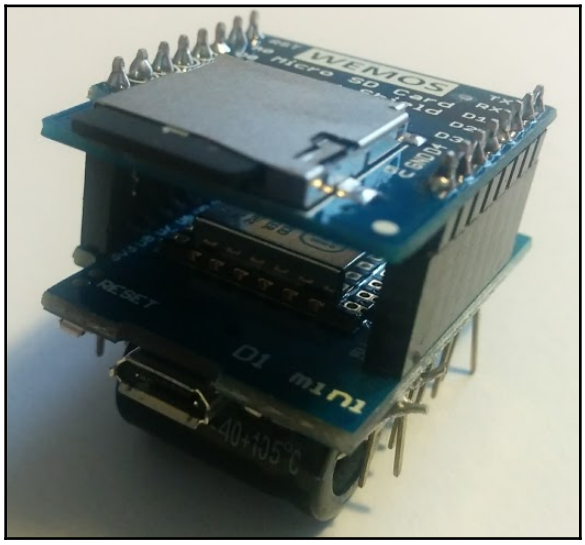
- Шилд карты microSD:



- карта microSD:



Поскольку карта microSD является шилдом для Wemos D1 mini, их легко сложить; вам просто нужно припаять штыри, которые входят в комплект:



Давайте определим размер SD-карты по следующему эскизу.

Включите SPI.h и библиотеку SD:

```
#include <SPI.h>
#include <SD.h>
```

Настройте переменные с помощью функций библиотеки служебных программ SD:

```
Sd2Card card;
SdVolume volume;
SdFile root;
const int chipSelect = D8;
```

В функции настройки мы определим, вставлена карта или нет, и данные карты будут считаны через SPI:

```
void setup()
{
  Serial.begin(115200);
  Serial.print("\nInitializing SD card...");
```

Используйте код инициализации из служебных библиотек:

```
    if (!card.init(SPI_HALF_SPEED, chipSelect)) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("* is a card inserted?");
    Serial.println("* is your wiring correct?");
    Serial.println("* did you change the chipSelect pin to match your
shield or module?");
    return;
  } else {
    Serial.println("Wiring is correct and a card is present.");
  }
```

подключение правильное, карта присутствует

```
  // выводим тип карты
  Serial.print("\nCard type: ");
  switch (card.type()) {
    case SD_CARD_TYPE_SD1:
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");
      break;
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");
      break;
    default:
      Serial.println("Unknown");
```

```
  }
      // Теперь попробуем открыть раздел 'volume' / 'partition' - это должна быть
FAT16
  if (!volume.init(card)) {
    Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've
formatted the card");
    return;
  }
  // выводим тип и размер первого тома типа
  FATuint32_t volumesize;
  Serial.print("\nVolume type is FAT");
  Serial.println(volume.fatType(), DEC);
  Serial.println();
  volumesize = volume.blocksPerCluster();    // кластеры - это наборы блоков

  volumesize *= volume.clusterCount();       // у нас будет много кластеров

  volumesize *= 512;                            // Блоки SD-карт
always 512 bytes
  Serial.print("Volume size (bytes): ");
  Serial.println(volumesize);
  Serial.print("Volume size (Kbytes): ");
  volumesize /= 1024;
  Serial.println(volumesize);
  Serial.print("Volume size (Mbytes): ");
  volumesize /= 1024;
  Serial.println(volumesize);
  Serial.println("\nFiles found on the card (name, date and size in bytes):
");                                              имя, дата и размер в байтах
  root.openRoot(volume);
  // выводим список всех файлов на карте с указанием даты и размера
  root.ls(LS_R | LS_DATE | LS_SIZE);
}
```

В функции цикла делать нечего, поскольку определение типа карты и ее свойств производилось в функции настройки:

```
void loop()
{
      // после установки ничего не происходит
}
```

Вывод последовательной консоли покажет тип карты, размер карты в байтах, килобайтах и мегабайтах, и, если на карте есть файлы, их имя и размер:

```
Card type: SD2

Volume type is FAT16

Volume size (bytes): 987463680
Volume size (Kbytes): 964320
Volume size (Mbytes): 941

Files found on the card (name, date and size in bytes):
DATALOG.TXT   2000-01-01 01:00:00 4978
```

Вы также можете использовать карту SDHC большего размера или карту FAT32:

```
Initializing SD card...Wiring is correct and a card is present.

Card type: SDHC

Volume type is FAT32

Volume size (bytes): 2670723072
Volume size (Kbytes): 2608128
Volume size (Mbytes): 2547

Files found on the card (name, date and size in bytes):
```

Сохранение данных на SD-карту

Предположим, что теперь вам нужно сохранить данные на SD-карту, которые впоследствии можно будет использовать в автономном режиме на ПК.

Давайте подключим *DHT22,* как мы это делали в главе *3* «Создание домашнего термостата с *ESP8266»,* прочитаем его значение и зарегистрируем его в файле на карте *microSD.*

Используя те же библиотеки для SPI и SD-карты:

```
#include <SPI.h>
#include <SD.h>
#include <DHT.h>
const int chipSelect = D8;
```

Определите тип DHT, поскольку библиотека может работать с DHT11 и DHT22:

```
#define DHTTYPE DHT22
#define DHTPIN  4
#define DEV_TYPE  "dht"
DHT dht(DHTPIN, DHTTYPE, 11);
float humidity, temp_f;  //  Значения считываются с датчика
```

Определите функцию, которая будет считывать температуру и обновлять глобальные переменные влажность и temp_f с влажностью и температурой:

```
void gettemperature()
{
  int runs=0;
  do {
       delay(2000);
       temp_f = dht.readTemperature(false);
       humidity = dht.readHumidity();

       if(runs > 0)
           Serial.println("##Failed to read from DHT sensor! ###");
       runs++;
    }
    while(isnan(temp_f) && isnan(humidity));
}
```

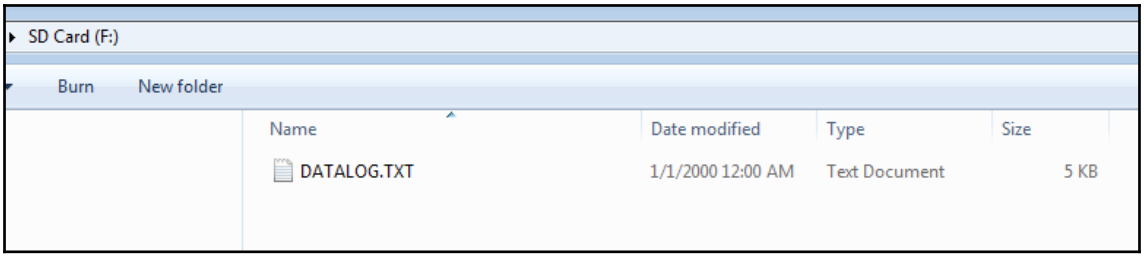
Инициализируйте SD-карту и выполните первое считывание влажности и температуры:

```
void setup()
{
// Открываем последовательную связь и ждем открытия порта:
  Serial.begin(115200);
  Serial.print("Initializing SD card...");
  // посмотрим, присутствует ли карта и может ли она быть инициализирована:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // больше ничего не делаем:
    return;
  }
  Serial.println("card initialized.");
  gettemperature();
}
```

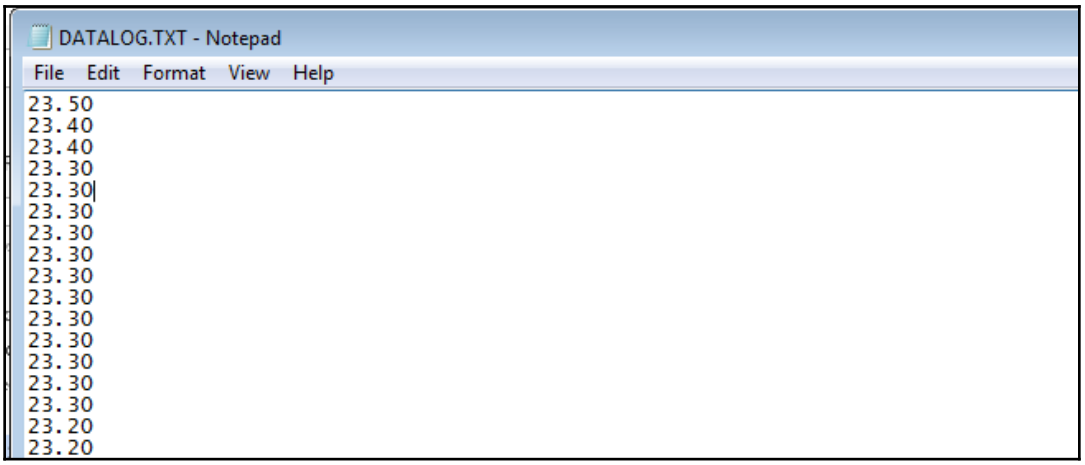
Каждые три секунды считывайте температуру и влажность, открывая существующий файл и добавляя температуру в файл DATALOG.TXT. В конце закройте файл:

```
void loop()
{
// создаем строку для сборки данных в журнал:
  String dataString = "";
  gettemperature();
  dataString += String(temp_f);
// открываем фай л. обратите внимание, что одновременно может быть открыт только один файл,
  // поэтому вам нужно закрыть этот, прежде чем открывать другой .
 // для записи в фай л вам нужен FILE_WRITE в качестве второго параметра.
  // для чтения из фай ла следует использовать SD.open (имя_фай ла).
  File dataFile = SD.open("datalog.txt", FILE_WRITE);
  // если фай л доступен, пишем в него:
  if (dataFile)
  {
    dataFile.println(dataString);
    dataFile.close();
   // печать в последовательный порт:
    Serial.println(dataString);
  }
  //  если файл не открыт, выдается сообщение об ошибке:
  else {
    Serial.println("error opening datalog.txt");
  }
  delay(3000);
}
```

Убедитесь, что на карте microSD на ПК отображается созданный файл:



Откройте файл, чтобы просмотреть зарегистрированные данные:



Если ваши данные являются конфиденциальными, вы можете зашифровать их, а затем записать их на SD-карту. В случае утери SD-карты никто не сможет увидеть ваши данные.

Резюме

Теперь вы можете передавать зашифрованные сообщения MQTT брокеру MQTT, а оттуда - другому ESP8266, базе данных или серверу. При этом никто не сможет перехватить и изменить ваши данные, поэтому ваш дом или ваши данные будут в безопасности. Безопасность в IoT в наши дни очень важна,поскольку существует много данных устройств, поэтому ваш дом или вы не защищены даже с помощью пользователя и пароля. Если вам нужно работать в автономном режиме, теперь у вас есть целая карта microSD для записи или чтения данных с нее.
Используя ESP8266 с батарейным питанием и функции глубокого сна, теперь вы можете записывать данные на карту microSD в течение нескольких месяцев. Зашифруйте их, чтобы быть уверенным, что вы единственный, у кого есть к ним доступ.
В следующей главе будет показано, как передавать данные через соединение WebSocket в потоке, данные, которые могут храниться в базе данных временных рядов или отображаться в виде графики в реальном времени.

7

Связь в реальном времени

Все сообщения, которые были представлены до сих пор, основывались на методе запроса-ответа, в котором один объект отправлял запрос, а другой объект отправлял ответ. Но бывают ситуации, когда вам нужна связь в реальном времени между модулем ESP8266 и сервером, а не только транзакции. Чтобы обеспечить связь в реальном времени, мы будем использовать WebSockets для потоковой передачи значений ускорения с датчика ускорения на сервер, чтобы отображать их в графике в реальном времени, а также сохранять их в базе данных временных рядов.

WebSockets

WebSockets - это протокол связи, обеспечивающий полнодуплексные сообщения с потоковой передачей поверх протокола управления транспортом (TCP). Он стандартизирован организациями W3 и присутствует во всех основных браузерах (Internet Explorer должен быть версии 11+). WebSockets обеспечивает связь между браузером и сервером, а также между нашим модулем и сервером. Через сервер данные с ESP8266 могут поступать в браузер. Еще одно большое преимущество WebSockets - двунаправленная связь без создания нового запроса. Все коммуникации выполняются по открытому TCP-соединению.

Детали протокола

Соединение WebSocket начинается как соединение HTTP с запросом на обновление до протокола websocket.

В этом случае клиент отправляет:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: t3JJHjbGL5EzHkh8GBMXGw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

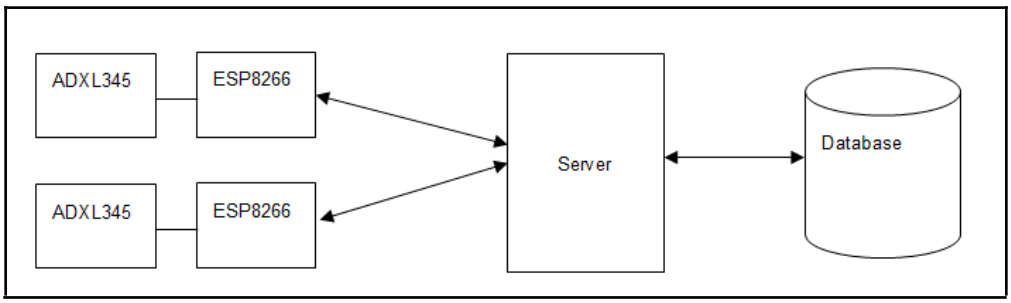
И если сервер поддерживает протокол websocket, он ответит:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: ADnfv8rNkTYjSFnn5OPpH2HaGWj=
```

Обратите внимание, что значение Sec-WebSocket-Key - это случайное значение, закодированное в base64, чтобы избежать кэширования. Сервер добавляет фиксированное значение 258EAFA5-E914-47DA-95CA- C5AB0DC85B11 к полученному значению Sec-WebSocket-Key и выполняет SHA1. Результат добавляется в Sec-WebSocket-Accept и отправляется обратно клиенту. С этого момента связь установлена, и сервер и клиент могут обмениваться сообщениями.

Потоковая передача данных с ESP8266

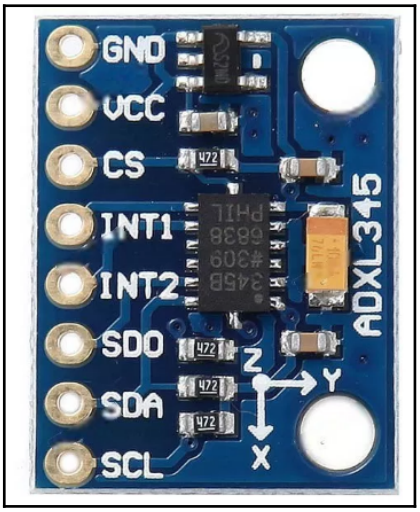
Для потоковой передачи данных из ESP8266 нам необходимо сначала установить соединение WebSocket между ESP8266 и сервером; данные, которые будут передаваться через соединение WebSocket, будут значениями ускорения для осей X, Y и Z. ESP8266 считывает их с микросхемы ADXL345 и отправляет на сервер nodeJS. С сервера данные могут быть отправлены в подключенный браузер на том же сервере или могут быть записаны в базу данных для дальнейшего анализа:



Последняя схема может также включать базу данных временных рядов, такую как InfluxDb, для хранения значений, передаваемых ADXL345.
Добавление базы данных может позволить вам собирать различные данные с нескольких датчиков ускорения, сохранять их в базе данных, извлекать их по запросу для сравнения с текущими значениями или рисовать красивые графики с текущими и историческими данными.
Сервер также может реагировать на некоторые значения и отправлять предупреждения (электронная почта, SMS) и отправлять данные другим модулям ESP8266 для реагирования или другим серверам.

Акселерометр ADXL345

Этот трехосевой акселерометр со сверхмалым энергопотреблением, производимый компанией Analog Devices (AD), может выполнять измерения с высоким разрешением до ± 16g. Если вы планируете измерять только гравитацию Земли, вам может хватить ± 2g. Для движения автомобиля будет достаточно ± 4g, но если вы хотите отслеживать внезапно останавливающийся объект, вам понадобится ± 16g:

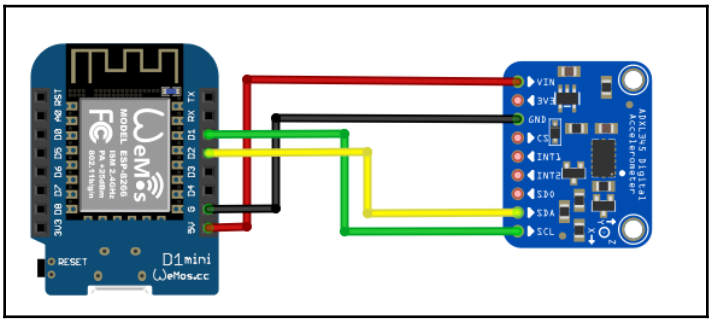


ADXL345 может быть подключен к ESP8266 по линиям SPI или I2C. В нашем случае мы будем использовать соединение I2C. Адрес шины, на который будет реагировать акселерометр, - 0x53.

Пин	Описание
GND	Подключение к земле
VCC	3V3. Некоторые модули используют 5 В, поскольку у них падение напряжения до 3 В
CS	Chip select. - Выбор чипа. High для I2C и LOW для подключения SPI к ESP8266
INT1	Программируемое прерывание
INT2	Программируемое прерывание
SDO	Последовательные данные для SPI. Для I2C выберите адрес шины
SDA	Последовательные данные для I2C
SCL	Последовательный тактовый сигнал

Подключение к ESP8266

ADXL345 будет подключен к ESP8266 по шине I2C, что означает, что будут использоваться только четыре провода: VCC, GND, SDA и SCL:



Код ESP8266

Поскольку в скетче также есть несколько дополнительных классов для подключения к WebSocket, здесь будет показана только основная часть файла .ino. Для всего проекта перейдите по следующему адресу GitHub:

`https://github.com/bcatalin/esp8266-book/tree/master/Chapter7.`

Необходимые включенные файлы:

```
#include "Wire.h"
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>
#include <FS.h>
#include <ESP8266WiFi.h>
#include "SocketIOClient.h"
#include <DNSServer.h>
#include <ESP8266WebServer.h>
#include <WiFiManager.h>
#include <ArduinoJson.h>
#include <Wire.h>
#include <ESP8266HTTPClient.h>
#include <ESP8266httpUpdate.h>
#include <SPI.h>
```

Создайте экземпляр объекта Accel и создайте уникальную идентификацию в качестве параметра для конструктора класса:

```
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(121212);
```

Выделите место для имени сервера или его IP-адреса и установите порт по умолчанию на 1234. Позже пользователи смогут выбрать один из них во время процедуры настройки Wi-Fi:

```
char acc_server[40];
char acc_port[6] = "1234";
```

Объявите глобальные переменные, которые будут использоваться:

```
#define ACC_CLIENT_ID         "RAM_%06X"
#define INFO Serial.printf
char dev_name[50];
```

Установите для clean_g значение 1, если вы хотите отформатировать SPIFFS и еще раз прошить скетч на ESP8266:

```
int clean_g = 0;  // флаг для сохранения данных

bool shouldSaveConfig = false;
```

Клиентский объект будет использоваться для отправки и получения данных по протоколу веб-сокета. Реализация класса является внешней по отношению к этому файлу и находится в файлах SocketIOClient.cpp и SocketIOClient.h:

```
SocketIOClient client;
StaticJsonBuffer<300> jsonBuffer;
extern String RID;
extern String Rname;  / / Содержимое внешней строки

unsigned long previousMillis =
0;long interval = 100;
unsigned long lastreply = 0;
unsigned long lastsend = 0;
```

Функция обратного вызова для уведомления нас о необходимости сохранить конфигурацию в файле config.json на SPIFFS:

```
void saveConfigCallback () {
  Serial.println("Should save config");
  shouldSaveConfig = true;
}
```

В функции setup () он инициализирует и настраивает микросхему ADXL345:

```
void setup()
{
  // поместите сюда свой установочный код, чтобы запустить его один раз:
  Serial.begin(115200); delay(10);
  Serial.println();
  pinMode(A0, INPUT);
  pinMode(SIGNAL_PIN, OUTPUT); digitalWrite(SIGNAL_PIN, LOW);
```

Инициализируйте датчик. Если датчик не подключен к плате ESP8266, дождитесь его:

```
 if(!accel.begin())
{
/* There was a problem detecting the ADXL345 ... check your connections */
Serial.println("Ooops, no ADXL345 detected ... Check your wiring!");
while(1);
}
```

/ * При обнаружении ADXL345 возникла проблема ... проверьте ваши соединения * /

Установите диапазон, подходящий для вашего проекта. ADXL345 поддерживает до ± 16 г.

В зависимости от вашего приложения вы можете выбрать другое значение, изменив параметр функции SetRange:

```
accel.setRange(ADXL345_RANGE_16_G);
// accel.setRange(ADXL345_RANGE_8_G);
```

```
 // accel.setRange(ADXL345_RANGE_4_G);
 // accel.setRange(ADXL345_RANGE_2_G);
Just for
  if(clean_g)
   SPIFFS.format();
```

Затем прочтите конфигурацию из файла SPIFFS config.json. Если файл конфигурации не найден, ESP8266 предполагает, что он не настроен, поэтому он запустится в режиме точки доступа и будет ждать, пока пользователь установит SSID Wi-Fi, пароль Wi-Fi, имя сервера и порт сервера, который будет использоваться для подключения к:

```
if (SPIFFS.begin())
{
  Serial.println(F("mounted file system"));
  if (SPIFFS.exists("/config.json"))
  {
    //файл существует, читается и загружается
    Serial.println("reading config file");
    File configFile = SPIFFS.open("/config.json", "r");if
    (configFile) {
      Serial.println("opened config file");
      size_t size = configFile.size();
      // Выделяем буфер для хранения содержимого файла.
      std::unique_ptr<char[]> buf(new char[size]);
      configFile.readBytes(buf.get(), size);
      DynamicJsonBuffer jsonBuffer;
      JsonObject& json = jsonBuffer.parseObject(buf.get());
      json.printTo(Serial);
      if (json.success()) {
        Serial.println(F("\nparsed json"));

        strcpy(acc_server, json["acc_server"]);
        strcpy(acc_port,   json["acc_port"]);
      } else {
        Serial.println(F("failed to load json config"));
      }
    }
  }
} else {
  Serial.println(F("failed to mount FS"));
}
```

Настройте WiFiManager с настраиваемыми полями, такими как имя сервера и порт сервера, а также SSID Wi-Fi и пароль Wi-Fi, которые будут сохраняться в SPIFFS для автоматического подключения при каждом перезапуске ESP8266:

```
  WiFiManagerParameter custom_acc_server("server", "RAM IP", acc_server,
40);
  WiFiManagerParameter custom_acc_port("port", "RAM port", acc_port, 5);
  WiFiManager wifiManager;
  wifiManager.setSaveConfigCallback(saveConfigCallback);
  wifiManager.addParameter(&custom_acc_server);
  wifiManager.addParameter(&custom_acc_port);

  if(clean_g)
   wifiManager.resetSettings();
  sprintf(dev_name, ACC_CLIENT_ID, ESP.getChipId());
  INFO("..DEV:%s \n",dev_name);
  if ( !wifiManager.autoConnect(dev_name) )
    {
      Serial.println(F("failed to connect and hit timeout"));
      delay(3000);
      // сбрасываем и пробуем снова, а может, переводим в глубокий сон
      ESP.reset();
      delay(5000);
    }
  // если вы попали сюда, значит, вы подключились к Wi-Fi
  Serial.println("connected...yeey :)");
  // читаем обновленные параметры
  strcpy(acc_server, custom_acc_server.getValue());
  strcpy(acc_port, custom_acc_port.getValue());
```

Теперь у нас есть вся информация, которая будет сохранена в SPIFFS. Эта часть будет вызываться только при первой настройке ESP8266. Введенная информация будет постоянной, поскольку теперь она сохраняется и извлекается при каждой загрузке:

```
  if (shouldSaveConfig)
  {
    Serial.println("saving config");
    DynamicJsonBuffer jsonBuffer;
    JsonObject& json = jsonBuffer.createObject();
    json["acc_server"] = acc_server;
    json["acc_port"]   = acc_port;
    File configFile = SPIFFS.open("/config.json", "w");
    if (!configFile) {
      Serial.println("failed to open config file for writing");
    }
```

```
      json.printTo(Serial);
      json.printTo(configFile);
      configFile.close();
    }
```

Теперь мы подключимся к серверу, и после этого мы отправим сообщение о подключении, которое содержит нашу уникальную идентификацию, полученную из MAC-адреса ESP8266.

Сервер может использовать это сообщение для идентификации ESP8266 и динамического создания веб-интерфейса. В этом случае, если сервер получит сообщение о соединении, он построит график, который покажет полученные значения для всех трех осей в реальном времени:

```
 if (!client.connect(acc_server, atoi(acc_port) ))
  {
    Serial.println(F("connection failed"));
    return;
  }
  if (client.connected())
  {
    client.sendJSON("connection", "{\"acc_id\":\"" + String(dev_name) + "\"
}" );
  }
}
```

В основном цикле мы:

- Прочтите значения ускорения каждый интервал и создайте сообщение JSON, которое будет отправлено на сервер.
- Проверьте наличие входящих сообщений с сервера. Помните, что веб-сокет - это дуплексный протокол. Вы также можете управлять ESP8266 с сервера; вы можете установить некоторые параметры, перезагрузить ESP8266, запустить определенный GPIO или даже сбросить ESP8266 до значений по умолчанию, отформатировав SPIFFS.
- Проверьте состояние подключения и, если необходимо, переподключитесь к серверу.

Также в функции loop () будет проверяться состояние соединения и при необходимости переподключиться к серверу, если соединение былопотеряно:

```
void loop()
{
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis > interval)
  {
    previousMillis = currentMillis;

    sensors_event_t event;
    accel.getEvent(&event);
```

```
  String acc_data;
  StaticJsonBuffer<100> jsonDeviceStatus;
  JsonObject& jsondeviceStatus = jsonDeviceStatus.createObject();
  jsondeviceStatus["device_name"] = dev_name;
  jsondeviceStatus["x"]         = event.acceleration.x;//x;
  jsondeviceStatus["y"]         = event.acceleration.y;//y;
  jsondeviceStatus["z"]         = event.acceleration.z;//z;

  jsondeviceStatus.printTo(acc_data);
  client.sendJSON("JSON", acc_data);
}
```

Проверьте наличие входящих сообщений с сервера:

```
if(client.monitor())
{
  lastreply = millis();
```

Если ESP8266 получит сообщение с именем welcome, он ответит сообщением с именем connection и своим уникальным идентификатором:

```
if(strcmp(String(RID).c_str(), "welcome") == 0)
{
  client.sendJSON("connection", "{\"acc_id\":\"" + String(dev_name) + "\"
}" );
}
if(RID != "")
{
  Serial.print(F("Message: ")); Serial.println(RID);
```

Если полученное сообщение - это resetModule, то EPS8266 перезагрузится. Сюда можно добавить множество сообщений, чтобы изменить статус GPIO, прочитать статус GPIO, прочитать значение из A0 или записать PWM в GPIO:

```
if(strcmp(String(RID).c_str(), "resetModule") == 0)
{       //reset the module        delay(1000);       ESP.reset();
  } }
}
```

Проверьте соединение с сервером и, если необходимо, переподключитесь к нему:

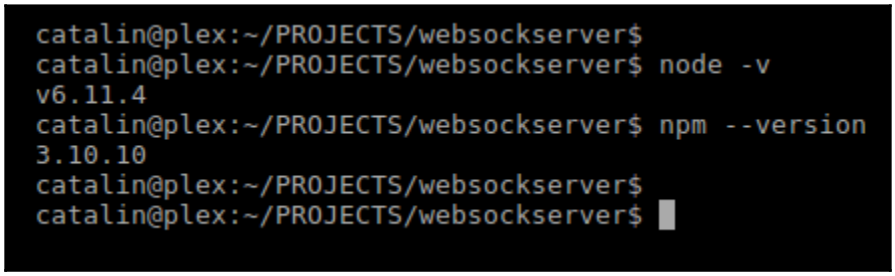
```
if (!client.connected())
{
   Serial.println("LOOP: Client not connected, try to reconnect");
   client.connect(acc_server, atoi(acc_port) );
   while(!client.connected())
   {
     client.connect(acc_server,atoi(acc_port));
```

```
        delay(1000);
      }
      client.sendJSON("connection", "{\"acc_id\":\"" + String(dev_name) +
"\" }" );
    }
}
```

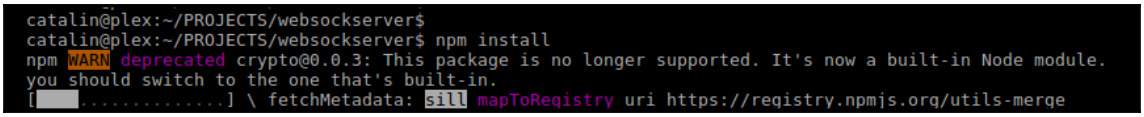
Сервер для подключения к WebSocket? для серверной части я выбрал Node.JS и ExpressJS. Код для сервера можно найти по этой ссылке: https://github.com/ bcatalin / esp8266 book / tree / master / Chapter7.

Шаги для запуска сервера:

1. Установите Node.js и npm. Убедитесь, что оба присутствуют в вашей системе:



2. Установите зависимости сервера с помощью команды npm -v:



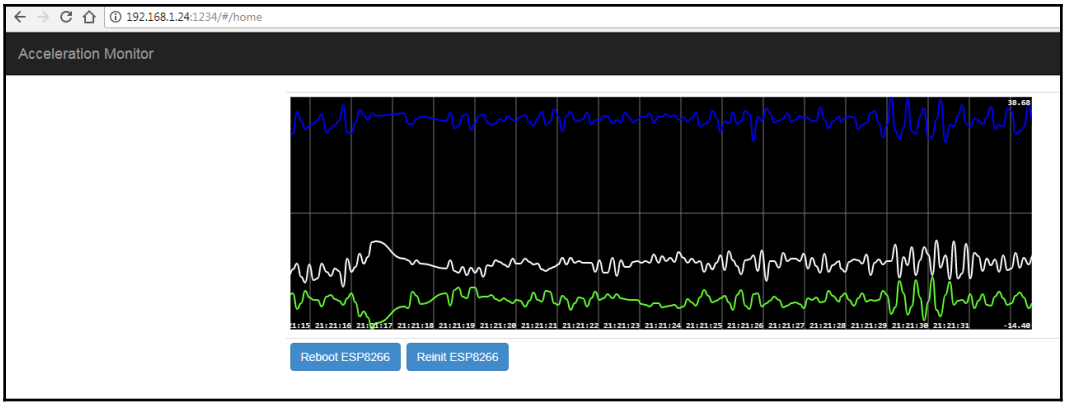
Отредактируйте файл websocketserver / public / services / wsDataService.js, указав свой IP-адрес сервера и используемый порт:

4. Запустите сервер с помощью команды node server.js, выданной вдиректории websocketserver:

```
catalin@plex:~/PROJECTS/websockserver$ node server.js
The magic happens on port 1234
Connected  socket id:HgmjiF_fAE3XE_3AAAAA
SOCK: connection - new WEB connection!!!!!
SOCK: RX connection => Send acc_ram with data
=======================================================
|ID| Web Socket ID                                      |
=======================================================
| 0| HgmjiF_fAE3XE_3AAAAA
=======================================================
=======================================
|ID| ACC ID     | Socket ID            |
=======================================
=======================================================
Connected  socket id:7-xIcezugppEdDXgAAAB
SOCK: connection - new ACC connection
{ acc_id: 'ACC_D357A3' }
=======================================
|ID| ACC ID     | Socket ID            |
=======================================
| 0| ACC_D357A3 | 7-xIcezugppEdDXgAAAB |
=======================================================
{ acc_id: 'ACC_D357A3' }
=======================================
```

На предыдущем снимке экрана вы видите два подключения: одно из Интернета с ID веб-сокета HgmjiF_fAE3XE_3AAAAA, а другое из модуля ESP8266 с ID сокета: 7- xIcezugppEdDXgAAAB.

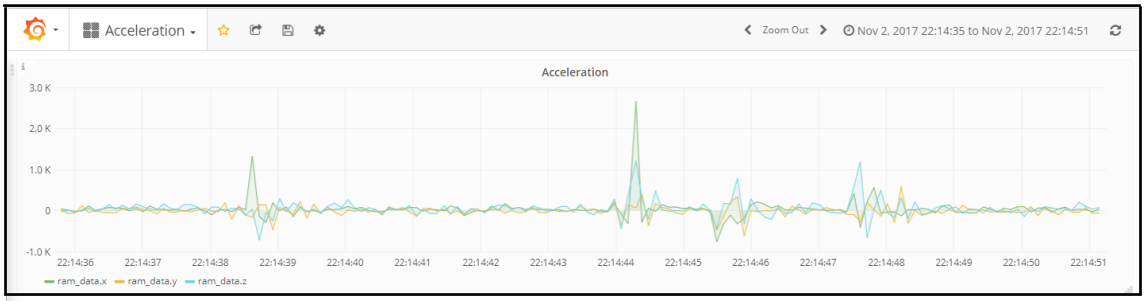
Откройте браузер и укажите в нем IP-адрес и порт вашего сервера, и вы увидите ускорение на красивом графике, предоставленном smoothieJS:



Нажатие Reboot ESP8266 удаленно перезагрузит ваш модуль, и если вы нажмете кнопку Reinit ESP8266 на веб-странице, ESP8266 сбросит все данные для учетных данных Wi-Fi, IP-адреса и номера порта.

Вы можете добавить несколько модулей на один и тот же сервер. Для каждого нового соединения сервер будет добавлять для него новую графику и новые кнопки. Таким образом, вы можете отслеживать несколько датчиков на одной странице, не обновляя ее и не предпринимая никаких действий. Это очень хорошо, если вы хотите создать красивую панель управления для своей платформы домашней автоматики.

В качестве улучшения вы можете добавить базу данных временных рядов, такую как InfluxDB (https: // www.Influxdata.com/), чтобы сохранить полученное значение с желаемой вами стойкостью. Чтобы создать более сложную панель управления, вы можете использовать Grafana (https://grafana.com/):



Код сервера состоит из двух частей:

1. Внутренний код, отвечающий за получение соединений от модулей и и веб-страниц ESP8266.
2. Код внешнего интерфейса, который подключается к серверу и показывает красивые графики.

Бэкэнд-код

Весь бэкэнд-код написан на JavaScript и находится в файле server.js.

Для создания сервера используется модуль, предоставляемый socket.io, потому что socket.io обеспечивает двунаправленную связь на основе событий вреальном времени:

```
var server = http.createServer(app);
var io = require('socket.io').listen(server);
```

Чтобы получать сообщения, необходимо добавить для них обработчик. В событии «connection - соединение», когда получено сообщение, это означает, что новый модуль ускорения ESP8266 хочет подключиться к серверу. Значения идентификатора и сокета извлекаются из «data - данных» и добавляются в локальный список ранее подключенных модулей ускорения ESP8266:

```
socket.on('connection', function (data)
 {
   var acc_ram = new Object();
   acc_ram.socket_id = socket.id;
   var data_json = ParseJson(JSON.stringify(data));
   acc_ram.acc_id = data_json.acc_id;
   acc_ram.socket = socket;
   addACCObject(acc_ram);
   printACC();
   for(var webBrowsers = 0; webBrowsers < webConnections.length;
webBrowsers++)
    {
      var sessionID = webConnections[webBrowsers].socket;
      sessionID.emit('acc_ram', { acc_ram: data })
    }
    });
```

Чтобы сообщить всем подключенным браузерам, что новый ACC готов к отправке данных, код внутреннего сервера отправляет им сообщение acc_ram и пересылает данные объекта, содержащий идентификатор модуля.

При получении сообщения браузер динамически построит графические элементы (холст для рисования, кнопки) для нового модуля.

Когда серверная часть получает тип сообщения JSON, который содержит значения для измеренного ускорения, сообщение пересылается во все браузеры, и значения станут входными данными для графики smoothie.js:

```
socket.on('JSON', function (data)
 {
 for(var webBrowsers = 0; webBrowsers < webConnections.length;
webBrowsers++)
 {

  // отправляем данные во все подключенные браузеры
  var sessionID = webConnections[webBrowsers].socket;

   sessionID.emit('acc_data', { acc_data: data });
 }
 });
```

Для сообщений, полученных с веб-страницы, которые адресованы определенному модулю ESP8266, нам нужно сначала идентифицировать целевой модуль на основе идентификатора сокета и отправить это сообщение в соответствующий модуль. В противном случае всем модулям ESP8266 будет отправлена команда сброса, и я уверен, что это не то, что вы намеревались сделать:

```
socket.on('resetModule' , function (data)
  {
    for(var j=0; j< accConnections.length; j++)
   {
   var acc_m = accConnections[j];
      if(acc_m.acc_id == data.acc_id)
   {
    var s = acc_m.socket; //Get the socket
     s.emit("resetModule", {message: data} );
     return; //======>
   }
  }
  });
```

Общедоступная веб-страница

Веб-страница, обслуживаемая вашим сервером Node.js, представляет собой простой SPA, разработанный с помощью BoostrapJS и AngularJS. Веб-страница попытается подключиться к серверу с помощью WebSocketconnection, а после этого отправит собственное регистрационное сообщение, чтобы сервер мог добавить свой сокет в свою базу данных.

Точка входа для значений ускорения от ESP8266, но пересылаемых внутренним сервером:

```
socket.on('acc_data', function(data)
 {
  for(var i=0; i < $scope.myacc_collection.length; i++)
 {
    if(data.acc_data.device_name == $scope.myacc_collection[i].acc_id)
    {
      $scope.myacc_collection[i].x = Number(data.acc_data.x);
      $scope.myacc_collection[i].y = Number(data.acc_data.y);
      $scope.myacc_collection[i].z = Number(data.acc_data.z);
      var currentDate = new Date().getTime();
      $scope.myacc_collection[i].axeX.append(currentDate,
Number(data.acc_data.x));
      $scope.myacc_collection[i].axeY.append(currentDate,
Number(data.acc_data.y));
      $scope.myacc_collection[i].axeZ.append(currentDate,
Number(data.acc_data.z));
       return;
```

```
    }
  }
});
```

Эта функция получает фактические данные с сервера и добавляет значения ускорения X, Y и Z в smoothie.js с помощью функции append - добавления.

Резюме

В этой главе мы завершили еще одну важную функцию, которая может быть реализована с помощью ESP8266, коммуникации в реальном времени. Вы узнали, как передавать данные ускорения в реальном времени с трехосного акселерометра ADXL345 на внутренний сервер Node.js, который пересылает полученные данные в подключенные браузеры. Данные отображаются в режиме реального времени с помощью smoothie.js. В продолжение этой главы я рекомендую вам хранить полученные данные в базе данных временных рядов, использовать Grafana в качестве инструмента отображения ваших значений и почему бы не отправить их через MQTT брокеру MQTT. Добавление базы данных и красивого инструмента для отображения значений может превратить это решение в коммерческое. Вы можете разработать платформу для хранения и отображения данных в реальном времени для других компаний или частных пользователей.

8

Добавление мобильного приложения в умный дом

В предыдущих главах мы обсуждали Blynk как цифровую панель управления для вашего проекта. Если вы хотите создать собственное мобильное приложение, которое подключается к вашему облаку MQTT, вы найдете в этой главе стартовый код приложения, который позволяет вам создать базовое мобильное приложение всего за несколько минут. Текущий код мобильного приложения позволяет вам войти на сайт http: // iotcentral.eu и управлять зарегистрированными устройствами.

В этой главе мы рассмотрим следующие темы:

- Установка Docker
- Получение образа разработки для одностраничного приложения
- Получение демонстрационного кода для мобильного приложения
- Получение кода смарт-сокета приложения для ESP8266
- Установка существующей прошивки для брокера ESP8266 MQTT
- Создание APK для устройств Android

Установка Docker и использование контейнеров

Образ контейнера - это легкий автономный исполняемый пакет программы, который включает в себя все необходимое для его запуска: код, среду выполнения, системные инструменты, системные библиотеки и настройки. Таким образом, вам не нужно устанавливать множество библиотек и программ. Вы можете использовать существующий образ и запускать из него контейнер.

Мы будем использовать контейнер с Ionic и Android SDK, который позволит вам разработать и протестировать мобильное приложение. В конце у вас будет APK-файл, который необходимо подписать с помощью консоли Google Play, после чего вы сможете распространить его среди миллионов людей.

В том же Ubuntu 16.04 в Virtual Box, который использовался ранее, мы установим и настроим Docker:

1. Установите ключ GPG в вашу систему:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
```

2. Добавьте репозиторий Docker к источникам APT:

```
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
```

3. Обновите базу данных пакетов с новым добавленным репозиторием:

```
sudo apt-get update
```

4. Установите Docker Community Edition из репозитория Docker вместо репозитория Ubuntu:

```
apt-cache policy docker-ce
```

Вывод показывает, что доступно несколько версий:

```
docker-ce:
  Installed: (none)
  Candidate: 17.09.0~ce-0~ubuntu
  Version table:
     17.09.0~ce-0~ubuntu 500
        500 https://download.docker.com/linux/ubuntu xenial/stable
        amd64 Packages
     17.06.2~ce-0~ubuntu 500
        500 https://download.docker.com/linux/ubuntu xenial/stable
        amd64 Packages
     17.06.1~ce-0~ubuntu 500
        500 https://download.docker.com/linux/ubuntu xenial/stable
        amd64 Packages
     17.06.0~ce-0~ubuntu 500
        500 https://download.docker.com/linux/ubuntu xenial/stable
        amd64 Packages
     17.03.2~ce-0~ubuntu-xenial 500
        500 https://download.docker.com/linux/ubuntu xenial/stable
        amd64 Packages
```

```
        17.03.1~ce-0~ubuntu-xenial 500
           500 https://download.docker.com/linux/ubuntu xenial/stable
           amd64 Packages
        17.03.0~ce-0~ubuntu-xenial 500
           500 https://download.docker.com/linux/ubuntu xenial/stable
           amd64 Packages
```

5. Теперь установите Docker:

```
sudo apt-get install -y docker-ce
```

6. Проверьте установленную версию Docker:

```
sudo docker version
Client:
 Version:      17.09.0-ce
 API version:  1.32
 Go version:   go1.8.3
 Git commit:   afdb6d4
 Built:        Tue Sep 26 22:42:18 2017
 OS/Arch:      linux/amd64
Server:
 Version:      17.09.0-ce
 API version:  1.32 (minimum version 1.12)
 Go version:   go1.8.3
 Git commit:   afdb6d4
 Built:        Tue Sep 26 22:40:56 2017
 OS/Arch:      linux/amd64
 Experimental: false
```

7. Запускать демон Docker при каждой загрузке:

```
sudo systemctl status docker
docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor
preset: enabled)
   Active: active (running) since Mon 2017-10-23 20:41:43 CEST; 46min
ago
     Docs: https://docs.docker.com
 Main PID: 12117 (dockerd)
   CGroup: /system.slice/docker.service
           ├─12117 /usr/bin/dockerd -H fd://
           └─12141 docker-containerd -l
unix:///var/run/docker/libcontainerd/docker-containerd.sock --metrics-
interval=0 --start-timeout 2m --state-dir /var/run/docker/libcontainerd
```

8. Чтобы запускать команды докеров без sudo перед ними, нам нужно:

```
sudo usermod -aG docker ${USER}
```

9. Теперь вам нужно выйти и повторно войти в систему, чтобы иметь возможность запускать команды докера:

```
su - ${USER}
```

10. Убедитесь, что ваш пользователь входит в группу Docker:

```
id
```

Группа Docker должна быть в выводе команды:

```
uid=1000(catalin) gid=1000(catalin)
groups=1000(catalin),4(adm),20(dialout),
24(cdrom),26(tape),27(sudo),29(audio),30(dip),44(video),46(plugdev),
109(netdev),119(scanner),120(lpadmin),121(sambashare),998(docker)
```

Теперь Docker установлен и настроен для запуска при каждом запуске виртуальной машины. Чтобы завершить настройку мобильного приложения, нам нужно получить изображение, которое будет использоваться для создания и разработки мобильного приложения, и некоторый начальный код для мобильного приложения.

Получение проявочного образа

Чтобы получить образ разработки из репозитория Docker, введите команду:

```
sudo docker pull agileek/ionic-framework
```

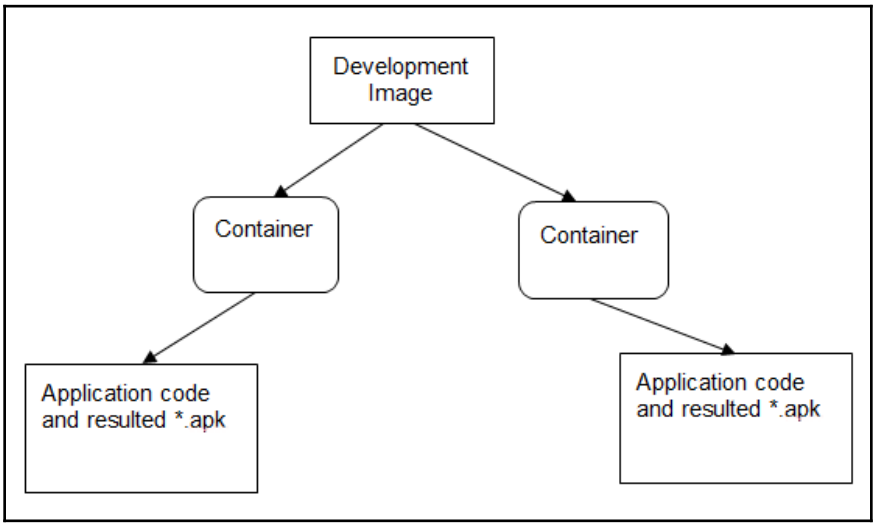
Это займет некоторое время, так что наберитесь терпения, но время, затрачиваемое на установку всех зависимостей и всех необходимых пакетов, по сравнению с исправлением ошибок больше, чем эта загрузка.

В конце образ будет в локальном репозитории. Вы можете увидеть, какие изображения находятся в вашем текущем репозитории, выполнив команду, как показано здесь:

```
catalin@plex:~$
catalin@plex:~$
catalin@plex:~$ docker images
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
agileek/ionic-framework   latest              559594ff121f        11 hours ago        4.5GB
catalin@plex:~$
catalin@plex:~$
```

Образы докеров

Из образа разработки вы можете указать, сколько контейнеров вы хотите. В каждом контейнере вы можете разрабатывать одно приложение полностью отдельно от другого. Код приложения находится вне контейнера, но он доступен для контейнера, чтобы скомпилировать его и запустить внутри контейнера. Результатом этого этапа будет приложение, которое будет запускаться в вашем браузере. В конце вы можете создать файл *.apk и протестировать его на своем мобильном телефоне:



Теперь, когда у нас построена инфраструктура, давайте возьмем код из GitHub и запустим с ним контейнер:

```
mkdir ~/PROJECTS
cd ~/PREOJECTS
git clone https://github.com/bcatalin/Homy4
```

Теперь в каталоге Homy4 есть код, который позволяет пользователю вводить свои учетные данные для учетной записи http://iotcentral.eu. После аутентификации приложение получит данные о подключении, назначенные для этой учетной записи, и подключится к iotcentral.eu через WebSockets.

Запустите контейнер, используя существующий код. По сути, каталог, в котором находится код, внутренне сопоставляется с контейнером с помощью сопоставления томов:

```
    docker run -it -p 8100:8100 -p 35729:35729 --name espbook -v
/home/catalin/PROJECTS/Homy4/:/myApp:rw agileek/ionic-framework
```

В команде используются следующие ключи:

- run: он указывает Docker на запуск нового контейнера.
- -p 8100: 8100: Внутренний порт контейнера 8100 внешне отображается на порт 8100. Он будет доступен позже в браузере. Если вы хотите запустить несколько контейнеров, измените внешний порт на другой. В этом случае команда станет -p 8101: 8100.
- -p 35729:35729: этот порт используется для перезагрузки в реальном времени. Если вы измените код (вне контейнера), приложение, работающее внутри контейнера, автоматически перезагрузится. Опять же, если вам нужно запустить несколько экземпляров контейнеров из одного образа, не забудьте изменить порт.
- --name: вы можете дать контейнеру имя с числами, например5961e5e90592. Лучше давать явное имя, чтобы потом можно было вспомнить, для чего оно использовалось.
- /home/catalin/PROJECTS/Homy4/: по этому пути на хост-машине существует загруженный код из Git.
- /myApp: это каталог из контейнера, в котором код будет виден приложение в контейнере.
- rw: монтирование между хост-системой и контейнером осуществляется на чтение.и запись
- agileek/ionic-framework: это имя изображения из вашего локального репозитория.

После создания контейнера вы можете увидеть его состояние с помощью этой команды:

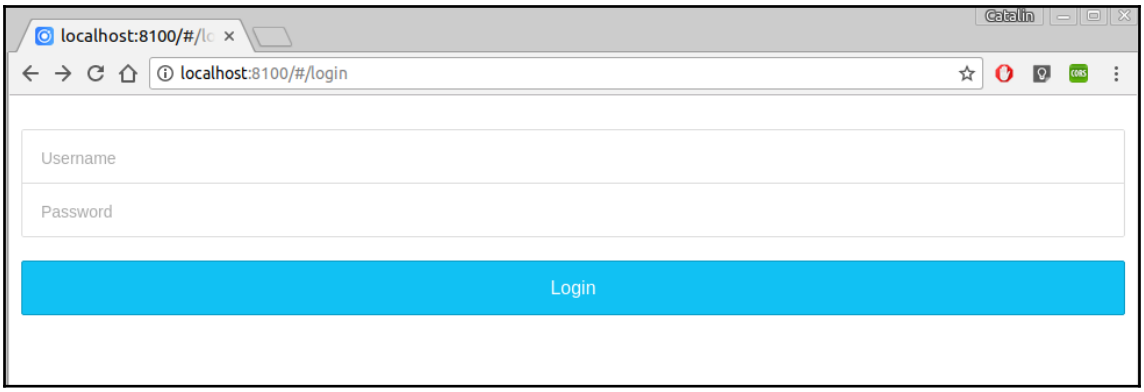
```
docker ps
```

```
catalin@lubuntuBIG:~$
catalin@lubuntuBIG:~$ docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS                                                  NAMES
5961e5e90592        agileek/ionic-framework   "ionic serve"            3 weeks ago         Up About an hour    0.0.0.0:8100->8100/tcp, 0.0.0.0:35729->35729/tcp   espbook
08b7ce62f41b        registry:2             "/entrypoint.sh /etc/"   5 weeks ago         Up About an hour    0.0.0.0:5000->5000/tcp                             registry
catalin@lubuntuBIG:~$
catalin@lubuntuBIG:~$
```

Чтобы просмотреть все запущенные и остановленные контейнеры, вы можете использовать следующую команду:

```
docker ps –a
```

Теперь, когда контейнер запущен, откройте браузер на хосте и перейдите к http: // локальный: 8100. Вы должны увидеть веб-страницу, загруженную в браузер, предлагающую вам ввести имя пользователя и пароль:

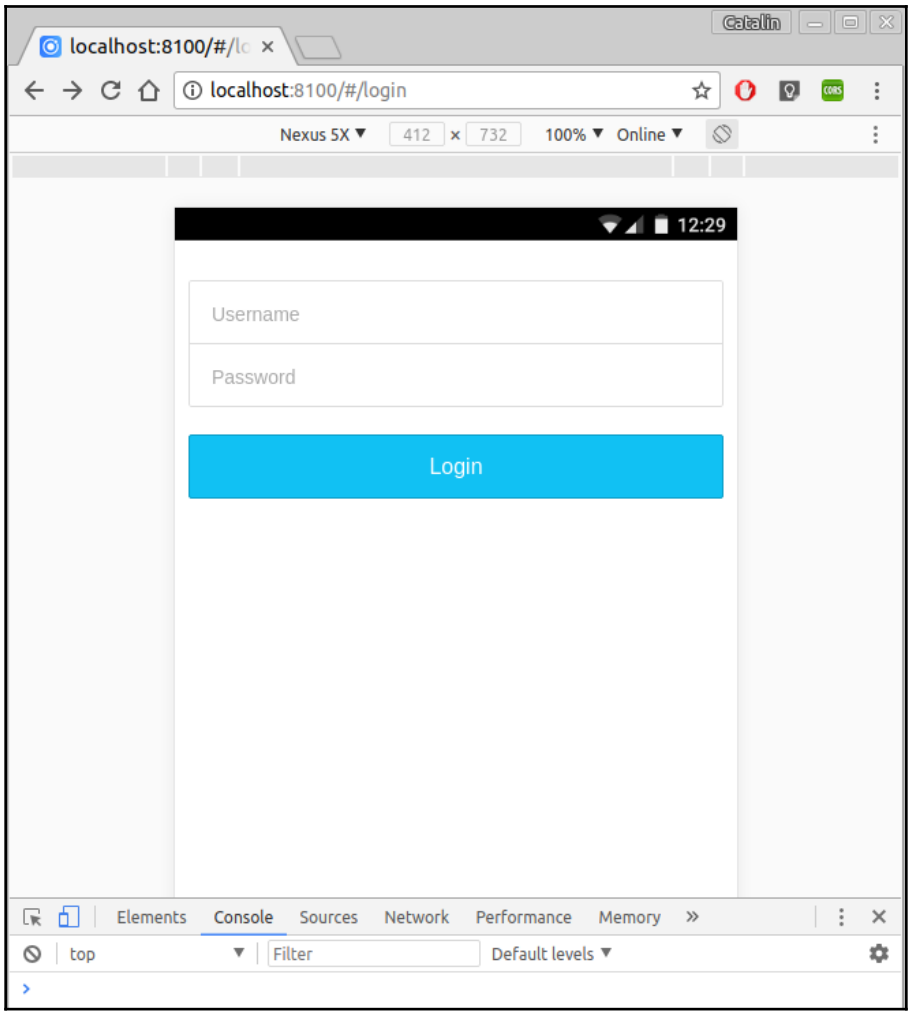


Теперь размер приложения отформатирован для браузера, но, поскольку вы разрабатываете мобильное приложение, было бы неплохо увидеть, как именно оно выглядит на мобильном телефоне. Для этого вам нужно ввести его в режиме Developer Tools. Для этого есть много режимов:

- Щелкните правой кнопкой мыши в браузере и выберите «Inspect» - Проверить.
- Используйте комбинацию клавиш Ctrl + Shift + I
- В меню выберите More Tools - Дополнительные инструменты, а затем выберите Developer Tools - Инструменты разработчика.

После того, как вы войдете в Developer Tools - Инструменты разработчика, вы можете изменить внешний вид страницы, используя комбинацию Ctrl + Shift + M, или вы можете щелкнуть изображение, похожее на телефон и планшет, влевом углу.

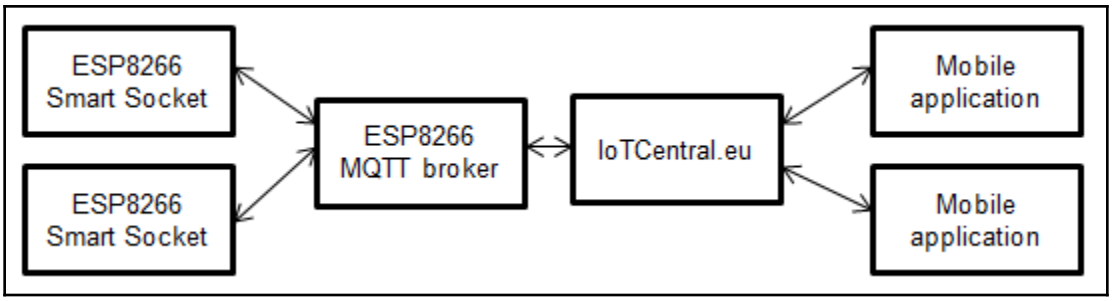
Теперь макет должен быть таким:



На верхней панели вы можете выбрать, как это выглядит на Nexus 5X, Nexus P или iPhone, или вы можете изменить ориентацию экрана и многие другие параметры.

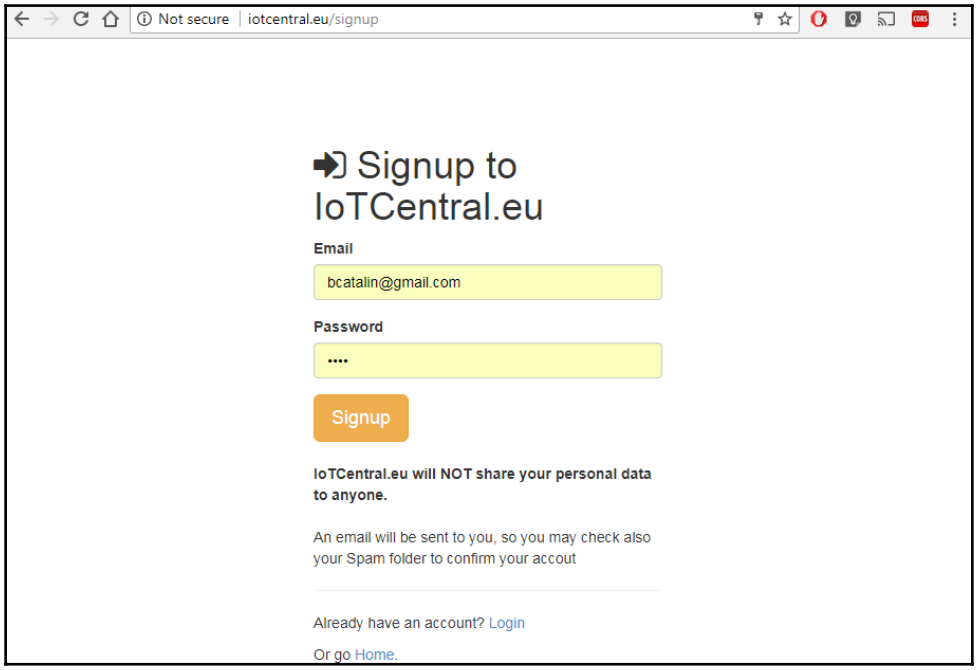
Настройка местного брокера

Теперь, когда все готово, давайте обсудим, чего мы хотим достичь в этой главе. Позже в этой главе мы вернемся к мобильному приложению:

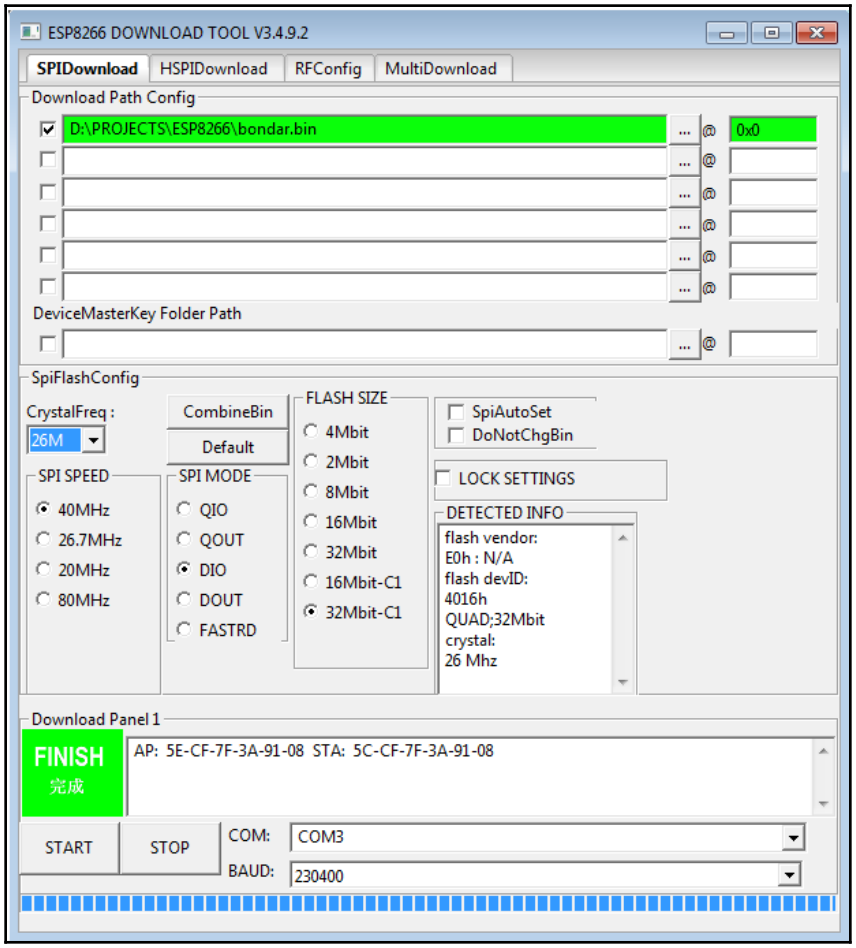


Теперь у вас достаточно знаний, чтобы завершить всю систему от кода ESP8266 до вашего местного брокера и мобильного приложения. Чтобы завершить эту цепочку, мы можем начать с ESP8266. Давайте создадим простой код плагина и воспользуемся услугой PaaS, предлагаемой iotcentral.eu, в качестве сервиса авторизации и облачного MQTT:

1. Создайте учетную запись и подтвердите свой email на iotcentral.eu:



2. Получите двоичный образ bondar.bin локального брокера MQTT и запишите его на плату nodeMcu 4 Мб. Выключите и снова включите плату:



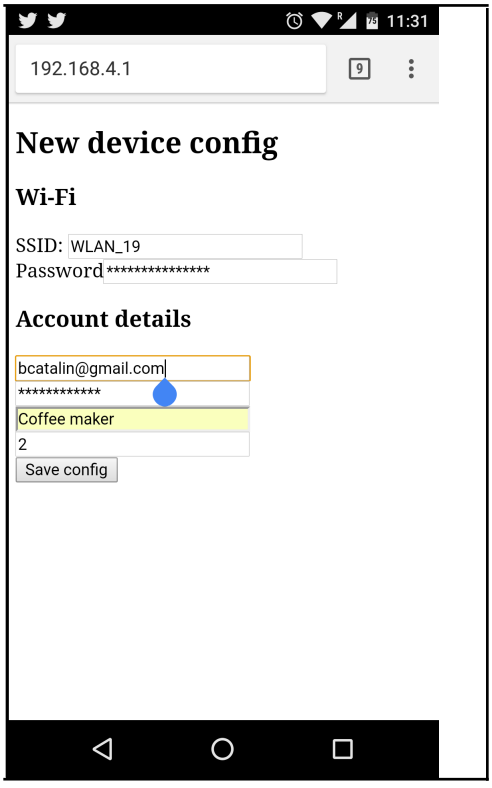
Используйте свое мобильное приложение для поиска точки доступа с именем Bondar_XXXXXX, где XXXXXX - это последние шесть цифр MAC-адреса.

4. Подключитесь к нему и используйте пароль 12345678, если он запрашивается. Перейдите в браузере телефона к адресу 192.168.4.1 и на веб-странице введите те же учетные данные, которые вы использовали для учетной записи iotcentral.eu. Местный брокер подключится к платформе iotcentral.eu и зарегистрируется.

На странице ваших устройств на iotcentral.eu вы должны увидеть вашего недавно зарегистрированного местного брокера. Использование локального брокера на ESP8266 не обязательно, но это упрощает ваш код на ESP. Если вы решите не использовать местного брокера, вам необходимо напрямую подключиться к облачному брокеру MQTT на iotcentral.eu. В этом случае нам нужно получить выделенную тему и добавить в код больше логики, чтобы справиться с этим:

На https: // github / bcatalin / demoapp есть код, который мы будем использовать в качестве кода ESP8266. Получите код и запишите его в ESP8266.

После перезагрузки ESP8266 подойдите к телефону и найдите точку доступа ESPap. Подключитесь к нему и перейдите в браузере по адресу http: //192.168.4.1, как вы это делали для локального брокера EP8266 MQTT. На странице, представленной ESP8266, введите учетные данные Wi-Fi, имя пользователя и пароль, используемые для iotcetral.eu, имя AmazonAlexa, которое вы хотите вызвать для своего модуля, чтобы включить или выключить что-то, и ваш часовой пояс (это не реализовано , вы можете просто сделать это как упражнение):



Характеристики кода ESP8266

Запросы на код ESP8266 и вам необходимо подписаться на темы:

/<TOPIC>/plug/command or <TOPIC>/plug/command

В этом случае вы подключаете ESP напрямую к iotcentral.eu.

Требуемое сообщение, которое должно быть получено ESP8266 для запуска его GPIO 12:

{"device_name":"ESP_3A9108", "type":"plug", "state":1}

Здесь:

- Device_name: имя устройства, которое мы хотим активировать.
- Type: тип устройства. В данном случае это вилка.
- State: желаемое состояние GPIO 12. В этом случае мы хотим изменить состояние на *ON - ВКЛ.*

/<TOPIC> /status/update or <TOPIC>/status/update

В этом случае есть прямое подключение к iotcentral.eu без местного брокера. При получении любого сообщения по этой теме ESP8266 отправит свой статус и сообщения о статусе устройства. Мобильное приложение отправляет сообщение:

{"1":"1"}

По этой теме, когда он запускается, чтобы получить статус и состояние со всех ваших устройств. Вам нужно будет публиковать сообщения по этим темам:

/<TOPIC>/plug/status or <TOPIC>/plug/status

Содержание опубликованного сообщения должно иметь следующий формат:

{"device_name":"ESP_3A9108", "type":"plug", "state":1}

Здесь:

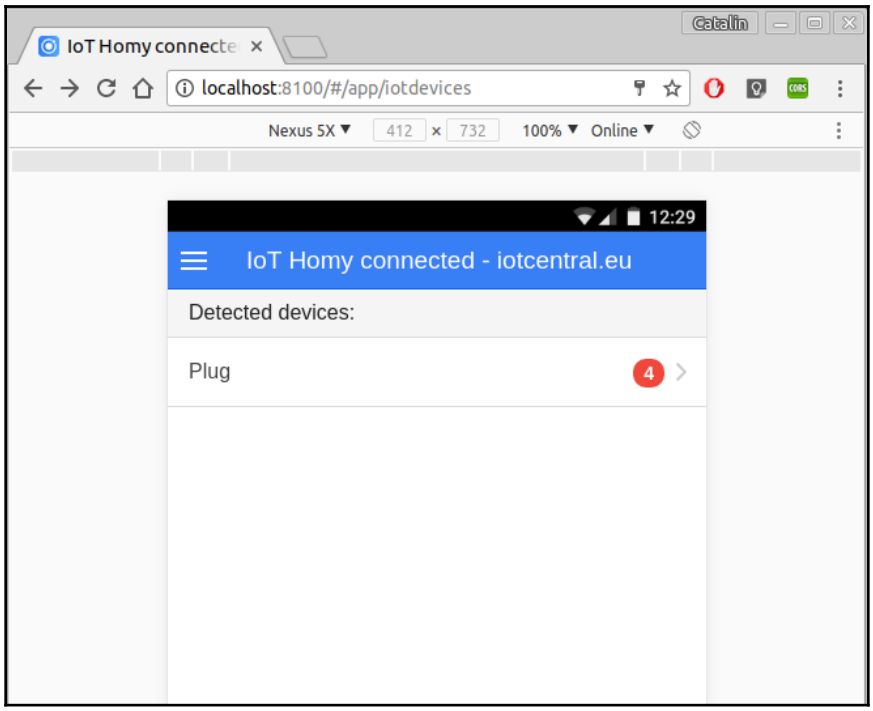
- Device_name: имя, созданное приложением на основе MAC-адреса.
- Type: тип устройства. В данном случае это вилка.
- State: текущее состояние 12-контактного разъема GPIO. В этом случае 1 означает ВКЛ

/<TOPIC>/device/status or <TOPIC>/device/statuswithout local broker.

Сообщение, опубликованное по этой теме, описывает статус этого устройства (IP-адрес, версия SDK, время безотказной работы и т. д.):

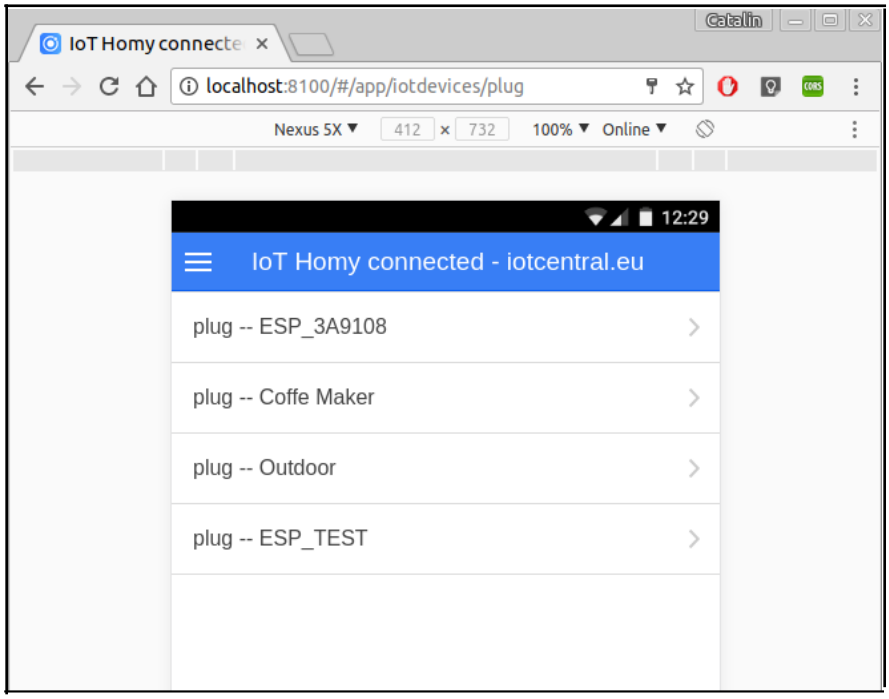
```
{"device_name":"ESP_3A9108","type":"plug","ipaddress":"192.168.8.222","alexa":"Coffee maker","bgn":3,"sdk":"1.5.3(aec24ac9)","version":"1039","uptime":"0 days00:40:12"}
```

На основе полученных сообщений мобильное приложение строит свой интерфейс и отображает устройства, группируя их по типу:

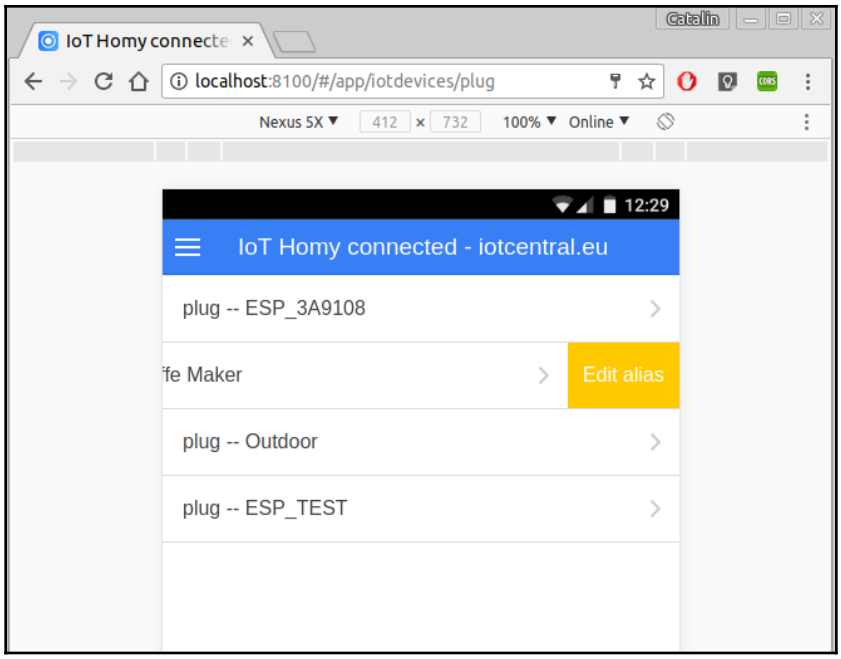


Мобильное приложение, подключеное к iotcentral.eu, обнаружило четыре устройства с одинаковыми типами Plug. Щелчок по типам Plug покажет все устройства, найденные для этого типа.

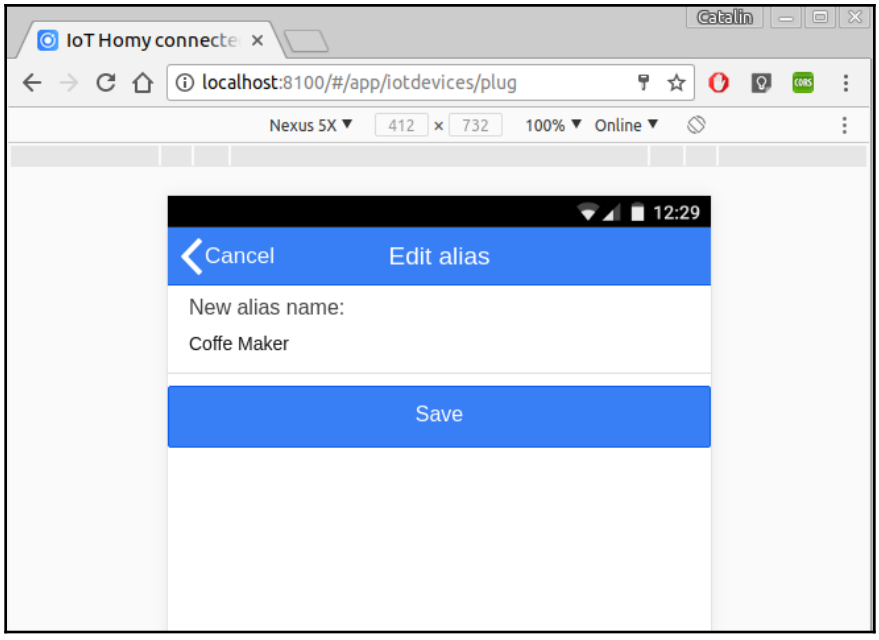
Если вы видите сообщение в консоли о том, что доступ был заблокирован из-за заголовка с перекрестным происхождением, установите плагин Chrome, который позволяет вам выполнять вызовы с перекрестным происхождением. Затем найдите расширение с именем Allow-Control-Allow-Origin. Проблемы с перекрестным происхождением не будут существовать на вашем мобильном телефоне:



В расширенном списке для всех устройств, у которых есть типовой штекер, есть еще наша Coffee Maker - кофеварка. Здесь, в начале, имя устройства отображается как ESP_, за которым следуют последние шесть цифр MAC-адреса, но вы можете дать ему понятное имя, переставив нужный штекер влево и увидев кнопку Edit alias - Изменить ярлык:

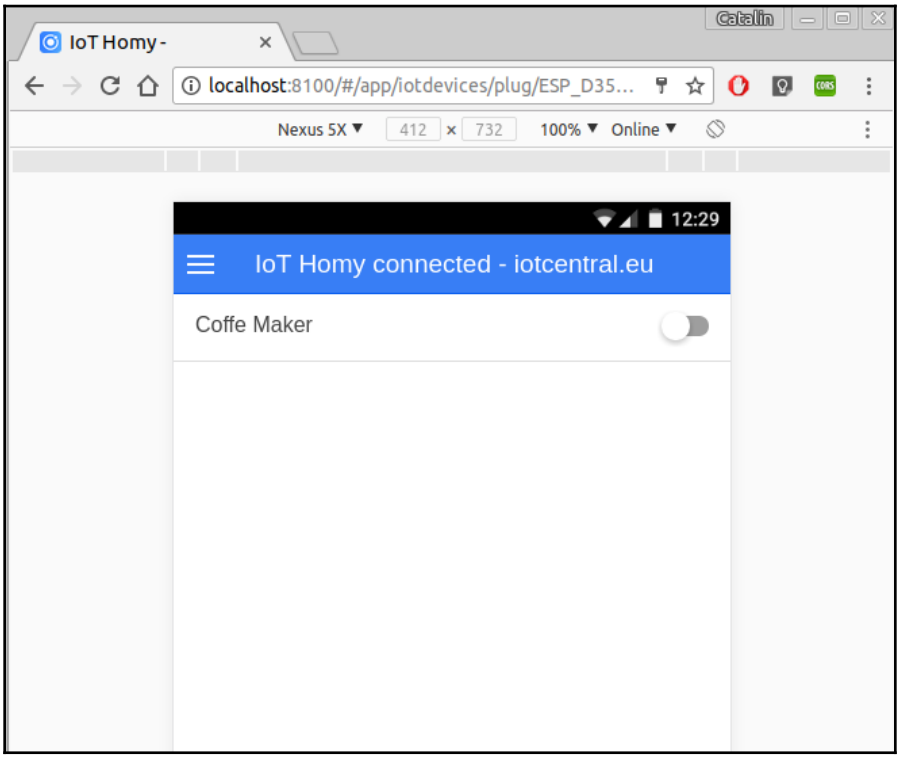


Вы можете изменить закодированное имя на более удобное:

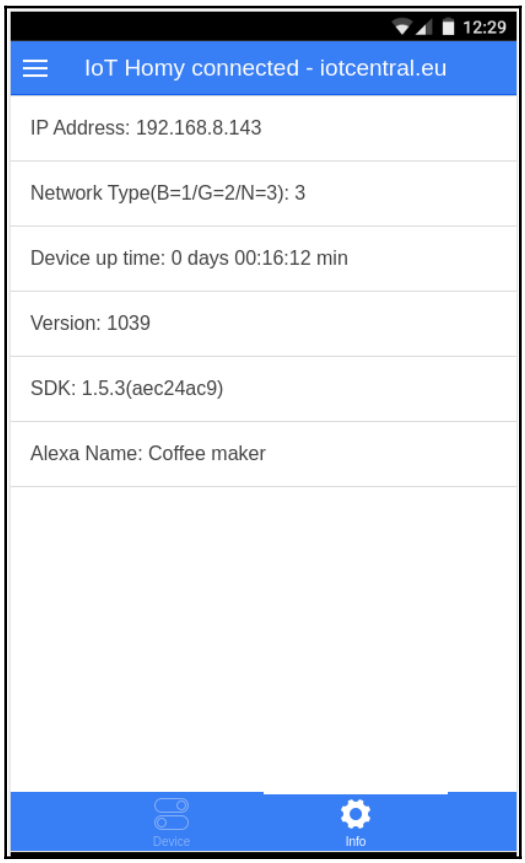


Нажатие кнопки « Save - Сохранить» сохранит новое имя ярлыка в локальном хранилище приложения и будет отображаться каждый раз, когда приложение обнаруживает это устройство.

Если вы выбираете из всех устройств, с одним и тем же типом Coffee Maker - кофеварки, мобильное приложение покажет страницу действий, с которой вы можете включить или выключить штекер (на самом деле GPIO 12):



В нижней части экрана вы увидите две вкладки, одна для статуса устройства, а другая для деталей устройства:



Вы можете увидеть содержание сообщения, опубликованного в теме <TOPIC> / device / status, подключенный IP-адрес ESP8266, время его работы, версию программного обеспечения, версию SDK, с которой он был скомпилирован, и понятное имя, присвоенное во время Начальная настройка.

После того как вы изменили мобильное приложение, добавив новые устройства или изменив внешний вид, вам необходимо создать приложение .apk, которое необходимо установить на ваш телефон.

Для этого сначала вам нужно авторизоваться в контейнере с помощью следующей команды:

```
docker exec -it espbook bash
```

Эта команда войдет в контейнер с именем espbook и запустит оболочку bash.

Перейдите в / myApp / www и введите команду, которая создаст APK:

```
ionic cordova build android
```

Через 30 секунд у вас будет APK, готовый для тестирования на реальном телефоне по адресу: / myApp / platform / android / build / output / apk.

Чтобы использовать apk-файл, необходимо подписать его. В Интернете есть множество руководств о том, как это сделать и как опубликовать мобильное приложение в Google Play.

Это демонстрационное мобильное приложение теперь в Google Play. Найдите Homy4 и установите его. Не забудьте сначала создать учетную запись на iotcentral.eu и использовать demoapp (демонстрационное приложение) на ESP8266.

Summary

В этой главе вы узнали, как использовать Docker для создания среды разработки мобильного приложения. В аспекте ESP8266 вы изучили темы и сообщения, которые должен иметь модуль, чтобы иметь возможность интегрироваться в систему, состоящую из облачного MQTT и мобильного приложения. Код для ESP8266 в демонстрационном приложении слишком велик, чтобы его можно было здесь представить, но он охватывает все главы книги о ESP8266: как подключиться к Wi-Fi, как хранить данные в SPIFFS, как подключиться к сломанному MQTT, и как подписаться и публиковать сообщения по темам. Это может быть хорошим началом для устройства Smart Plug, поскольку это полнофункциональный пример.
В мобильной части код, который существует в примере GitHub, полностью работает с платформой iotcentral.eu и с демонстрационным приложением. Глядя на код, чтобы узнать больше о том, как разработать мобильное приложение с помощью Ionic и Cordova, вы легко сможете добавить новые устройства, такие как мониторинг температуры, управление кондиционерами и управление внутренним оборудованием. Теперь у вас есть все знания, чтобы превратить свой дом в умный. Используя ESP8266 в качестве облачных вычислений, теперь вы можете создать мультиарендное облако MQTT и даже умное мобильное приложение. Обладая всеми этими знаниями, вы даже можете начать создавать свой собственный продукт и продавать его на рынке, поскольку вы не зависите от какой-либо другой платформы, и все находится в вашей области знаний.