**ПРОЕКТ**

«Удалённая лаборатория»

Проект находится в стадии разработки.

Оглавление

[1. Удалённое управление 3](#_Toc189296969)

[1.1. Общая идея реализации 3](#_Toc189296970)

[1.2. Простейший пример работы в локальной сети 3](#_Toc189296971)

[1.2.1. Подготовка 3](#_Toc189296972)

[1.2.2. Реализация 4](#_Toc189296973)

[1.2.3. Демонстрация 5](#_Toc189296974)

[1.3. Более сложный случай 6](#_Toc189296975)

[3. Архитектура взаимодействия между ESP и Arduino через UART 9](#_Toc189296976)

[3.1. Введение 9](#_Toc189296977)

[3.2. Сравнение протокола Modbus с кастомным 9](#_Toc189296978)

[3.3. Кастомный протокол 10](#_Toc189296979)

[3.3.1. Структура пакета 10](#_Toc189296980)

[3.3.2. Алгоритм кодирования (на стороне ESP) 10](#_Toc189296981)

[3.3.3. Код программы 11](#_Toc189296982)

[3.3.4. Демонстрация 11](#_Toc189296983)

[3.4. Реализация через Modbus 11](#_Toc189296984)

[3.4.1. Решение проблем 11](#_Toc189296985)

[3.4.2. Код программы 11](#_Toc189296986)

[3.4.3 Демонстрация 11](#_Toc189296987)

[4. Приложения 12](#_Toc189296988)

[4.1. Техническое описание модуля микроконтроллера ESP8266 12](#_Toc189296989)

[5. Источники 13](#_Toc189296990)

[5.1. Простым языком об HTTP 13](#_Toc189296991)

[5.2. Веб-фреймворк Django (Python) 13](#_Toc189296992)

[5.3. Установка ESP8266 в Arduino IDE (руководство для ОС Windows) 13](#_Toc189296993)

# 1. Удалённое управление

## 1.1. Общая идея реализации

Сначала разберем способ «на пальцах», а потом перейдем к примеру, который всё прояснит.

Для удаленного управления машинкой будет использоваться механизм связи по сети Wi-Fi. Этот метод работает медленнее чем Bluetooth, однако только с его помощью можно будет управлять машинкой из любой точки мира. Таким образом, для нашего проекта необходимо наличие Wi-Fi маршрутизатора в зоне работы машинки.

Подключение машинки обеспечивает плата микроконтроллера ESP8266 (см. [Приложение 1](#_4.1._Техническое_описание)). Она будет получать запросы по протоколу HTTP (см. [Источник 1](#_5.1._Простым_языком)), обрабатывать их и отправлять ответы (в том числе и телеметрию).

При подключении платы к сети Wi-Fi она получает свой уникальный IP-адрес, с помощью которого можно будет управлять конкретной машинкой. Все запросы посылаются изнутри локальной сети, т. е. с устройств, подключенных к той же сети Wi-Fi, что и машинка, и имеют подобный вид: *http://<IP\_платы>/<команда\_машинке>*. Наша плата будет запрограммирована принимать такие запросы и давать соответствующие инструкции машинке. Таким образом и будет организовано удалённое управление.

Для удобного и централизованного управления машинками предлагается также создать графический интерфейс, в нашем случае это веб-приложение на Django (см. [Источник 2](#_5.1_Веб-фреймворк_Django)). Приложение будет представлять из себя обычный вебсайт, на котором можно будет «подключиться» к определенной машинке и управлять ей, например нажимая кнопки на клавиатуре, или просто выбирая нужные команды на экране. Нужные запросы машинке будут отправляться автоматически в соответствии с действиями пользователя.

Для того чтобы обеспечить удалённый доступ к машинкам за пределами локальной сети, необходимо сделать наше приложение доступным в интернете. Этот пункт будет рассмотрен подробнее далее (см TODO…).

## 1.2. Простейший пример работы в локальной сети

Рассмотрим простейший случай удалённого управления машинкой – будем включать/выключать встроенный светодиод платы ESP8266.

### 1.2.1. Подготовка

Оборудование:

* Модуль ESP8266 CH340 NodeMCU
* Кабель USB Type-c
* Компьютер с установленной средой разработки Arduino IDE версии 1.8.0 и выше
* Сеть Wi-Fi с известным именем (SSID) и паролем

Перед тем как писать скетч нужно подготовить среду разработки, т.к. по умолчанию она не умеет работать с платами ESP8266. Информация о том, как это сделать, подробно изложена в [Источнике 3](#_5.1_Установка_ESP8266). После подготовки среды можно начинать писать код…

### 1.2.2. Реализация

Код с подробными комментариями, который необходимо записать на микроконтроллер ESP8266:

// Библиотека для работы с платой

#include <ESP8266WiFi.h>

// Библиотека для создания сервера, прослушивающего HTTP запросы

#include <ESP8266WebServer.h>

// Данные сети Wi-Fi

const char\* ssid = "Nepike";

const char\* password = "123453119670";

// Создаем объект server, который будет слушать HTTP-запросы на порту 80 (стандартный порт HTTP)

ESP8266WebServer server(80);

// Создаем объект localIp для хранения локального IP-адреса устройства.

IPAddress localIp;

const int ledPin = 2; // Встроенный светодиод подключен к GPIO-2

void setup() {

  Serial.begin(115200); // Инициализируем последовательный порт для отладки со скоростью 115200 бод

  pinMode(ledPin, OUTPUT); // Установка GPIO-2 как выход

  digitalWrite(ledPin, HIGH); // Выключаем светодиод

  // Подключение к Wi-Fi

  WiFi.begin(ssid, password);

  // Ожидаем пока плата успешно подключится

  while (WiFi.status() != WL\_CONNECTED) {

    delay(1000);

    Serial.println("connecting Wi-Fi...");

  }

  Serial.println("connected Wi-Fi!");

  // Получаем IP-адрес устройства и выводим его в консоль

  localIp = WiFi.localIP();

  Serial.print("IP: ");

  Serial.println(localIp); // Именно по этому адресу мы будем обращаться к машинке

  // Настройка HTTP-обработчиков

  server.on("/lumos", HTTP\_GET, handle\_lumos); // при запросе http://<localIp>/lumos вызывается функция handle\_lumos

  server.on("/nox", HTTP\_GET, handle\_nox); // при запросе http://<localIp>/nox вызывается функция handle\_nox

  server.begin(); // Запускаем HTTP-сервер

}

void loop() {

  // Проверяем, есть ли входящие HTTP-запросы, и вызываем соответствующие обработчики

  server.handleClient();

}

// Функции, обрабатывающие запросы

void handle\_lumos() {

  digitalWrite(ledPin, LOW); // Включаем светодиод

  String response = "LED on! IP: " + localIp.toString(); // Формируем сообщение для ответа

  server.send(200, "text/plain", response); // Отправляем ответ

}

void handle\_nox() {

  digitalWrite(ledPin, HIGH); // Выключаем светодиод

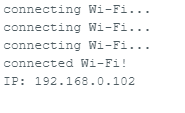
  String response = "LED off! IP: " + localIp.toString(); // Формируем сообщение для ответа

  server.send(200, "text/plain", response); // Отправляем ответ

}

### 1.2.3. Демонстрация

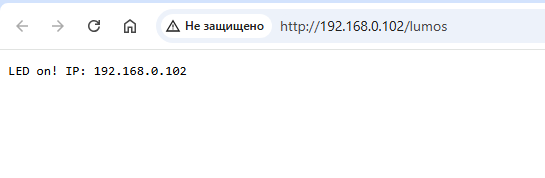
После записи скетча на плату переподключаем её к компьютеру и записываем IP полученный после подключения к Wi-Fi:



Теперь можем отключить плату от компьютера и проверить удалённую работу светодиода, посылая HTTP-запросы через браузер с компьютера, подключенного к указанной в коде сети Wi-Fi. Для начала подключим плату к источнику питания:



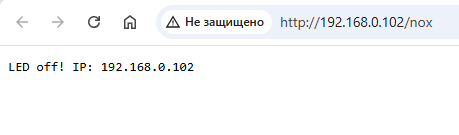
Обратимся по адресу: <http://192.168.0.102/lumos> , ожидая что светодиод загорится:



На загрузившейся странице мы видим сообщение об успешном выполнении команды, что и можем наблюдать на плате:



Теперь попробуем выключить светодиод, отправив запрос: <http://192.168.0.102/nox> :



Опять же видим сообщение об успехе на странице, как и в реальности:

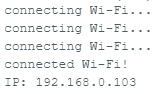


Как мы видим, наш способ удаленного управления полностью функционирует. Плата успешно получает и обрабатывает HTTP-запросы.

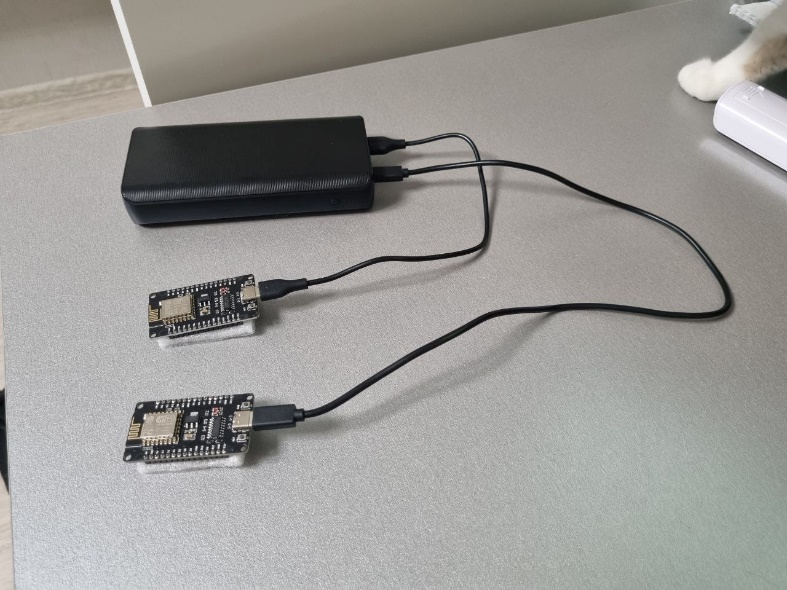
## 1.3. Более сложный случай

Теперь усложним наш пример, добавив в рассмотрение еще одну плату ESP8266 (потенциально еще одну машинку), а также напишем простейший графический веб-интерфейс на Django.

Для начала действуем ровно как в первом примере: запишем на вторую плату такой же скетч, а затем переподключим к компьютеру и запишем её IP-адрес:



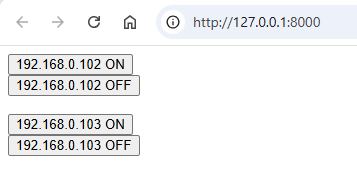
Подключим обе платы к источнику питания и перейдем к веб-интерфейсу:



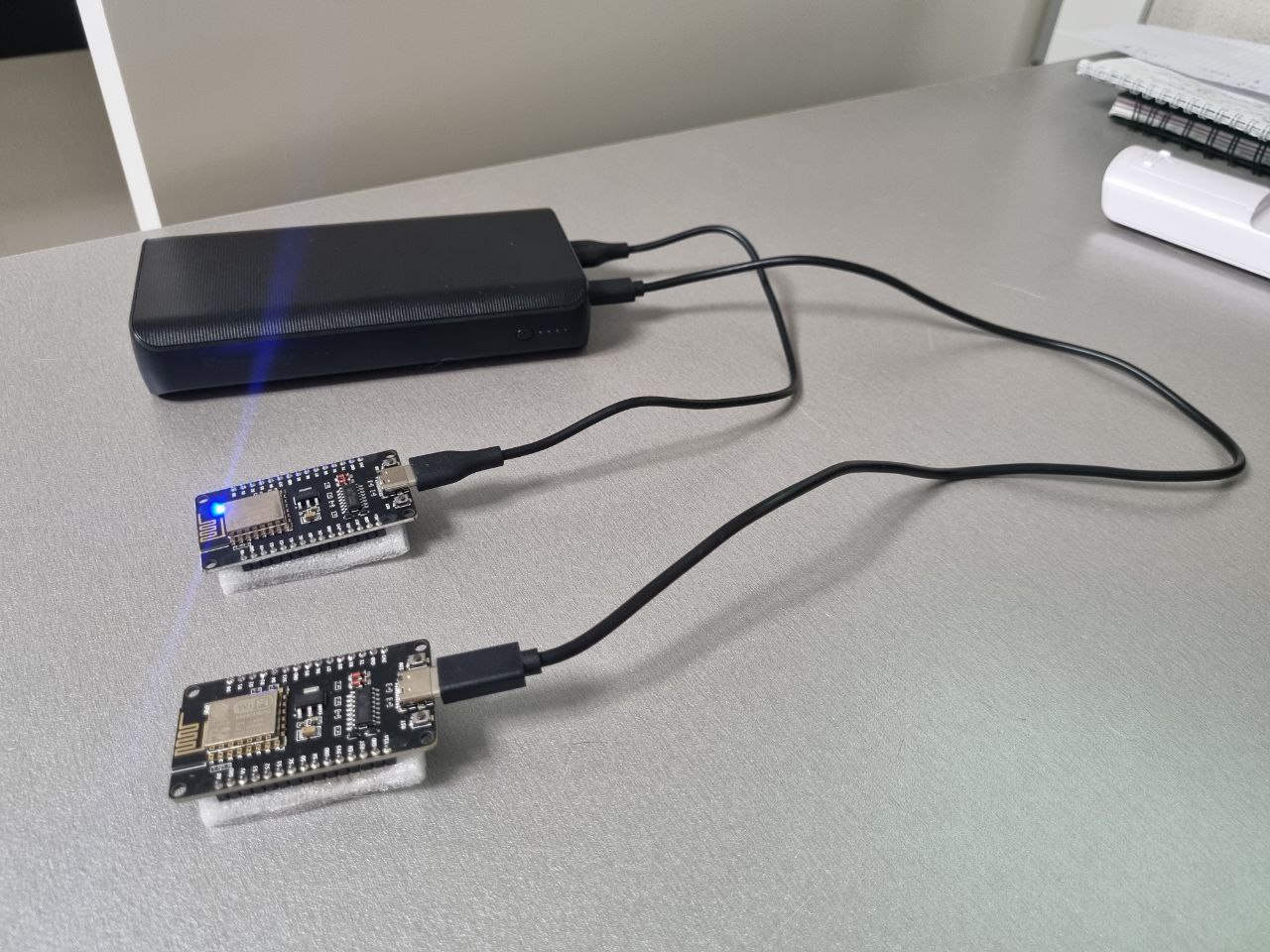
К сожалению, из-за сложности фреймворка я не смогу кратко и понятно прокомментировать здесь код веб-приложения, т.к., во-первых, программа достаточно большая и разделена на несколько файлов на разных языках программирования, а во-вторых, для понимания требуется хотя-бы базовое знание специфики фреймворка и веб-разработки, объяснение чего не является целью текущей работы. В любом случае, код находится на открытом GitHub репозитории: <https://github.com/Nepike/remote> , где в том числе рассказано, как запустить эту программу на своем устройстве.

Вместо того чтобы объяснять каждую строчку программы, я объясню идею того, что происходит. К счастью, происходят там совсем простые вещи.

Наш простейший веб-интерфейс представляет из себя 4 кнопки, по две кнопки на каждую из плат. Кнопки отвечают за включение/выключение светодиодов на платах:



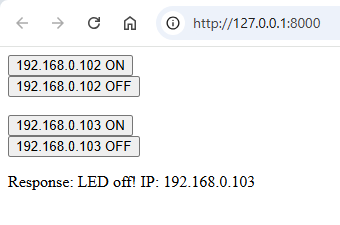
При нажатии кнопки, веб-приложение отправляет соответствующий запрос к одной из плат, например при нажатии на первую кнопку будет выполнен запрос <http://192.168.0.102/lumos> , что, как мы уже знаем, ведет к включению светодиода на первой плате:



Вторая плата точно так же успешно откликается на нажатие собственных кнопок:



Собственно, на этом и заканчивается принцип совместной работы плат и веб-интерфейса. Всё, для чего он нужен это удобная отправка HTTP-запросов соответствующим машинкам и получение от них ответов (в том числе телеметрии):



Конечно же, впоследствии веб-сайт будет усложняться, дорабатываться и обрастать новыми функциями и элементами взаимодействия с машинками, но принцип его работы будет таким же: отправка HTTP-запроса и получение ответа.

# 3. Архитектура взаимодействия между ESP и Arduino через UART

## 3.1. Введение

Наша цель – разработать систему управления периферийными устройствами (например машинкой или манипулятором) через микроконтроллер ESP8266, получая команды удалённо (через интернет). Ключевые требования:

* **Гибкость:** Возможность отправлять команды с произвольным количеством аргументов различных типов (bool, int, float, string).
* **Единообразие:** Одинаковый формат обмена для всех устройств, независимо от функционала.

ESP8266 выступает в роли “мозга” системы, преобразуя команды (например, в формате json) в низкоуровневые инструкции Arduino. Последние отвечают за выполнение действий и отправку данных обратно в интернет через ESP.

## 3.2. Сравнение протокола Modbus с кастомным

Для начала рассмотрим стандартизированный промышленный протокол связи - Modbus. Подробно про него можно почитать в приложении (см. TODO). Рассмотрим важные особенности этого протокола:

* **Фиксированные типы данных:** Только 16-битные регистры и однобитовые coils.
* **Фиксированная структура пакета:** Функции представляют из себя только чтение/запись регистров и битовые операции.
* **Slave-устройство должно знать структуру регистров:** Добавление нового параметра требует изменения прошивки Arduino.

Эти особенности создают проблемы для данного проекта: из-за жесткой структуры невозможно передать переменное число аргументов в одной команде, к тому же, совсем непонятно как работать с типами данных, занимающими больше 16 бит (например float или строки).

Чтобы избежать этого, предлагается в качестве альтернативы использовать кастомный протокол. Это намного лучше т.к., во-первых, это решает фундаментальные проблемы протокола Modbus, с которыми еще предстоит повоевать, а во-вторых, такое решение легко адаптировать под конкретные задачи проекта (например, можно добавлять новые типы данных без изменения структуры протокола).

В этом документе будет рассмотрена как реализация при помощи Modbus, так и с помощью кастомного протокола. В последствии, отталкиваясь от конкретных задач, примем решение, какой протокол нам подойдет больше.

Для начала рассмотрим идею самописного протокола.

## 3.3. Кастомный протокол

### 3.3.1. Структура пакета

Предлагается использовать такую структуру пакета:  
[Start Marker][Command][Data Type Flags][Payload Length][Payload][CRC]

* **Start Marker (2 байта):** Уникальная последовательность (например, 0xAA55), необходимая для синхронизации приемника и защиты от мусора в UART.
* **Command (1 байт):** Идентификатор команды (например, 0x01 – движение манипулятора).
* **Data Type Flags (N байт):** Битовая маска типов данных аргументов в payload.  
  Формат(байты): [кол-во аргументов][тип 1][тип 2]…[тип N].  
  Типы предлагается закодировать подобным образом:

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип данных** | **Код** | **Размер (байт)** | **Пример значения** |
| bool | 0x01 | 1 | 0x01 (true) |
| uint8\_t | 0x02 | 1 | 0x64 (100) |
| int16\_t | 0x03 | 2 | 0x00 0x96 (150) (big-endian) |
| uint16\_t | 0x04 | 2 | 0x003 0xE8 (1000) |
| float | 0x05 | 4 | 0x42 0x36 0x00 0x00 (45.5) (IEEE754 float) |
| char[] | 0x06 | N | 0x05 0x48 0x65 0x6C 0x6C 0x6F (“Hello”) |

Пример для [bool, int16, float]: [0x03 0x01 0x03 0x05]

* **Payload Length (1 байт):** Общий размер аргументов в байтах.
* **Payload (N байт):** Непосредственные значения аргументов, упакованные последовательно.
* **CRC (2 байта):** Контрольная сумма для проверки целостности данных (алгоритм CRC-16).

### 3.3.2. Алгоритм кодирования (на стороне ESP)

1. **Прием команды из интернета (например, JSON):** {“cmd”: “SET\_POSITION”,  
    “params”: [true, 150, 45.5]}
2. **Определение идентификатора команды:**  
    “SET\_POSITION” → 0x02 (например)
3. **Определение типа данных:**  
    true → bool (0x01)  
    150 → int16 (0x03)  
    45.5 → float (0x05)  
   Data Type Flags: 0x03 0x01 0x03 0x05
4. **Упаковка Payload:**  
    true → 0x01  
    150 → 0x00 0x96  
    45.5 → 0x42 0x36 0x00 0x00
5. **Расчет CRC для всего пакета.**

Алгоритм декодирования (на стороне Arduino) – очевиден (последовательное чтение данных согласно правилам записи).

### 3.3.3. Код программы

В файлах проекта. (если нужно – перенесу сюда)

### 3.3.4. Демонстрация

…

## 3.4. Реализация через Modbus

### 3.4.1. Решение проблем

Modbus работает только с 16-битными регистрами, а значит единственным выходом будет преобразовать все параметры в этот формат (рассмотрим на примере той же команды):

* true (bool) → 0x00 0x01
* 150 (in16) → 0x00 0x96
* 45.5 (float) → Нужно разбить на два регистра [0x42 0x36] [0x00 0x00]

Со строками интереснее – придется использовать разное количество регистров, в зависимости от длины строки:

* “Hello” → 0x48 0x65 0x6C 0x6C 0x6F – имеем 5 байт информации, а значит нужно 3 регистра:
  + Регистр 1: [0x48 0x65]
  + Регистр 2: [0x6C 0x6C]
  + Регистр 3: [0x6F 0x00] (добавлен нулевой байт)
  + + еще один регистр для передачи длины строки!

Отлично, теперь мы умеем передавать любой тип данных, однако некоторые ограничения все же остались:

Во-первых, сохраняется жесткая привязка к структуре регистров, т.е. принимающее устройство должно “знать”, что регистр 0 – bool, регистр 1 – int16 и т.д. Таким образом, добавление нового параметра потребует перепрошивки принимающего устройства (в кастомном варианте пакет самодокументируемый – перепрошивка может и не понадобиться).

Во-вторых, количество регистров, используемых в одной передаче пакета ограничено. Это означает, что мы не сможем за один раз передать, например, слишком длинную строку (длиннее 250 символов) или просто большое количество данных (например, объемную телеметрию). В кастомном варианте пакеты формально не ограничены в размерах т.к. представляют из себя непрерывный поток данных, разделяемый стартовым маркером пакета (Start Marker).

### 3.4.2. Код программы

В файлах проекта. (если нужно – перенесу сюда)

### 3.4.3 Демонстрация

…

# 4. Приложения

## 4.1. Техническое описание модуля микроконтроллера ESP8266

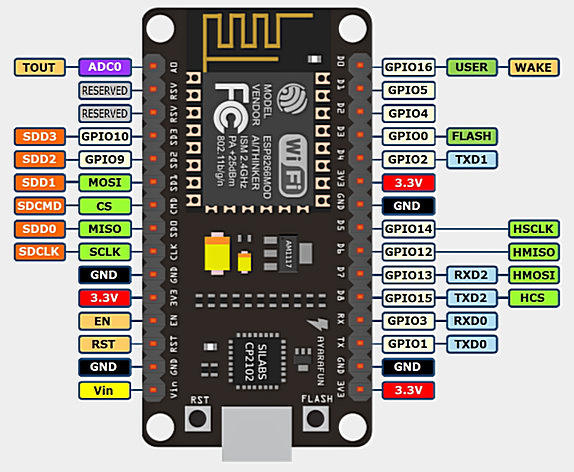
**Технические характеристики:**

* Микроконтроллер: ESP8266;
* Тактовая частота: 80 МГц (до 160 МГц);
* Конвертер USB-UART: чип ft232;
* Wi-Fi 802.11 b/g/n 2,4 ГГц;
* Загрузчик: NodeMCU;
* ОЗУ (RAM): 160 КБ, доступно до 50 КБ;
* ПЗУ (ROM): 4 МБ;
* Портов ввода-вывода: 11;
* Рабочее напряжение: 3,3 В;
* Входное напряжение: 5-12 В;
* Потребляемый ток: до 220 мА;
* Размер: 58 x 31 x 13 мм;
* Вес: 11 г.

Плата содержит 11 портов ввода-вывода общего назначения, которые дополнительно могут использоваться в качестве ШИМ, АЦП, UART, SPI, I²C/TWI, SDIO и разъем TYPE-C, кнопка сброса и кнопка отладки.

Устройство может получать питание непосредственно от USB, либо от вывода Vin, контроллер автоматически определяет и выбирает наиболее подходящий источник тока.

Модуль совместим с макетными и монтажными платами с шагом 2,54 мм.



Документация по ESP8266 на английском: <https://arduino-esp8266.readthedocs.io/en/latest/>

# 5. Источники

## 5.1. Простым языком об HTTP

* Ссылка: <https://habr.com/ru/articles/215117/>
* Дата обращения: 21.11.24
* Тип: открытый
* Язык: русский

## 5.2. Веб-фреймворк Django (Python)

* Ссылка: <https://developer.mozilla.org/ru/docs/Learn/Server-side/Django>
* Дата обращения: 21.11.24
* Тип: открытый
* Язык: русский

## 5.3. Установка ESP8266 в Arduino IDE (руководство для ОС Windows)

* Ссылка: <https://wiki.iarduino.ru/page/wemos_start/>
* Дата обращения: 22.11.24
* Тип: открытый
* Язык: русский