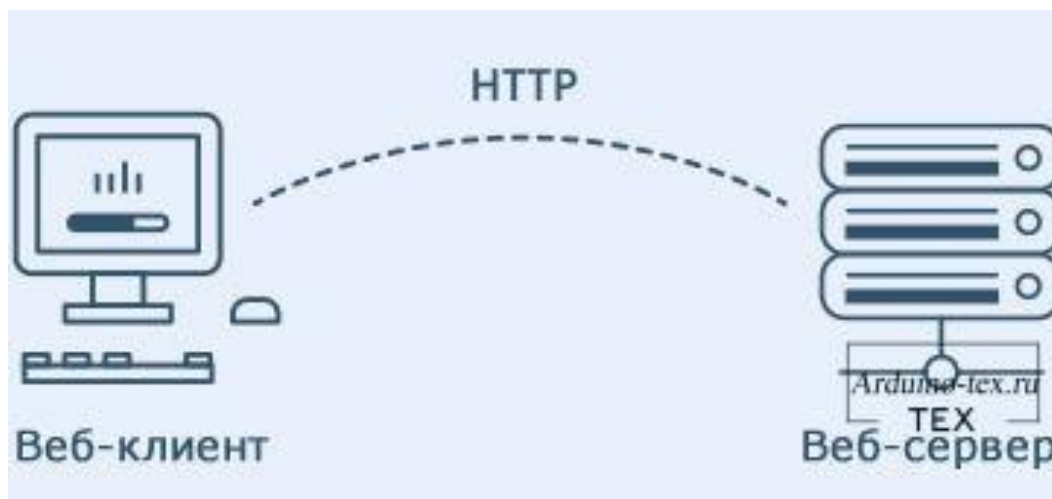


# Проект 2.1

## Веб-сервер ESP32 (ESP8266)

### Теоретическая часть:

Веб-сервер - это место, где хранятся, обрабатываются и отсылаются веб-страницы веб-клиентам. Веб-клиент - это не что иное, как веб-браузер на наших ноутбуках и смартфонах. Связь между клиентом и сервером происходит с использованием специального протокола, называемого протоколом передачи гипертекста (HTTP).



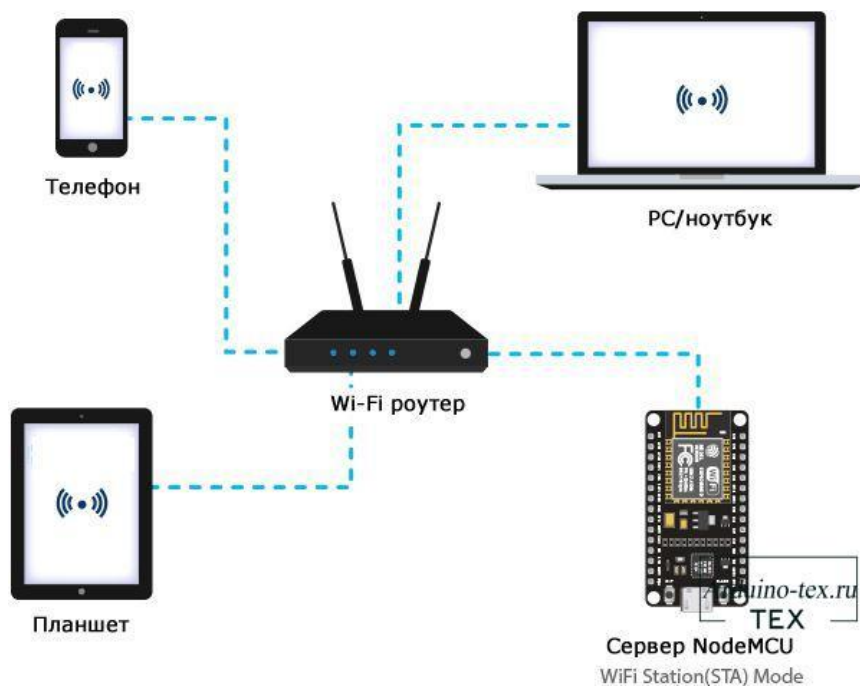
В этом протоколе клиент инициирует связь, отправляя запрос на конкретную веб-страницу с помощью HTTP, а сервер отдает содержимое этой веб-страницы, или сообщение об ошибке, если не может этого сделать (например: страница не найдена, ошибка 404). Страницы, которые отдает сервер, в основном представляют собой HTML-документы.

Режимы работы ESP32.

Одна из особенностей ESP32 заключается в том, что он может не только подключаться к существующей сети Wi-Fi и действовать как веб-сервер, но также может настраивать собственную сеть, позволяя другим устройствам напрямую подключаться к ней и получать доступ к веб-страницам. Это возможно, потому что ESP32 работает в трех разных режимах: режим станции (STA), режим точки доступа (AP) и оба режима одновременно.

Режим станции (STA).

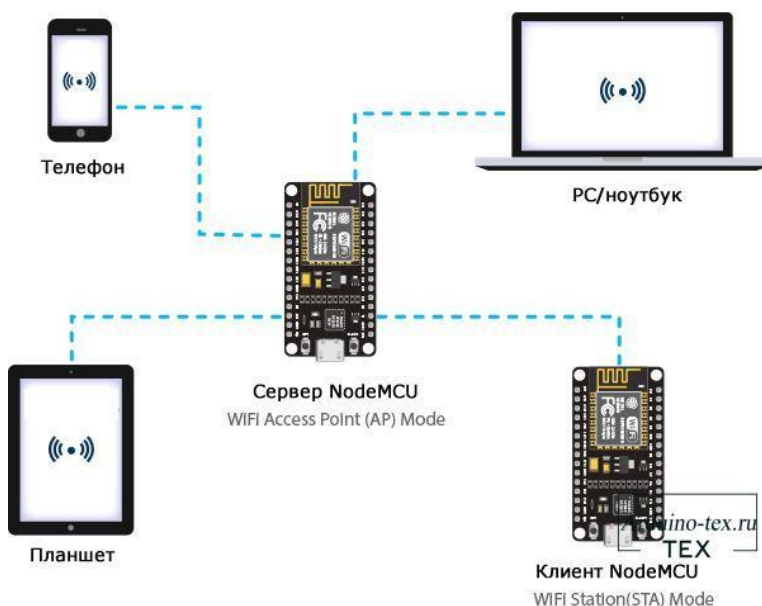
ESP32 (ESP8266), который подключается к существующей сети Wi-Fi (созданной вашим беспроводным маршрутизатором), называется станцией (STA).



В режиме STA ESP32 получает IP от маршрутизатора, к которому он подключен. С этим IP-адресом он может настроить веб-сервер и отдавать веб-страницы на все подключенные устройства в существующей сети Wi-Fi .

Режим точки доступа (AP).

ESP32 (ESP8266), который создает свою собственную сеть Wi-Fi и действует как концентратор (как Wi-Fi маршрутизатор) для одного или нескольких устройств, называется точкой доступа (AP). В отличие от Wi-Fi роутера, у него нет интерфейса для проводной сети. Итак, такой режим работы называется Soft Access Point (soft-AP). Также максимальное количество устройств, которые могут подключиться к нему, ограничено пятью.



Как происходит взаимодействие между клиентом и веб-сервером?

Клиент, которым обычно является веб-браузер, передает веб-серверу запросы на получение ресурсов, обозначенных URL-адресами. Ресурсы — это HTML-

страницы, изображения, файлы, медиа-потoki или другие данные, которые необходимы клиенту. В ответ веб-сервер передаёт клиенту запрошенную информацию.

Более детально этот процесс можно описать следующим образом:

1. Браузер делит запрошенный URL на три части: протокол, доменное имя, название файла. Например, URL: `https:// www.site.com/ file.html` будет разделен на следующие части:
  - Протокол («https»).
  - Доменное имя («www.site.com»).
  - Имя файла («file.html»).
2. Браузер при помощи сервисов DNS преобразует доменное имя в IP адрес, который используется для связи с серверной машиной.
3. Используя полученный IP адрес, браузер связывается с нужным веб-сервером и посылает на него запрос, требующий отправки файла «www.site.com / file.html».
4. Веб-сервер получает запрос и ищет указанную в нём HTML-страницу. Если страница существует, сервер отправляет браузеру ответ, в котором будет HTML-код веб-страницы и заголовок с информацией о запрошенном содержимом. Он предназначен для браузера и нужен, чтобы правильно определить размер, язык переданного файла и другие параметры. Пользователь не видит эту информацию. В заголовке передается и код ответа сервера – например, 200 ОК, когда страница найдена, или сообщение об ошибке 404, если сервер не может найти запрошенную страницу. Подробнее о кодах ответа читайте ниже - в разделе «Коды состояния http».
5. Браузер читает полученную от сервера HTML-страницу и подгружает другие необходимые ресурсы, которые с ней связаны - изображения, файлы стилей CSS и JavaScript, отвечающие за оформление и функционал страницы, и т. д.
6. После того, как браузер закончит загрузку всех этих ресурсов, веб-страница будет отображена на экране монитора.

## Что такое HTTP-взаимодействие?

HTTP (HyperText Transfer Protocol, то есть «протокол передачи гипертекста») — это протокол, предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам). Правила HTTP лежат в основе передачи информации в сети Интернет.

Задача, которая традиционно решается с помощью протокола HTTP — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером.

Клиентское приложение формирует HTTP-запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует HTTP-ответ и передаёт его обратно клиенту, как правило, вместе с HTML-страницей или другими запрошенными данными.

Каждое HTTP-сообщение состоит из трех частей:

1. Стартовая строка (Starting line) — включает служебные данные, определяет тип сообщения.
2. Заголовки (Headers) — описывают клиента и сервер, содержимое сообщения, параметры передачи и др.

Все заголовки HTTP разделяются на четыре основных группы:

- General Headers («Основные заголовки») — могут включаться в любое сообщение клиента и сервера. Например, заголовок Date используется для хранения даты и времени запроса/ответа.
- Request Headers («Заголовки запроса») — используются только в запросах клиента. Например, заголовки Host, Referer и User-Agent.
- Response Headers («Заголовки ответа») — предназначены только для ответов от сервера. Например, заголовок Server указывает название и версию веб-сервера.
- Entity Headers («Заголовки сущности») — могут находиться как в запросах, так и в ответах. Передают мета-информацию об объекте, отправленном в теле сообщения. Например, все заголовки с префиксом Content- предоставляют информацию о структуре, кодировке и размере тела сообщения. Заголовок Last-Modified содержит время и дату последнего изменения документа.

Больше узнать о заголовках можно в спецификации протокола HTTP.

3. Тело сообщения (Message Body) — содержит данные, запрошенные с сайта.

Стартовая строка и заголовок являются обязательными элементами, а тело сообщения может отсутствовать.

## Структура HTTP-запроса от клиента

Если клиент хочет загрузить, например, страницу «About», HTTP-запрос может включать следующие элементы:

GET /about.html HTTP/1.1

Host: www.site.com

Referer: https://www.google.com/

User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64)

Accept-Language: en-us

Где:

GET /about.html HTTP/1.1 Стартовая строка, в которой содержатся:

GET - метод запроса, указывающий на основную операцию, которую необходимо осуществить над ресурсом;

/ about.html — запрашиваемый документ;

HTTP — протокол передачи данных;

/1.1 — версия протокола 1.1.

Host: [www.site.com](http://www.site.com) Заголовок Host содержит имя сайта, с которого нужно загрузить документ.

Referer: <https://www.google.com/> В заголовке Referer прописан URL страницы, откуда сделан запрос.

User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) Заголовок User-Agent включает:

Mozilla/5.0 – название и версию браузера, отправившего запрос;

Windows NT 6.3; Win64; x64 – название и версию операционной системы.

Accept-Language: en-us Заголовок Accept-Language указывает язык, используемый в браузере.

## Структура HTTP-ответа веб-сервера

- Стартовая строка в HTTP-ответе указывает версию протокола передачи данных (HTTP/1.1), код статуса ответа (трехзначное число - например, 200), описание статуса (короткое пояснение к коду ответа – например, OK).
- Заголовки – предоставляют информацию об ответе, сервере или об объекте, отправленном в теле сообщения. Например, в заголовках Content-Type и Content-Length будут указаны тип и размер контента (в байтах).
- Тело ответа – при запросе содержимого веб-ресурса в нём содержится HTML-код запрошенной страницы или выгружаемые файлы. Отделяется от заголовков пустой строкой.

Пример HTTP-ответа:

HTTP/1.1 200 OK

Server: nginx/1.4.6 (Ubuntu)

Date: Mon, 25 Jan 2021 16:54:33 GMT

Content-Type: text/html; charset=UTF-8

Content-Length: 98

Last-Modified: Mon, 25 Jan 2021 16:22:21 GMT

<html>

<head>

<title>An Example Page</title>

</head>

<body>

<p>Hello World</p>

</body>

</html>

Где:

HTTP/1.1 200 OK   Стартовая строка, в которой прописаны:

HTTP – протокол передачи данных;

/1.1 – версия протокола 1.1;

200 – код ответа сервера (страница доступна);

OK - пояснение к коду ответа.

Server: nginx/1.4.6 (Ubuntu)   Заголовок Server указывает название и версию сервера, который ответил - nginx/1.4.6 (Ubuntu).

Date: Mon, 25 Jan 2021 16:54:33 GMT   Заголовок Date используется для указания даты и времени ответа.

Content-Type: text/html; charset=UTF-8   В заголовке Content-Type прописаны тип отправленного контента (text/html) и его кодировка (charset=UTF-8).

Content-Length: 98   Заголовок Content-Length указывает размер контента (в байтах) – 98.

Last-Modified: Mon, 25 Jan 2021 16:22:21 GMT   Заголовок Last-Modified содержит дату и время последнего изменения документа.

<html>

  <head>

    <title>An Example Page</title>

  </head>

  <body>

    <p>Hello World</p>

  </body>

</html>

Тело ответа, в котором содержится HTML-код запрошенной страницы.

## Коды состояния HTTP

Трехзначный код, возвращаемый сервером в стартовой строке ответа, называется кодом состояния HTTP. Он определяет результат совершения запроса. Коды состояния HTTP разработаны в соответствии со стандартами, определенными Инженерным советом Интернета (IETF).

Первая цифра кода указывает на класс состояния. В настоящее время выделено пять классов состояния:

- 1xx — Информационное сообщение (информирует о процессе передачи).
- 2xx — Сообщение об успехе (запрос получен и обработан).
- 3xx — Сообщение о перенаправлении (запрашиваемый ресурс был перемещен на другой адрес).
- 4xx — Сообщения об ошибках со стороны клиента (запрос содержит ошибки или не отвечает протоколу).
- 5xx — Сообщения об ошибках, относящихся к серверу (сервер не смог обработать запрос, хотя тот был составлен правильно).

Вторая и третья цифры в коде детализируют статус ответа. Например:

- 200 OK — запрос получен и успешно обработан.
- 201 Created — запрос получен и успешно обработан, в результате чего создан новый ресурс или его экземпляр.
- 301 Moved Permanently — запрашиваемый ресурс был перемещен навсегда, и последующие запросы к нему должны происходить по новому адресу. Адрес, по которому клиенту следует произвести запрос, сервер указывает в заголовке Location.
- 302 Moved Temporarily - ресурс перемещен временно.
- 401 Unauthorized (Неавторизованный запрос) - для доступа к документу необходимо вводить пароль или быть зарегистрированным пользователем.
- 404 Not Found — сервер не нашел ресурс по этому адресу.
- 500 Internal Server Error - сервер столкнулся с непредвиденным условием, которое не позволяет ему выполнить запрос.
- 503 Service Unavailable - сервис недоступен из-за временной перегрузки или отключения на техническое обслуживание.

Набор кодов состояния описан в соответствующих документах RFC («Request for Comments»), содержащих технические спецификации и стандарты, установленные IETF.

Вы можете проверить код статуса ответа страниц вашего сайта на сервисе Labrika в таблице «Свойства страниц сайта», расположенной в разделе «Инструменты».

Свойства страниц сайта на 11.01.2022

Все страницы	Ошибки 4xx	Ошибки 5xx	Редиректы	H1 = TITLE	Медленные	robots.txt
Недавний поиск	Поиск или фильтрация результатов...					
Тех. информация	URL	Конечный URL	rel="canonical"	Глубина	Код ответа	
Показать	https://example.com/	https://example.com/		1	200	
Показать	https://example.com/seo-audit	https://example.com/seo-audit		2	200	
Показать	https://example.com/tech-audit	https://example.com/tech-audit		2	200	

## Что такое куки (cookie)?

Куки или cookie (от англ. «печенье») — это небольшой текстовый файл с набором служебных данных о посещении сайтов, который веб-сервер отправляет для сохранения в браузере пользователя.

Браузер всякий раз при попытке открыть страницу сайта пересылает cookie обратно веб-серверу.

Примеры использования cookie:

- Автоматическая авторизация - с помощью cookie будут «запоминаться» логин и пароль.
- Для сохранения информации о товарах, добавленных в корзину.
- Для персонализации страниц с помощью cookie сохраняются личная информация и персональные настройки пользователя.
- В инструментах веб-аналитики - для отслеживания действий пользователей и сбора статистики.

От сервера браузеру cookie передаются в составе HTTP-ответа с помощью заголовка Set-Cookie:

HTTP/1.1 200 OK

Content-type: text/html

Set-Cookie: имя=значение

Браузер отправляет их серверу в HTTP-запросе в заголовке Cookie:

GET /spec.html HTTP/1.1

Host: www.example.org

Cookie: имя=значение

Например, cookie могут содержать пароль, если пользователь авторизован на данном сайте:

GET /cart HTTP/1.1

Host: www.site.com

Referer: https://www.google.com/

User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64)

Accept-Language: en-us

Cookie: login=joe@example.com; password=a497f4d4h7b956

Параметры HTTP-заголовков Cookie и Set-Cookie определяются в спецификации RFC 6265.



Без cookie каждый просмотр веб-страницы является изолированным действием, не связанным с другими просмотрами страниц того же сайта. Cookie позволяют выявить связь между просмотрами веб-страниц одним и тем же пользователем.

Аутентификация пользователей (то есть проверка их подлинности) с помощью cookie происходит следующим образом:

1. Пользователь вводит логин и пароль в соответствующих полях на странице входа и отправляет их на сервер.
2. Сервер получает имя пользователя и пароль, проверяет их и, если они верные, отправляет страницу успешного входа, прикрепив cookie с неким идентификатором сессии. Эти cookie могут быть действительны не только для текущей сессии браузера, но и быть настроены на длительное хранение.
3. Каждый раз, когда пользователь запрашивает страницу с сервера, браузер автоматически отправляет cookie с идентификатором сессии серверу. Сервер проверяет идентификатор по своей базе и, при наличии в ней такого идентификатора, «узнаёт» пользователя.

## HTTP и HTTPS

При передаче по протоколу HTTP данные не защищаются и передаются в открытом виде.

Расширение HTTPS (аббр. от англ. HyperText Transfer Protocol Secure) создано для того, чтобы соединение было безопасным, а данные передавались в зашифрованном виде по криптографическому протоколу SSL (secure sockets layer) или TLS (transport layer security). Это специальная «обертка» поверх HTTP, которая шифрует данные, делая их недоступными для злоумышленников.

Чтобы сайт стал работать по протоколу безопасного соединения HTTPS, нужен SSL-сертификат, который содержит криптографические ключи, а также данные о веб-сайте и подтверждает его подлинность. Надежные сертификаты выдаются специальными удостоверяющими центрами.

Схема передачи данных по протоколу HTTPS

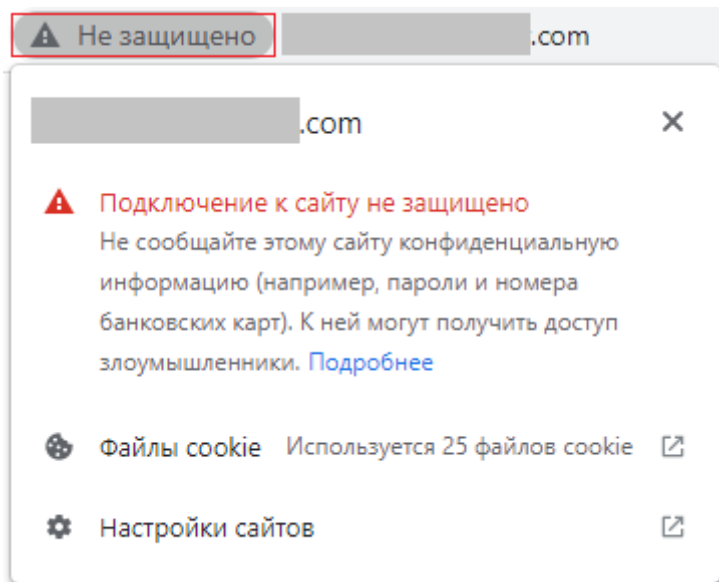
1. Браузер пользователя просит сайт предоставить SSL-сертификат для подтверждения подлинности веб-ресурса.
2. Сайт, поддерживающий HTTPS, отправляет сертификат.
3. Браузер проверяет легальность сертификата в центре сертификации.
4. Браузер и сайт договариваются о секретном ключе для шифрования данных. Они делают это при помощи асимметричного шифрования, которое предполагает использование двух ключей — открытого (для шифровки отправляемого сообщения) и закрытого (для расшифровки полученного сообщения).
5. Браузер и сайт передают информацию, используя симметричное шифрование, при котором данные зашифровываются и расшифровываются с помощью одного и того же общего ключа.
6. Синтаксически HTTPS идентичен протоколу HTTP, то есть использует те же стартовые строки и заголовки.

## Значение HTTPS для SEO

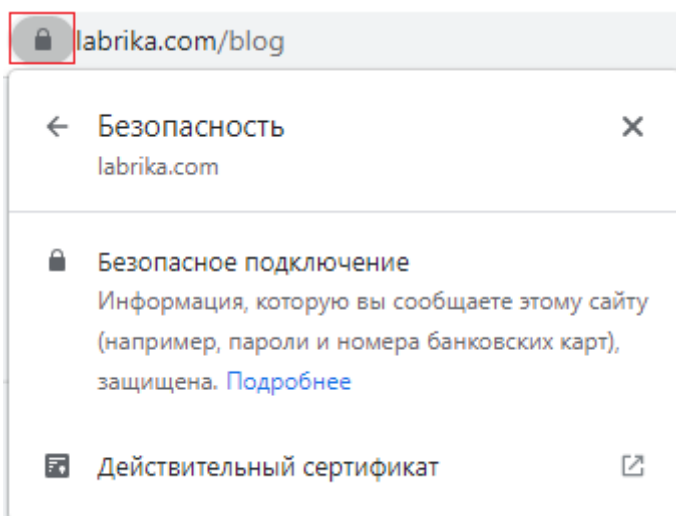
Использование протокола HTTPS является значимым фактором ранжирования сайтов в результатах поиска. Ещё в 2015 году специалист Google Гэри Иллис заявил, что повышение рейтинга компании по протоколу HTTPS может стать решающим фактором, когда в остальном сигналы качества для двух результатов поиска равны. Если поисковый робот обнаружит две одинаковые страницы - одну с HTTP, а вторую с HTTPS, он добавит в индекс страницу с безопасным протоколом.

Сайт, который не использует HTTPS-протокол, со временем может потерять свои позиции в поиске. Отсутствие HTTPS говорит пользователям и поисковым системам, что безопасность их данных находится под угрозой, а значит, отображать такой сайт на первых страницах выдачи и переходить на него не стоит.

Кроме того, в строке браузера, например, Google Chrome, рядом с адресом сайта без HTTPS ставится отметка «Не защищено».



Тогда как сайты, использующие безопасный протокол HTTPS, помечаются значком в виде замка.



Посмотреть данные о наличии на вашем сайте протокола HTTPS и SSL-сертификата, позволяющего использовать зашифрованное соединение, вы можете на сервисе Labrika в отчете «Безопасность» раздела «Технический аудит».

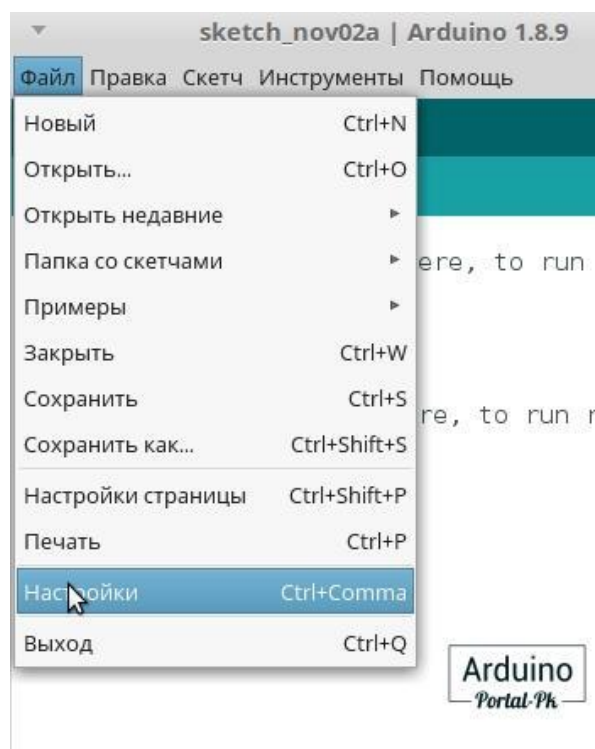
Безопасность на 18.11.2021			
Наличие HTTPS	Есть		Описание
Дата завершения SSL	05.08.2022		Описание
Самоподписанный сертификат	Нет		Описание
Сертификат для этого домена	Да		Описание
Подтвержден центром регистрации	Да		Описание
Есть редирект с HTTP на HTTPS	Да		Описание
В URL присутствует порт :443	Нет		Описание
IP занесён в черные списки RBL/DNSB	Нет		Описание
Страницы с тегом frame/iframe	0	Показать	Описание

Протокол HTTPS обязательно должен быть на тех сайтах, которые запрашивают конфиденциальные данные пользователей: логины и пароли от аккаунтов, номера банковских карт, адреса электронной почты и т. п. В первую очередь это касается коммерческих ресурсов, социальных сетей, почтовых сервисов, сайтов с регистрацией пользователей.

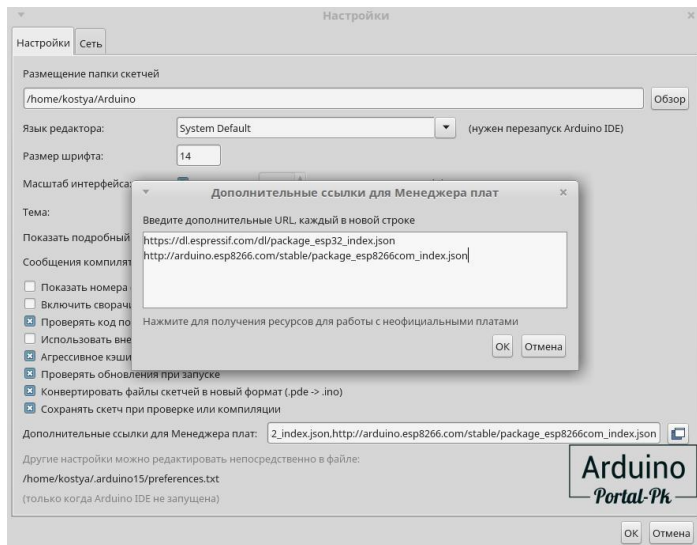
## Практическая часть:

### Установка дополнения ESP32 в Arduino IDE

Чтобы установить плату **ESP32** в **Arduino IDE**, выполните следующие действия:  
1. В вашем **Arduino IDE** перейдите в **Файл > Настройки**



2. Войдите [https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json). в поле "Дополнительные ссылки для менеджера платы", как показано на рисунке ниже. Затем нажмите кнопку ОК.



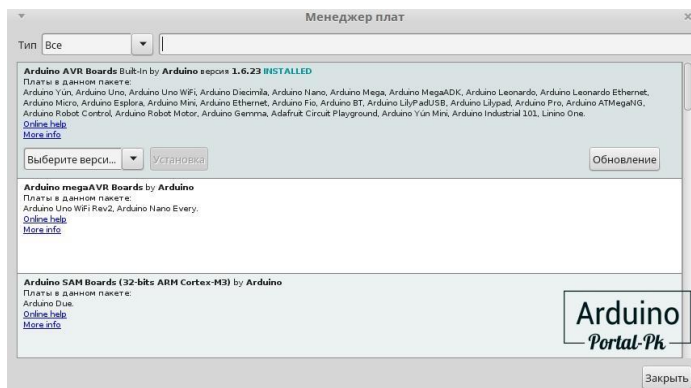
**Внимание:** Если вы у вас установлена в **Arduino IDE ESP8266** то добавьте 2 строки или через запятую:

[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json)

[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)

Сору

3. Откройте Менеджер плат. Перейдите в меню **Инструменты > Платы > Менеджер плат**



4. Ищем **ESP32**. И нажимаем кнопку **Установка**.

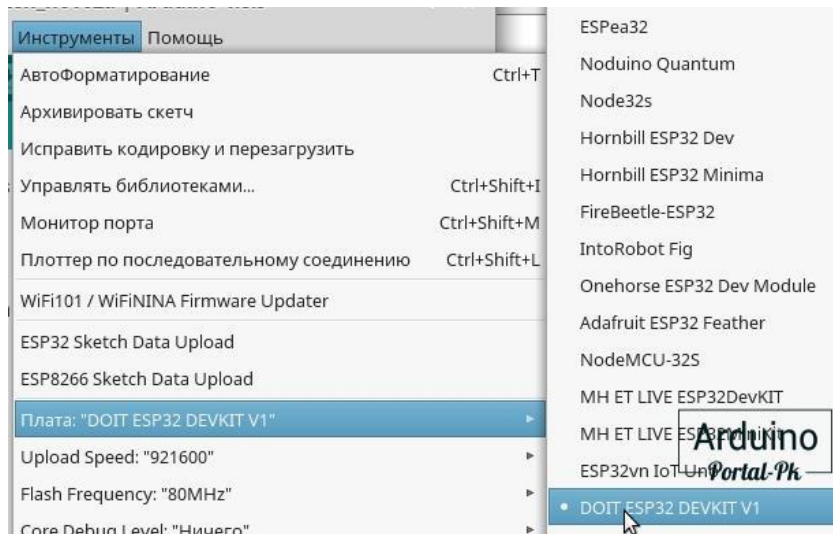


5. Вот и все. Через пару минут у вас все установиться.

**Первая загрузка скетча в ESP32.**

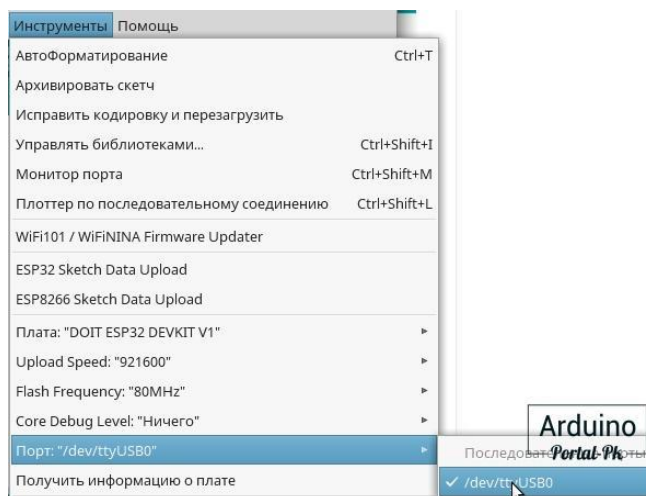
Подключите плату ESP32 к компьютеру. Открываем Arduino IDE и выполним следующие действия:

1. Выберите доску в меню **Инструменты > Плата** (в моем случае это **DOIT ESP32 DEVKIT V1**)

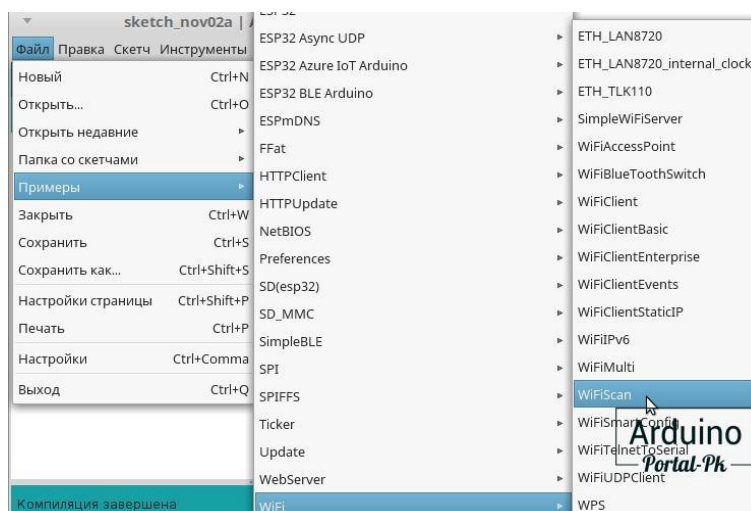


Плата (в моем случае это DOIT ESP32 DEVKIT V1)" width="382" height="240" style="display: block; margin: auto; width: 382px; height: 240px;" rel="display: block; margin: auto; width: 382px; height: 240px;">

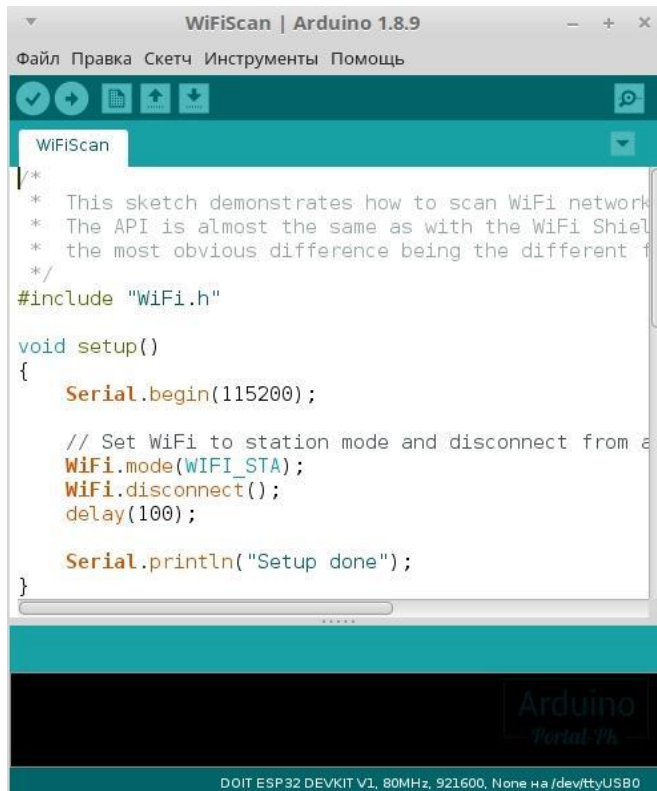
2. Выберите порт (если вы не видите COM-порт в Arduino IDE, вам необходимо установить драйверы CP210x USB to UART Bridge VCP). В моем случае ком порт отображается по другому. Это связано с тем что я использую OS Linux.



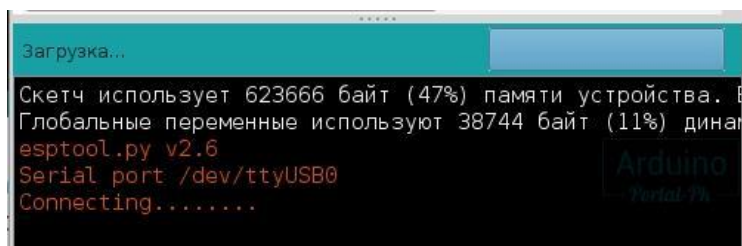
3. Открыть в следующем примере в **Файл > Примеры > WiFi > WiFiScan**



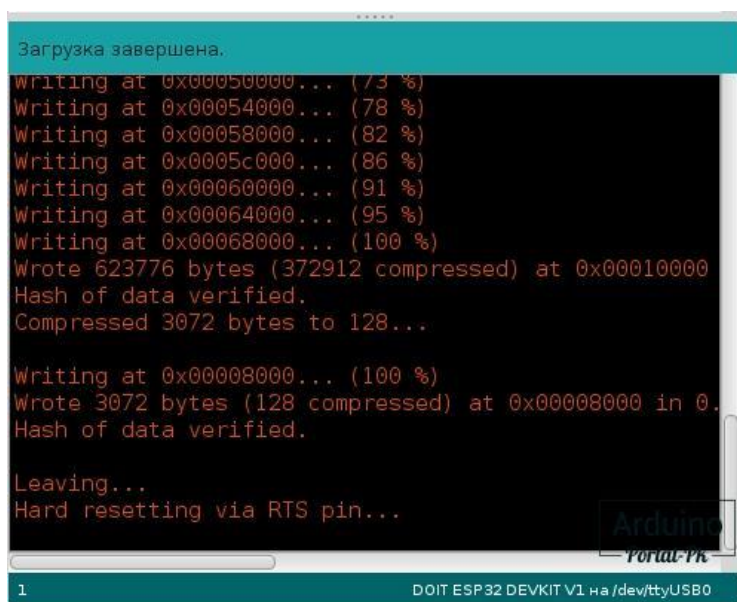
4. В вашей среде Arduino IDE откроется новый скетч.



5. Нажмите кнопку **Загрузить** в **Arduino IDE**. Подождите несколько секунд, пока код компилируется и загружается на вашу плату.



**Внимание!** Если у вас бежит строка с повторяющимися точками и линиями. Нажмите кнопку **Boot** на плате и удерживайте ее пока не побегут проценты загрузки. После чего кнопку можно отпустить.





6. Откройте последовательный **монитор Arduino IDE** со скоростью передачи данных **115200**.

```

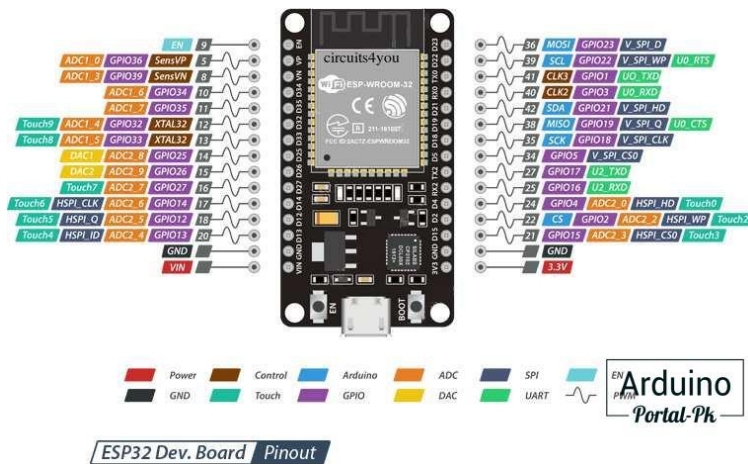
/dev/ttyUSB0

7 networks found
1: Wi-Fi (-65)*
2: kv67 (-67)*
3: Polina (-74)*
4: kv48 (-75)*
5: kv-45 (-81)*
6: TP-LINK_50 (-90)*
7: TP-LINK_A71228 (-92)*

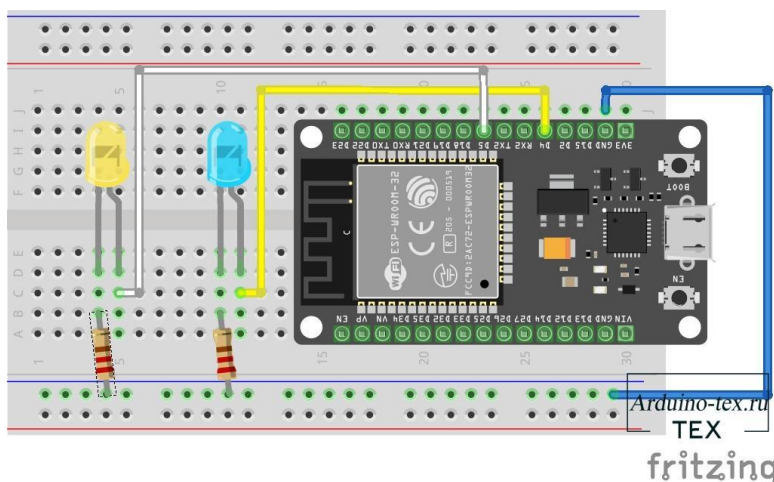
scan start
scan done
7 networks found
1: Wi-Fi (-65)*
2: kv67 (-67)*
3: kv48 (-76)*
4: Polina (-78)*
5: kv-45 (-82)*
6: TP-LINK_50 (-92)*
7: TP-LINK_A71228 (-93)*

```

7. Нажмите кнопку **EN** на борту **ESP32**, и вы увидите список сетей, доступных для вашей ESP32.



**ESP32 в качестве HTTP-сервера с использованием режима точки доступа WiFi (AP).**



Начните с размещения **ESP32** на макетной плате. Затем подключите два светодиода к цифровым **GPIO 4 и 5** через ограничительный резистор **220 Ом**.

И соответствующий код для нашей схемы:

```
#include <WiFi.h>
#include <WebServer.h>
/* Установите здесь свои SSID и пароль */
const char* ssid = "ESP32";
const char* password = "01234567";
/* Настройки IP адреса */
IPAddress local_ip(192,168,2,1);
IPAddress gateway(192,168,2,1);
IPAddress subnet(255,255,255,0);
WebServer server(80);
uint8_t LED1pin = 4;
bool LED1status = LOW;
uint8_t LED2pin = 5;
bool LED2status = LOW;
void setup() {
    Serial.begin(115200);
    pinMode(LED1pin, OUTPUT);
    pinMode(LED2pin, OUTPUT);
    WiFi.softAP(ssid, password);
    WiFi.softAPConfig(local_ip, gateway, subnet);
    delay(100);
    server.on("/", handle_OnConnect);
    server.on("/led1on", handle_led1on);
    server.on("/led1off", handle_led1off);
    server.on("/led2on", handle_led2on);
    server.on("/led2off", handle_led2off);
    server.onNotFound(handle_NotFound);
    server.begin();
    Serial.println("HTTP server started");
}
void loop() {
    server.handleClient();
    if(LED1status)
    {digitalWrite(LED1pin, HIGH);}
    else
    {digitalWrite(LED1pin, LOW);}
    if(LED2status)
    {digitalWrite(LED2pin, HIGH);}
    else
    {digitalWrite(LED2pin, LOW);}
}
void handle_OnConnect() {
    LED1status = LOW;
    LED2status = LOW;
    Serial.println("GPIO4 Status: OFF | GPIO5 Status: OFF");
    server.send(200, "text/html",
    SendHTML(LED1status,LED2status));
}
void handle_led1on() {
```



```

    LED1status = HIGH;
    Serial.println("GPIO4 Status: ON");
    server.send(200, "text/html", SendHTML(true,LED2status));
}
void handle_led1off() {
    LED1status = LOW;
    Serial.println("GPIO4 Status: OFF");
    server.send(200, "text/html", SendHTML(false,LED2status));
}
void handle_led2on() {
    LED2status = HIGH;
    Serial.println("GPIO5 Status: ON");
    server.send(200, "text/html", SendHTML(LED1status,true));
}
void handle_led2off() {
    LED2status = LOW;
    Serial.println("GPIO5 Status: OFF");
    server.send(200, "text/html", SendHTML(LED1status,false));
}
void handle_NotFound(){
    server.send(404, "text/plain", "Not found");
}
String SendHTML(uint8_t led1stat,uint8_t led2stat){
    String ptr = "<!DOCTYPE html> <html>\n";
    ptr += "<meta http-equiv=\"Content-type\" content=\"text/html; charset=utf-8\"><head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";
    ptr += "<title>Управление светодиодами</title>\n";
    ptr += "<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;} \n";
    ptr += "body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;} \n";
    ptr += ".button {display: block;width: 80px;background-color: #3498db;border: none;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;} \n";
    ptr += ".button-on {background-color: #3498db;} \n";
    ptr += ".button-on:active {background-color: #2980b9;} \n";
    ptr += ".button-off {background-color: #34495e;} \n";
    ptr += ".button-off:active {background-color: #2c3e50;} \n";
    ptr += "p {font-size: 14px;color: #888;margin-bottom: 10px;} \n";
    ptr += "</style>\n";
    ptr += "</head>\n";
    ptr += "<body>\n";
    ptr += "<h1>ESP32 Веб сервер</h1>\n";
    ptr += "<h3>Режим точка доступа WiFi (AP)</h3>\n";
    if(led1stat)
        {ptr += "<p>Состояние LED1: ВКЛ.</p><a class=\"button button-off\" href=\"/led1off\">ВЫКЛ.</a>\n";}
    else

```

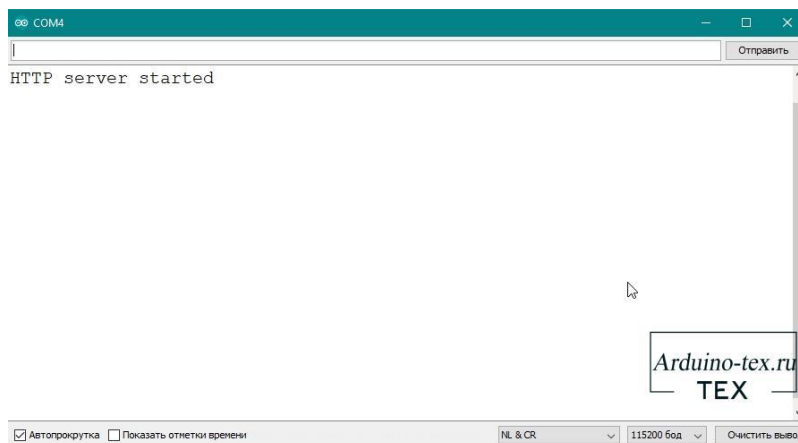
```

    {ptr += "<p>Состояние LED1: ВЫКЛ.</p><a class=\"button button-on\" href=\"/led1on\">ВКЛ.</a>\n";}
    if(led2stat)
    {ptr += "<p>Состояние LED2: ВКЛ.</p><a class=\"button button-off\" href=\"/led2off\">ВЫКЛ.</a>\n";}
    else
    {ptr += "<p>Состояние LED2: ВЫКЛ.</p><a class=\"button button-on\" href=\"/led2on\">ВКЛ.</a>\n";}
    ptr += "</body>\n";
    ptr += "</html>\n";
    return ptr;
}

```

## Доступ к веб-серверу в режиме AP.

После загрузки скетча откройте *Serial Monitor* со скоростью 115200 бод. И нажмите кнопку **RESET** на **ESP32**. Если все в порядке, то отобразится сообщение о запуске **HTTP-сервера**.



Затем найдите любое устройство, которое можно подключить к сети **Wi-Fi - телефон, ноутбук** и т. д. И найдите сеть под названием «**ESP32**». Подключитесь к сети с паролем «01234567».

После подключения к **сети ESP32 AP**, загрузите браузер и укажите адрес 192.168.1.1. **ESP32** должен открыть веб-страницу, показывающую текущее состояние светодиодов, и две кнопки для управления ими.



## ESP32 Веб сервер

Режим точка доступа WiFi (AP)

Состояние LED1: ВЫКЛ.

ВКЛ.

Состояние LED2: ВЫКЛ.

ВКЛ.

Arduino-tex.ru  
TEX

Если одновременно взглянуть на монитор последовательного порта, то можно увидеть состояние контактов GPIO ESP32.

Теперь нажмите кнопку, чтобы включить LED1, следя за URL-адресом. После того, как вы нажали кнопку, ESP32 получит запрос URL-адреса /led1on . Затем он включит LED1 и обновит веб-страницу с новым статусом LED.



## ESP32 Веб сервер

Режим точка доступа WiFi (AP)

Состояние LED1: Вкл.

ВЫКЛ.

Состояние LED2: Выкл.

ВКЛ.

Arduino-tex.ru  
— TEX —

Он также покажет состояние вывода GPIO в последовательном мониторе порта.

Вы можете протестировать кнопку LED2 и убедиться, что она работает аналогичным образом.



## ESP32 Веб сервер

Режим точка доступа WiFi (AP)

Состояние LED1: Вкл.

ВЫКЛ.

Состояние LED2: Вкл.

ВЫКЛ.

Arduino-tex.ru  
— TEX —

Скетч начинается с подключения библиотеки WiFi.h. Эта библиотека предоставляет специальные методы WiFi для ESP32, которые мы вызываем для подключения к сети. После этого мы также подключаем библиотеку WebServer.h, в которой есть несколько доступных методов, которые помогут нам настроить сервер и обрабатывать входящие HTTP-запросы, не беспокоясь о деталях реализации на низком уровне.

```
#include <WiFi.h>

#include <WebServer.h>
```

Когда мы устанавливаем ESP32 в режим точки доступа (AP), он создает сеть Wi-Fi. Следовательно, нам нужно установить его SSID, пароль, IP-адрес, маску IP-подсети и IP-шлюз.

```
/* Установите здесь свои SSID и пароль */

const char* ssid = "ESP32"; // Enter SSID here

const char* password = "0123456"; //Enter Password here

/* Настройки IP адреса */

IPAddress local_ip(192,168,2,1);

IPAddress gateway(192,168,2,1);

IPAddress subnet(255,255,255,0);
```

Затем мы объявляем объект библиотеки WebServer, чтобы получить доступ к его функциям. Конструктор этого объекта принимает порт (который сервер будет слушать) в качестве параметра. Поскольку 80 - порт по умолчанию для HTTP, мы будем использовать это значение. Теперь вы можете получить доступ к серверу без необходимости указывать порт в URL-адресе.

```
WebServer server(80);
```

После чего мы объявляем выводы GPIO ESP32, к которым подключены светодиоды, и их начальное состояние.

```
uint8_t LED1pin = 4;

bool LED1status = LOW;

uint8_t LED2pin = 5;

bool LED2status = LOW;
```

### **Функция Setup ().**

Мы настраиваем наш HTTP-сервер перед его запуском. Прежде всего, мы открываем последовательное соединение для отладки и устанавливаем для GPIO значение OUTPUT.

```
Serial.begin(115200);

pinMode(LED1pin, OUTPUT);

pinMode(LED2pin, OUTPUT);
```

Затем мы настраиваем программную точку доступа для создания сети Wi-Fi, проверяя SSID, пароль, IP-адрес, маску IP-подсети и IP-шлюз.

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

WiFi.softAPConfig(local_ip, gateway, subnet);
```

```
delay(100);
```

Чтобы обрабатывать входящие HTTP-запросы, нам нужно указать, какой код выполнять при вызове конкретного URL. Для этого мы используем метод `on`. Этот метод принимает два параметра. Первый - это URL-путь, а второй - имя функции, которую мы хотим выполнить при переходе по этому URL-адресу.

Например, первая строка приведенного ниже фрагмента кода указывает, что, когда сервер получает HTTP-запрос по корневому ( / ) пути, он запускает `handle_OnConnect()` функцию. Обратите внимание, что указанный URL-адрес является относительным путем.

Точно так же нам нужно указать еще 4 URL-адреса для обработки двух состояний 2 светодиодов.

```
server.on("/", handle_OnConnect);  
  
server.on("/led1on", handle_led1on);  
  
server.on("/led1off", handle_led1off);  
  
server.on("/led2on", handle_led2on);  
  
server.on("/led2off", handle_led2off);
```

Мы не указали, что должен делать сервер, если клиент запрашивает любой URL-адрес, отличный от указанного в `server.on()`. Он должен ответить HTTP-статусом 404 (Not Found) и сообщением для пользователя. Мы также помещаем в функцию и используем `server.onNotFound()`, чтобы сообщить, что она должна выполнить ее, когда получит запрос на URI, который не был указан в `server.on`.

```
server.onNotFound(handle_NotFound);
```

Теперь, чтобы запустить наш сервер, мы вызываем метод `begin` объекта `server`.

```
server.begin();  
  
Serial.println("HTTP server started");
```

## Основной цикл `Loop()`.

Чтобы обрабатывать входящие HTTP-запросы, нам нужно вызвать `handleClient()` метод объекта сервера. Также мы меняем состояние светодиода по запросу.

```
void loop() {  
  
    server.handleClient();  
  
    if(LED1status)  
  
        {digitalWrite(LED1pin, HIGH);}  
  
    else  
  
        {digitalWrite(LED1pin, LOW);}  
  
    if(LED2status)
```

```
{digitalWrite(LED2pin, HIGH);}

else

{digitalWrite(LED2pin, LOW);}

}
```

Затем нам нужно создать функцию, которую мы назначили корневому (/) URL с помощью `server.on`. В начале этой функции мы устанавливаем состояние обоих светодиодов на LOW (исходное состояние светодиодов) и выводим состояние светодиодов в последовательный монитор. Чтобы ответить на HTTP-запрос, мы используем метод `send`. Хотя метод может быть вызван с другим набором аргументов, его простейшая форма состоит из кода ответа HTTP.

В нашем случае мы отправляем **код 200** (один из кодов состояния HTTP), который соответствует ответу **ОК** - ответ получен. Затем мы указываем тип содержимого как «текст / html» и, наконец, вызываем пользовательскую функцию `SendHTML()`, которая создает динамическую HTML-страницу, содержащую состояние светодиодов.

```
void handle_OnConnect() {

    LED1status = LOW;

    LED2status = LOW;

    Serial.println("GPIO4 Status: OFF | GPIO5 Status: OFF");

    server.send(200, "text/html", SendHTML(LED1status, LED2status));

}
```

Точно так же нам нужно создать четыре функции для обработки запросов включения / выключения светодиодов и страницы **ошибки 404**.

```
void handle_led1on() {

    LED1status = HIGH;

    Serial.println("GPIO4 Status: ON");

    server.send(200, "text/html", SendHTML(true, LED2status));

}

void handle_led1off() {

    LED1status = LOW;

    Serial.println("GPIO4 Status: OFF");

    server.send(200, "text/html", SendHTML(false, LED2status));

}

void handle_led2on() {

    LED2status = HIGH;
```

```

Serial.println("GPIO5 Status: ON");

server.send(200, "text/html", SendHTML(LED1status,true));
}

void handle_led2off() {

  LED2status = LOW;

  Serial.println("GPIO5 Status: OFF");

  server.send(200, "text/html", SendHTML(LED1status,false));
}

void handle_NotFound(){

  server.send(404, "text/plain", "Not found");
}

```

## Отображение веб-страницы HTML.

SendHTML() Функция отвечает за создание веб-страницы всякий раз, когда веб-сервер ESP32 получает запрос от веб-клиента. Он просто объединяет HTML-код в большую строку и возвращает server.send() функцию, которую мы обсуждали ранее. Функция принимает состояние светодиодов в качестве параметров для динамической генерации содержимого HTML.

Первый текст, который вы всегда должны отправить **<!DOCTYPE>**, который указывает на то, что мы посылаем HTML код.

```

String SendHTML(uint8_t led1stat,uint8_t led2stat){

String ptr = "<!DOCTYPE html> <html>\n";

```

Затем элемент <meta> viewport заставляет веб-страницу загружаться в любом браузере. В то время как тег title устанавливает заголовок страницы.

```

ptr += "<meta http-equiv=\"Content-type\" content=\"text/html; charset=utf-8\"><head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr += "<title>Управление светодиодом</title>\n";

```

## Стилизация веб-страницы.

Затем у нас есть CSS для стилизации кнопок и внешнего вида веб-страницы. Мы выбираем шрифт Helvetica, определяем контент, который будет отображаться, в виде встроенного блока, и выравниваться по центру.

```

ptr += "<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

```

Затем следующий код устанавливает цвет, шрифт и поля тегов H1, H3 и p.

```

ptr += "body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";

```

```
ptr += "p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";
```

Дальше прописывая стили, которые применяются к кнопкам, указав свойства: цвет, размер, поля и т. д. Кнопки ВКЛ. и ВЫКЛ. имеют разный цвет фона, в то время как :active selector для кнопок обеспечивает эффект нажатия кнопки.

```
ptr += ".button {display: block;width: 80px;background-color: #3498db;border: none;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";

ptr += ".button-on {background-color: #3498db;}\n";

ptr += ".button-on:active {background-color: #2980b9;}\n";

ptr += ".button-off {background-color: #34495e;}\n";

ptr += ".button-off:active {background-color: #2c3e50;}\n";
```

### **Установка заголовка веб-страницы.**

Далее задается заголовок веб-страницы. Вы можете изменить этот текст на любой, который подходит для вашего приложения.

```
ptr += "<h1>ESP32 Веб сервер</h1>\n";

ptr += "<h3>Режим точка доступа WiFi (AP)</h3>\n";
```

### **Отображение кнопок, и соответствующего их состояния**

Чтобы динамически генерировать кнопки и статус светодиода, мы используем оператор if. Таким образом, в зависимости от состояния контактов GPIO отображается кнопка ВКЛ / ВЫКЛ.

```
if(led1stat)

    {ptr += "<p>Состояние LED1: ВКЛ.</p><a class=\"button button-off\" href=\"/led1off\">ВЫКЛ.</a>\n";}

    else

    {ptr += "<p>Состояние LED1: ВЫКЛ.</p><a class=\"button button-on\" href=\"/led1on\">ВКЛ.</a>\n";}

if(led2stat)

    {ptr += "<p>Состояние LED2: ВКЛ.</p><a class=\"button button-off\" href=\"/led2off\">ВЫКЛ.</a>\n";}

    else

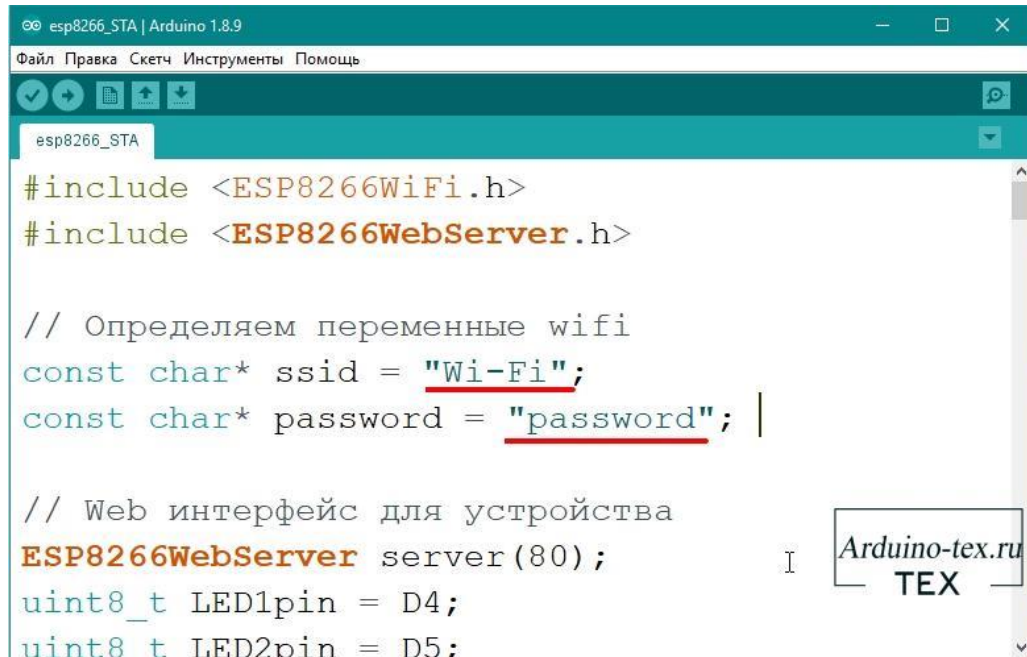
    {ptr += "<p>Состояние LED2: ВЫКЛ.</p><a class=\"button button-on\" href=\"/led2on\">ВКЛ.</a>\n";}
```

## **Использование ESP8266 (NodeMCU) в качестве HTTP сервера в режиме Wi-Fi Station (STA)**

Давайте рассмотрим пример использования ESP8266 (NodeMCU) в качестве HTTP сервера в режиме Wi-Fi Station (STA). Прежде чем приступить к загрузке



скетча, чтобы он у вас заработал, необходимо внести некоторые изменения. Чтобы ESP8266 мог установить соединение с существующей сетью, вам необходимо изменить две следующие переменные с учетными данными вашей сети.



```
esp8266_STA | Arduino 1.8.9
Файл Правка Скетч Инструменты Помощь

esp8266_STA

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

// Определяем переменные wifi
const char* ssid = "Wi-Fi";
const char* password = "password";

// Web интерфейс для устройства
ESP8266WebServer server(80);
uint8_t LED1pin = D4;
uint8_t LED2pin = D5;
```

После чего можно загрузить код в **ESP8266 (NodeMCU)**.

```
#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>

// Определяем переменные wifi
const char* ssid = "Wi-Fi";
const char* password = "password";

// Web интерфейс для устройства
ESP8266WebServer server(80);

uint8_t LED1pin = D4;
uint8_t LED2pin = D5;

bool LED1status = LOW;
bool LED2status = LOW;

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

    delay(100);

    pinMode(LED1pin, OUTPUT);
    pinMode(LED2pin, OUTPUT);
}
```

```

Serial.println("Connecting to ");

Serial.println(ssid);

//connect to your local wi-fi network

WiFi.begin(ssid, password);

//check wi-fi is connected to wi-fi network

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

  delay(1000);

  Serial.print(".");

}

Serial.println("");

Serial.println("WiFi connected..!");

Serial.print("Got IP: "); Serial.println(WiFi.localIP());

server.on("/", handle_OnConnect);

server.on("/led1on", handle_led1on);

server.on("/led1off", handle_led1off);

server.on("/led2on", handle_led2on);

server.on("/led2off", handle_led2off);

server.onNotFound(handle_NotFound);

server.begin();

Serial.println("HTTP server started");

}

void loop() {

  server.handleClient();

  if(LED1status)

  {digitalWrite(LED1pin, HIGH);}

  else

  {digitalWrite(LED1pin, LOW);}

  if(LED2status)

  {digitalWrite(LED2pin, HIGH);}

  else

  {digitalWrite(LED2pin, LOW);}

```

```

}

void handle_OnConnect() {
    LED1status = LOW;
    LED2status = LOW;
    Serial.println("GPIO4 Status: OFF | GPIO5 Status: OFF");
    server.send(200, "text/html", SendHTML(LED1status, LED2status));
}

void handle_led1on() {
    LED1status = HIGH;
    Serial.println("GPIO4 Status: ON");
    server.send(200, "text/html", SendHTML(true, LED2status));
}

void handle_led1off() {
    LED1status = LOW;
    Serial.println("GPIO4 Status: OFF");
    server.send(200, "text/html", SendHTML(false, LED2status));
}

void handle_led2on() {
    LED2status = HIGH;
    Serial.println("GPIO5 Status: ON");
    server.send(200, "text/html", SendHTML(LED1status, true));
}

void handle_led2off() {
    LED2status = LOW;
    Serial.println("GPIO5 Status: OFF");
    server.send(200, "text/html", SendHTML(LED1status, false));
}

void handle_NotFound(){
    server.send(404, "text/plain", "Not found");
}

String SendHTML(uint8_t led1stat, uint8_t led2stat){

```

```
String ptr = "<!DOCTYPE html> <html>\n";

ptr += "<meta http-equiv=\"Content-type\" content=\"text/html; charset=utf-8\"><head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

ptr += "<title>Управление светодиодом</title>\n";

ptr += "<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

ptr += "body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";

ptr += ".button {display: block;width: 80px;background-color: #3498db;border: none;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";

ptr += ".button-on {background-color: #3498db;}\n";

ptr += ".button-on:active {background-color: #2980b9;}\n";

ptr += ".button-off {background-color: #34495e;}\n";

ptr += ".button-off:active {background-color: #2c3e50;}\n";

ptr += "p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";

ptr += "</style>\n";

ptr += "</head>\n";

ptr += "<body>\n";

ptr += "<h1>ESP8266 Веб сервер</h1>\n";

ptr += "<h3>Режим станции (STA)</h3>\n";

    if(led1stat)

        {ptr += "<p>Состояние LED1: ВКЛ.</p><a class=\"button button-off\" href=\"/led1off\">ВЫКЛ.</a>\n";}

    else

        {ptr += "<p>Состояние LED1: ВЫКЛ.</p><a class=\"button button-on\" href=\"/led1on\">ВКЛ.</a>\n";}

    if(led2stat)

        {ptr += "<p>Состояние LED2: ВКЛ.</p><a class=\"button button-off\" href=\"/led2off\">ВЫКЛ.</a>\n";}

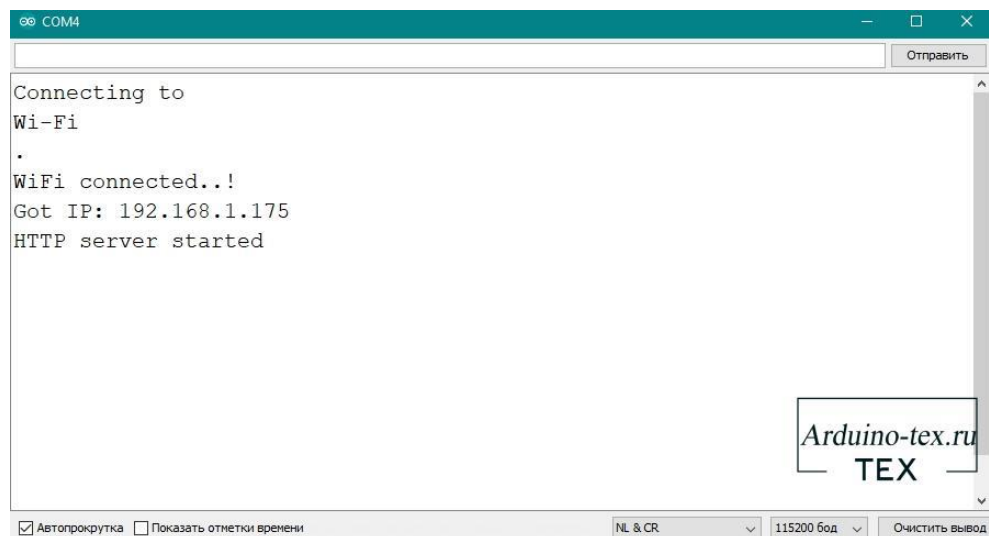
    else

        {ptr += "<p>Состояние LED2: ВЫКЛ.</p><a class=\"button button-on\" href=\"/led2on\">ВКЛ.</a>\n";}

ptr += "</body>\n";

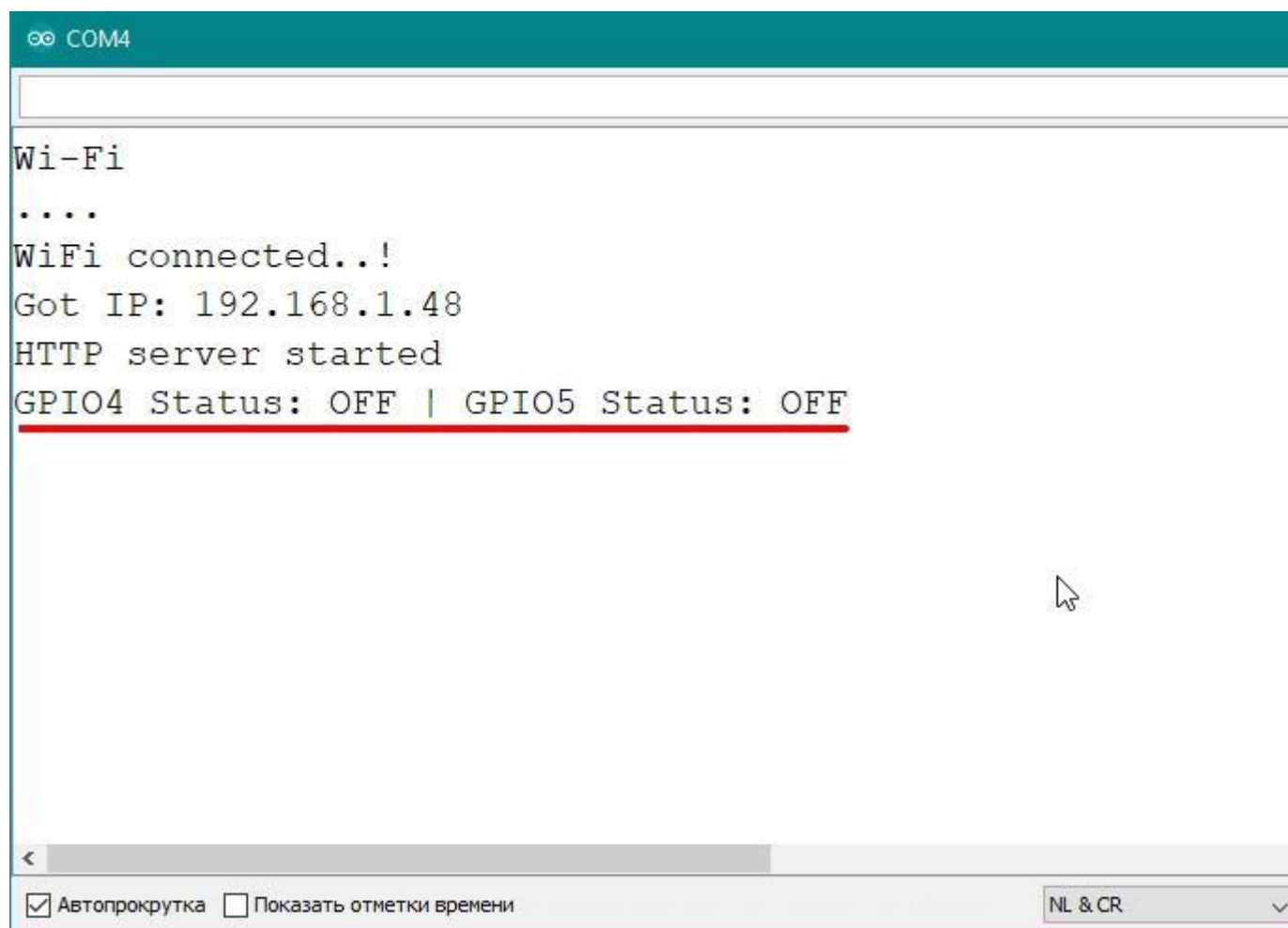
ptr += "</html>\n";
```

```
return ptr;  
}
```



Затем загрузите браузер и введите IP адрес, указанный в мониторе последовательного порта. **NodeMCU** должен выдать веб-страницу, показывающую текущее состояние светодиодов, и две кнопки для управления ими.

Если в это время взглянуть на монитор последовательного порта, то можно увидеть состояние выводов GPIO **NodeMCU**.



В работе данного примера нет ни каких изменений в сравнении с работой ESP32.

## **Изменения в скетче для ESP8266 NodeMCU.**

Первым делом нам нужно изменить библиотеки для работы с ESP8266.

```
#include <ESP8266WiFi.h>

#include <ESP8266WebServer.h>
```

Затем объявляем объект библиотеки ESP8266WebServer, чтобы получить доступ к ее функциям. Конструктор этого объекта в качестве параметра принимает номер порта (который сервер будет прослушивать).

```
// Web интерфейс для устройства

ESP8266WebServer server(80);
```

GPIO у данной платы также обозначаются по другому, их следует также изменить.

```
uint8_t LED1pin = D4;

uint8_t LED2pin = D5;
```

Осталось подправить HTML странице выводимую информацию. Здесь изменим заголовок страницы. Чтобы было понятно, что данный пример работает на **ESP8266**.

```
ptr += "<h1>ESP8266 Веб сервер</h1>\n";

ptr += "<h3>Режим станции (STA)</h3>\n";
```

## **Усложняем задачу:**

- 1) Подключение системы климат контроля из лабораторной работы 1.3 к системе веб-сервиса. То есть вам нужно сделать интерактивную панель с отображением температуры, влажности, наличия пожара и сухости растений в браузере.
- 2) Выполните стилизацию страницы: измените шрифт, добавьте цвета, расположение на странице, заголовки. Отдельным плюсом будет, если проявите креативность (закастомите страницу)
- 3) Добавить автоматическое обновление страницы для достоверного отображения информации. Например, вы можете использовать AJAX для отображения текущего статуса вашего устройства или обновления информации о сенсорных данных в реальном времени.
- 4) Добавить счетчик работы сервера. Временной таймер сколько сервер был запущен
- 5) \*\*
- 6) Добавьте на ваш клиент мультистраничность. Чтобы каждый данные каждого датчика открывались в своей вкладке, доступной с главной страницы.
- 7) Добавьте авторизацию. Чтобы увидеть данные мог только 1 пользователь, который вошел в систему.