Welcome!

Thank you for purchasing our *AZ-Delivery ArduiTouch ESP*. On the following pages, we will introduce you to how to use and set-up this handy device. We will also guide you through our *Weather App*, which can be used on this device.

Have fun!

Popular ESP8266 and ESP32 based modules are used in many IoT applications, such as smart home projects, remote control, robotics, etc. Many of ESP based projects can be found online.

ArduiTouch is a touchscreen enclosure set made for wall mounting with NodeMCU or Wemos D1 Mini boards based on ESP8266 or ESP32 platforms.

ArduiTouch comes in enclosure for wall mounting (dimensions of enclosure are: 120 x 80 x 35mm), together with a 2.4 inch resistive touch screen. This enables you to permanently mount your project in your room or other exposed places.

Touchscreen:

» 2.4 inch resistive touchscreen, with 320 x 240 pixel resolution
» Display driver chip ILI9341, communication via SPI interface
» Touch driver chip XPT2046, also communication via SPI interface

ArduiTouch ESP can be used together with:

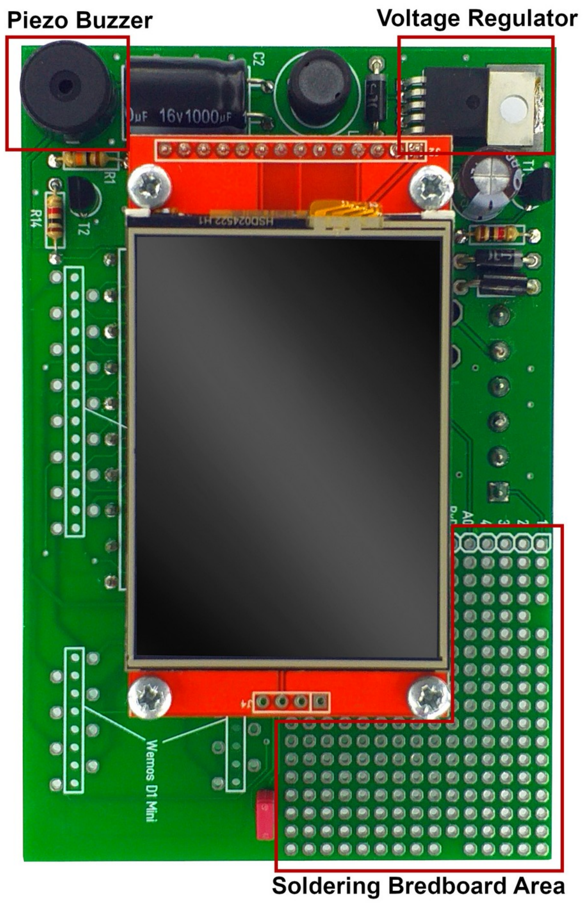
» NodeMCU Amica (ESP8266)
» Wemos D1 Mini (with some restrictions with Wemos Mini D1 ESP32 too)
» NodeMCU-32s (or ESP32 Dev Kit C)

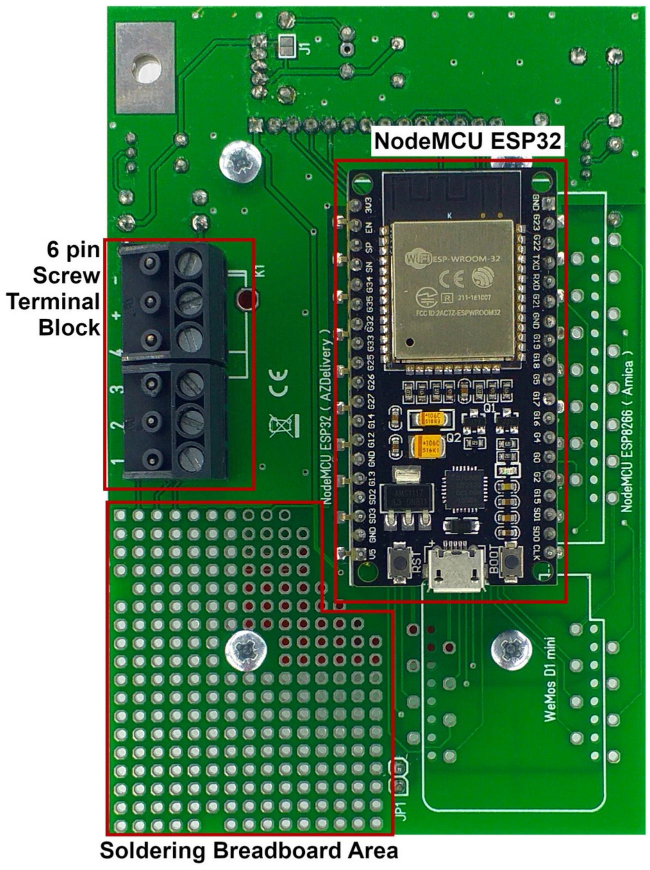
Depending on your project or your software you can mount the ArduiTouch set vertically or horizontally directly on the wall. A removable 6 pin screw terminal block makes installation easier. The board is mounted on the enclosure allowing a rapid substitution of the devices when needed.

The ArduiTouch comes with an integrated voltage regulator with input voltage range from 9V to 35V DC. The voltage regulator outputs stable *5V - 1.4A*. You can connect the power supply via screw terminal block or micro usb cable (via NodeMCU or D1 mini). *4* pins of the screw terminal are connected together with some GPIO pins.

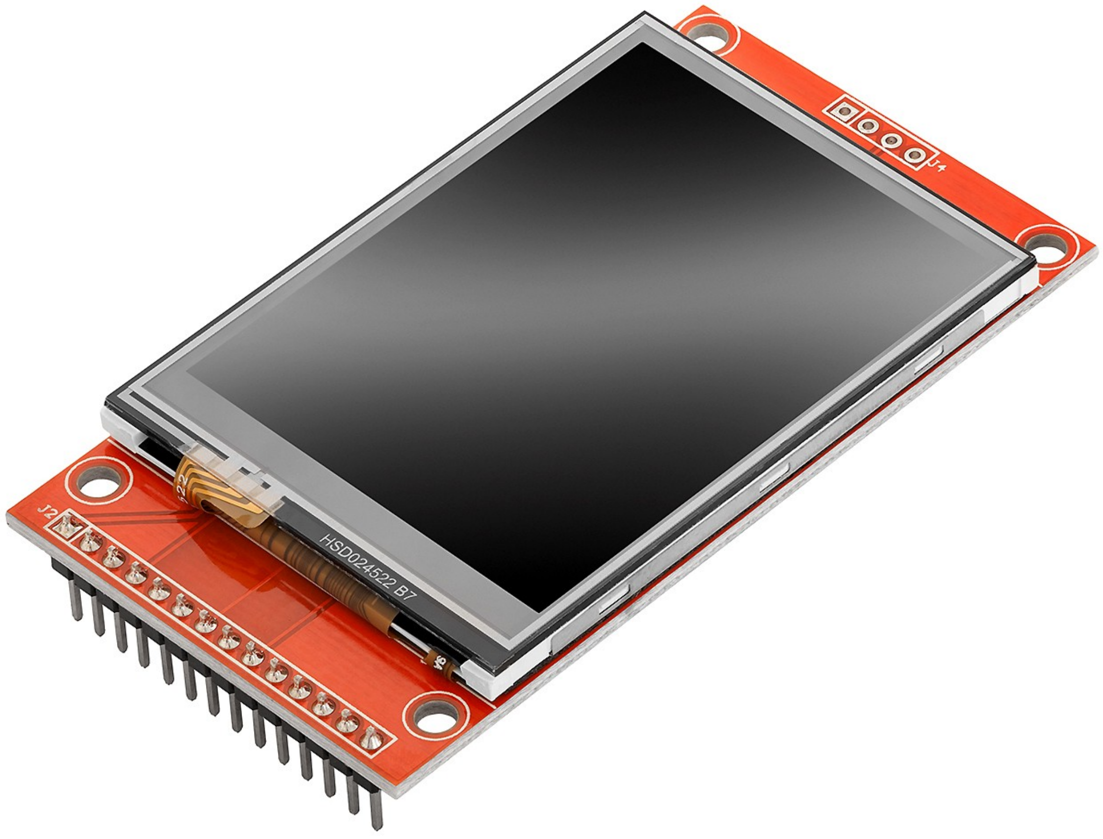
The integrated piezo buzzer with resonance frequency of *4kHz,* can be used to generate acoustic feedback, acknowledged tunes or melodies.

There is a small soldering breadboard area on the ArduiTouch pcb which can be used for your own projects.





TFT LCD touchscreen



TFT LCD touchscreens have wide usage in many projects. The touchscreen that comes with ArduiTouch has ILI9341 driver chip for LCD, and XPT2046 driver chip for touch. Both chips support SPI interface for communication.

It also has SD card reader (2GB) with SPI hardware, 4 pins (we will not cover it in this eBook).

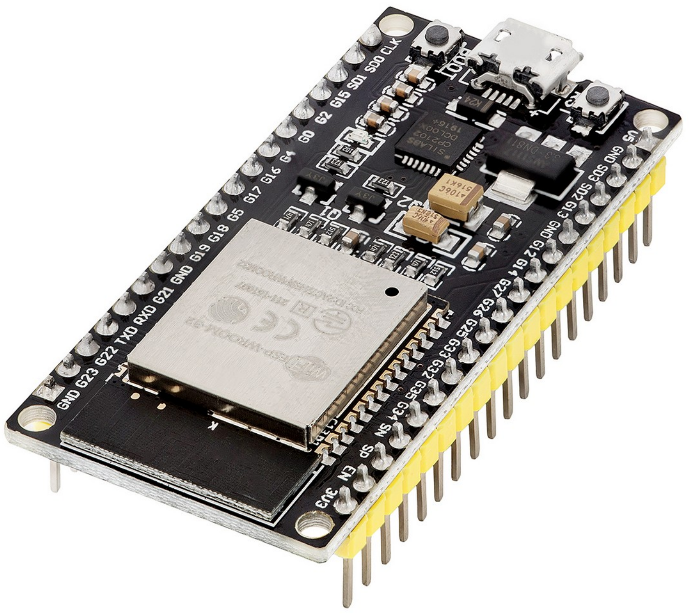
Specifications:

VCC power voltage:	from 3.3V to 5V DC
Logic IO port voltage:	3.3V DC (TTL)
Screen Size:	2.4 inch
Display Type:	TFT
Driver IC:	ILI9341
Display Colors:	RGB 65K colors
Resolution:	320 x 240 pixels
Module Interface:	SPI interface
Operating Temperature:	-20℃ ~ 60℃
LCD Active Area:	37 x 49mm
Module PCB Size:	77 x 43mm

TFT LCD touchscreen pinout:

1	VCC	3.3V or 5V power input		
2	GND	Ground		
3	CS	LCD chip select signal, low level enable		
4	RESET	LCD reset signal, low level reset		
5	DC/RS	LCD register / data selection signal,	HIGH level:	register
			LOW level:	data
6	SDI	(MOSI) SPI bus write data signal		
7	SCK	SPI bus clock signal		
8	LED	Backlight control, high level lighting, if not controlled, connect it to 3.3V - always bright		
9	SDO	(MISO) SPI bus read data signal, if you do not need to read function, leave it unconnected		
10	T_CLK	Touch SPI bus clock signal		
11	T_CS	Touch screen chip select signal, low level enable		
12	T_DIN	Touch SPI bus input		
13	T_DO	Touch SPI bus output		
14	T_IRQ	Touch screen interrupt signal LOW level when touch is detected		
15	SD_CS	SD card chip select signal, LOW level enable		
16	SD_MOSI	SD card SPI bus write data signal		
17	SD_MISO	SD card SPI bus read data signal		
18	SD_SCK	SD card SPI bus clock signal		

NodeMCU-32S (ESP32 Dev Kit C)



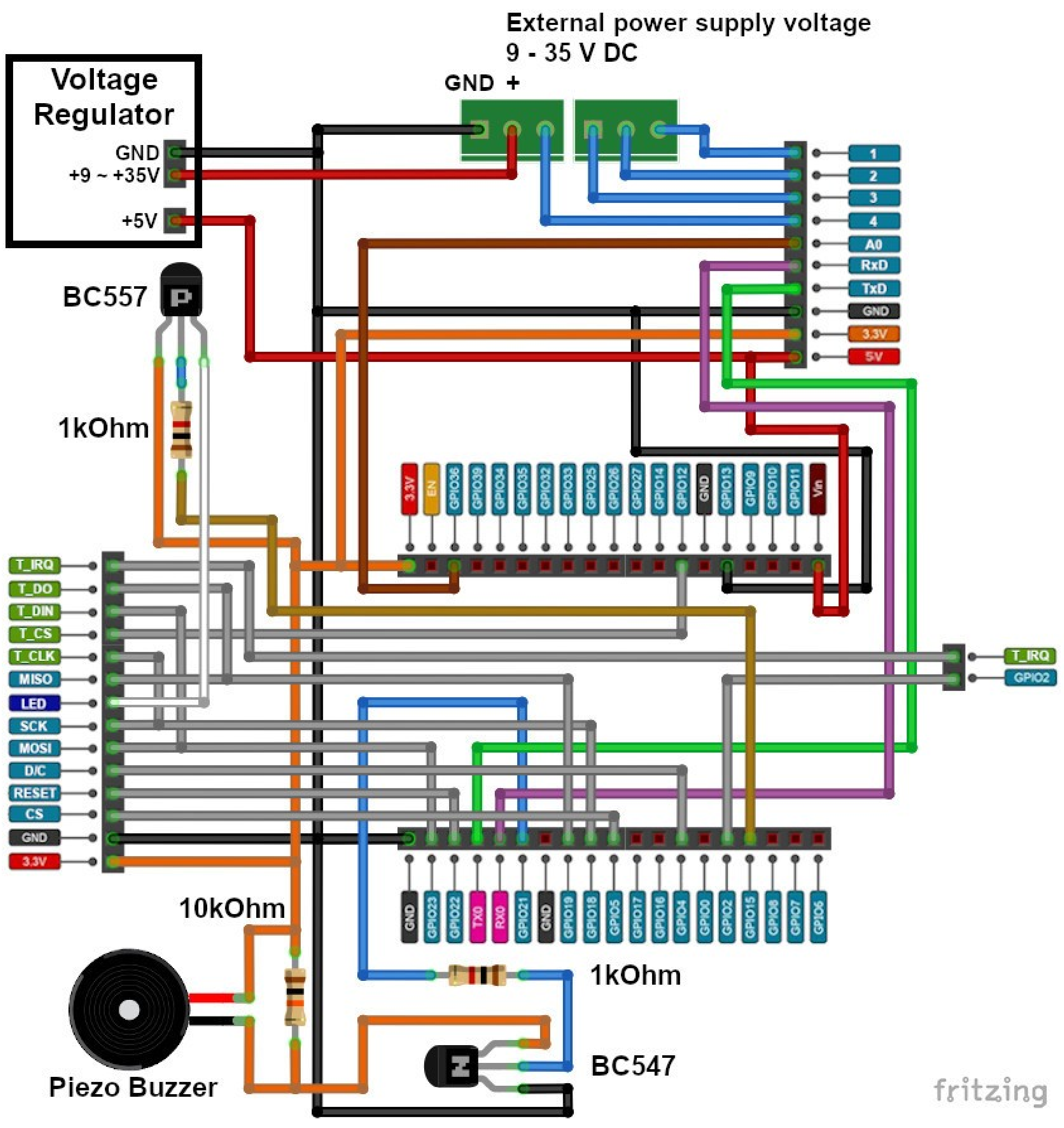
We all love Arduino and the Arduino IDE, but it is the time to introduce small but powerful NodeMCU-32S board. You have to spend only five minutes to install the ESP32 Arduino core and USB drivers and you are good to go!

The NodeMCU-32S is one of the development boards created by NodeMCU. Based on the ESP32 board, the NodeMCU-32S is an open source IoT (Internet of Things) platform. It is based on the ESP32 microcontroller that features wifi, bluetooth, ethernet and low power support all in a single chip.

Specifications:

Operating voltage:	3.3V DC
Input voltage (VIN pin):	from 7V to 12V DC
Microcontroller:	ESP32
Frequency:	240MHz
Flash:	4MB
SRAM:	520KB
RAM:	320KB
Digital I/O pins:	28
Analog input pins:	8
Analog outputs pins:	2
UARTs:	3
SPIs:	2
I2Cs:	3
Wi-Fi:	IEEE 802.11 b/g/n/e/i
	- Integrated TR switch, balun, LNA, power amplifier
	- WEP or WPA/WPA2 authentication

When we connect the NodeMCU-32S with ArduiTouch and 2.4 inch touchscreen, connection diagram looks like connection diagram on the image below:



You can see a piezo buzzer onboard pcb of the ArduiTouch. It is connected to the GPIO pin 21 of NodeMCU-32s via *BC547* (NPN) transistor and *1k*Ω resistor. We need a transistor and a resistor, because GPIO pin of the NodeMCU is not powerful enough to drive piezo buzzer.

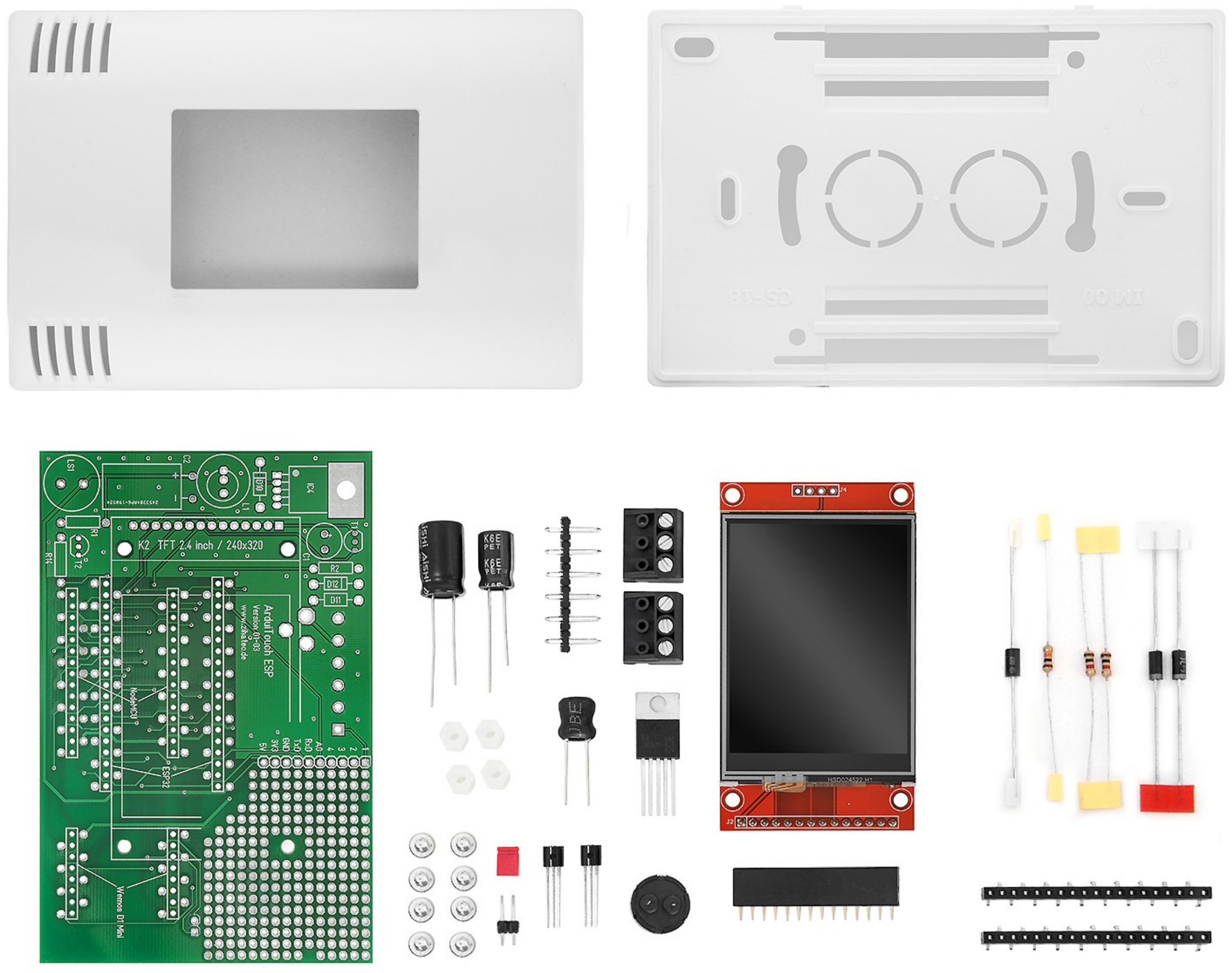
Backlight of the TFT LCD screen is connected to GPIO pin *15* of the NodeMCU via *BC557* (PNP) transistor and *1k*Ω resistor. We need a transistor and a resistor because of the same reason we need them for piezo buzzer.

You have the option to use touchscreen interrupt pin T_IRQ. To use it, connect two pins jumper near the soldering breadboard area (on the connection diagram, middle right side). By connecting this jumper, you connect *T_IRQ* pin of the TFT LCD screen to the GPIO pin *2* of the NodeMCU.

There is also *10* pin female header onboard pcb of the ArduiTouch. It is located beside soldering breadboard area. The header is connected with NodeMCU and screw terminal block. It is used to help you make easy connections between screw terminal, soldering breadboard area and NodeMCU.

The ArduiTouch comes unsoldered. You have to solder all elements on the pcb of the ArduiTouch. For this matter, you can find detailed instruction guid on our site:

https://delivery.shopifyapps.com/-/50463a3cab3c904e/853d04a9eac67d14



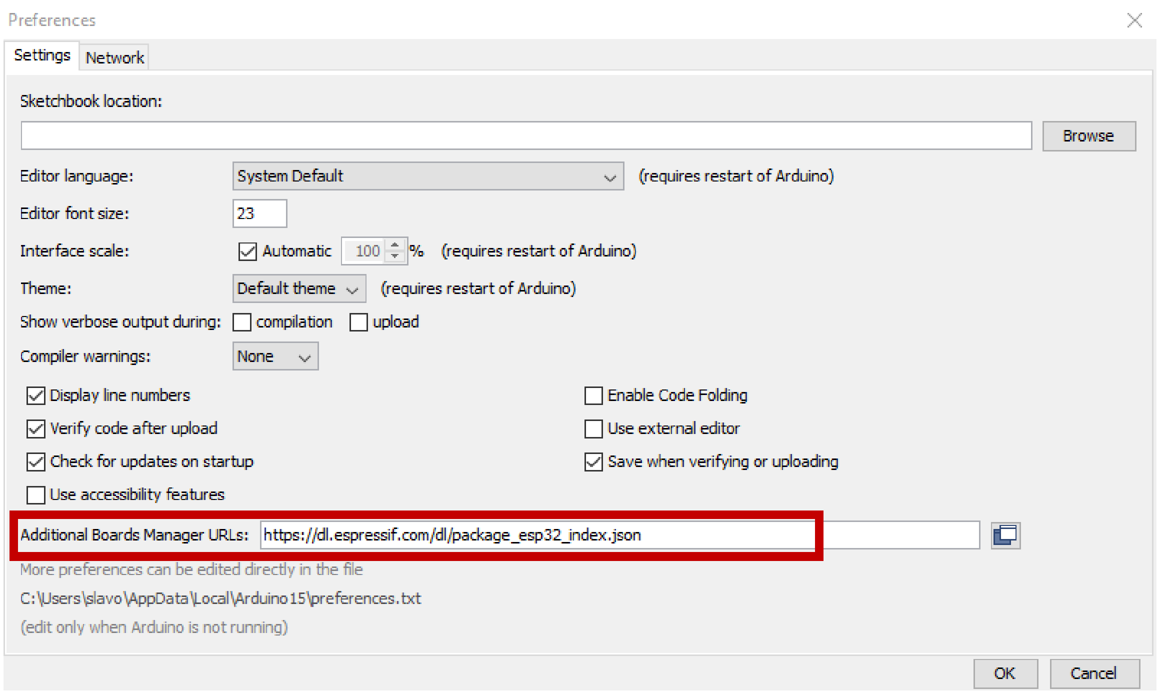
ESP32 and Arduino IDE

In order to program NodeMCU-32S with Arduino IDE, first install ESP32 core. There are several easy steps to follow in orter to install ESP32 core.

First you have to add one URL link to the Arduino IDE preferences. Open Arduino IDE and go to *File > Preferences*, a new window will open. Find field called "*Additional Board Manager URLs*", and copy following url: https://dl.espressif.com/dl/package_esp32_index.json

Then, paste it in "*Additional Board Manager URLs*" field. If you already have one URL inside, just add one comma after existing URL and paste new URL after it.

After this, click the *OK* button, and close the Arduino IDE.


Preferences

Settings | Network

Sketchbook location:

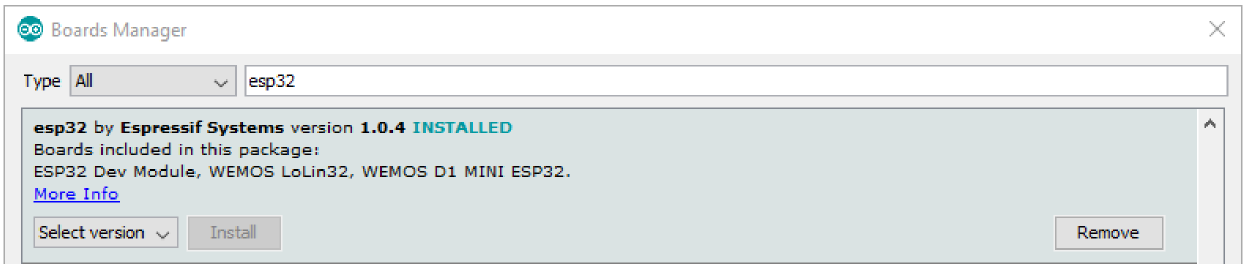
Editor language: System Default (requires restart of Arduino)
Editor font size: 23
Interface scale: ☑ Automatic 100 % (requires restart of Arduino)
Theme: Default theme (requires restart of Arduino)
Show verbose output during: ☐ compilation ☐ upload
Compiler warnings: None

☑ Display line numbers
☑ Verify code after upload
☑ Check for updates on startup
☐ Use accessibility features
☐ Enable Code Folding
☐ Use external editor
☑ Save when verifying or uploading

Additional Boards Manager URLs: https://dl.espressif.com/dl/package_esp32_index.json

More preferences can be edited directly in the file
C:\Users\slavo\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

Then you have to start Arduino IDE again. Go to:
Tools > Board > Boards Manager
A new window will open. Type in "*esp32*" the search box and install board called "*esp32*" by "*Espressif Systems*", as shown on the image below:



esp32 by **Espressif Systems** version **1.0.4** **INSTALLED**
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32, WEMOS D1 MINI ESP32.
More Info

And that is it! You have ESP32 core installed.

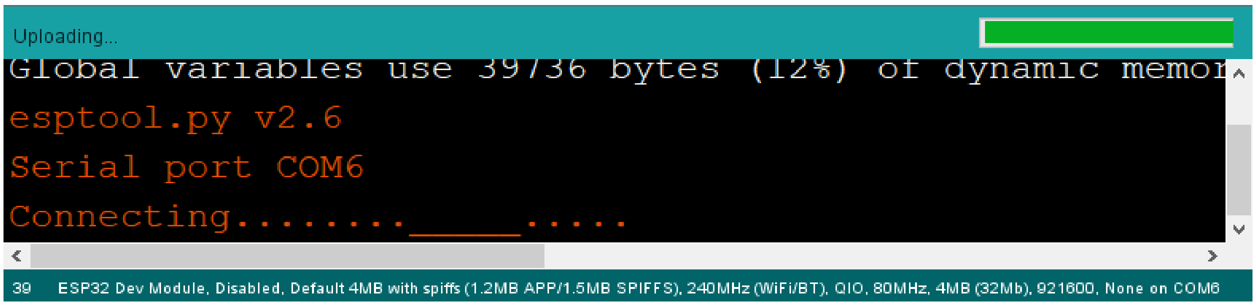
In order to use NodeMCU-32S (ESP32 Dev Kit C), we have to select this board in Arduino IDE. Go to: *Tools > Board > ESP32 Dev Module*.

Then you have to select port on which NodeMCU is connected,
Tools > Port > {port name}

And you are ready to upload your first sketch to the NodeMCU.

Uploading sketches to the NodeMCU-32S

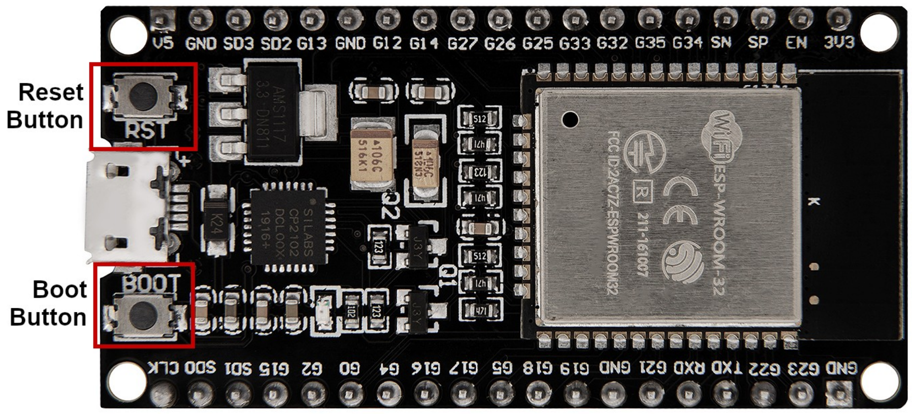
Uploading sketches to the NodeMCU is different than uploading them to the Arduino boards. When you hit the upload button in the Arduino IDE, you have to wait for compilation of the sketch. After that, message will be shown like on the image below:



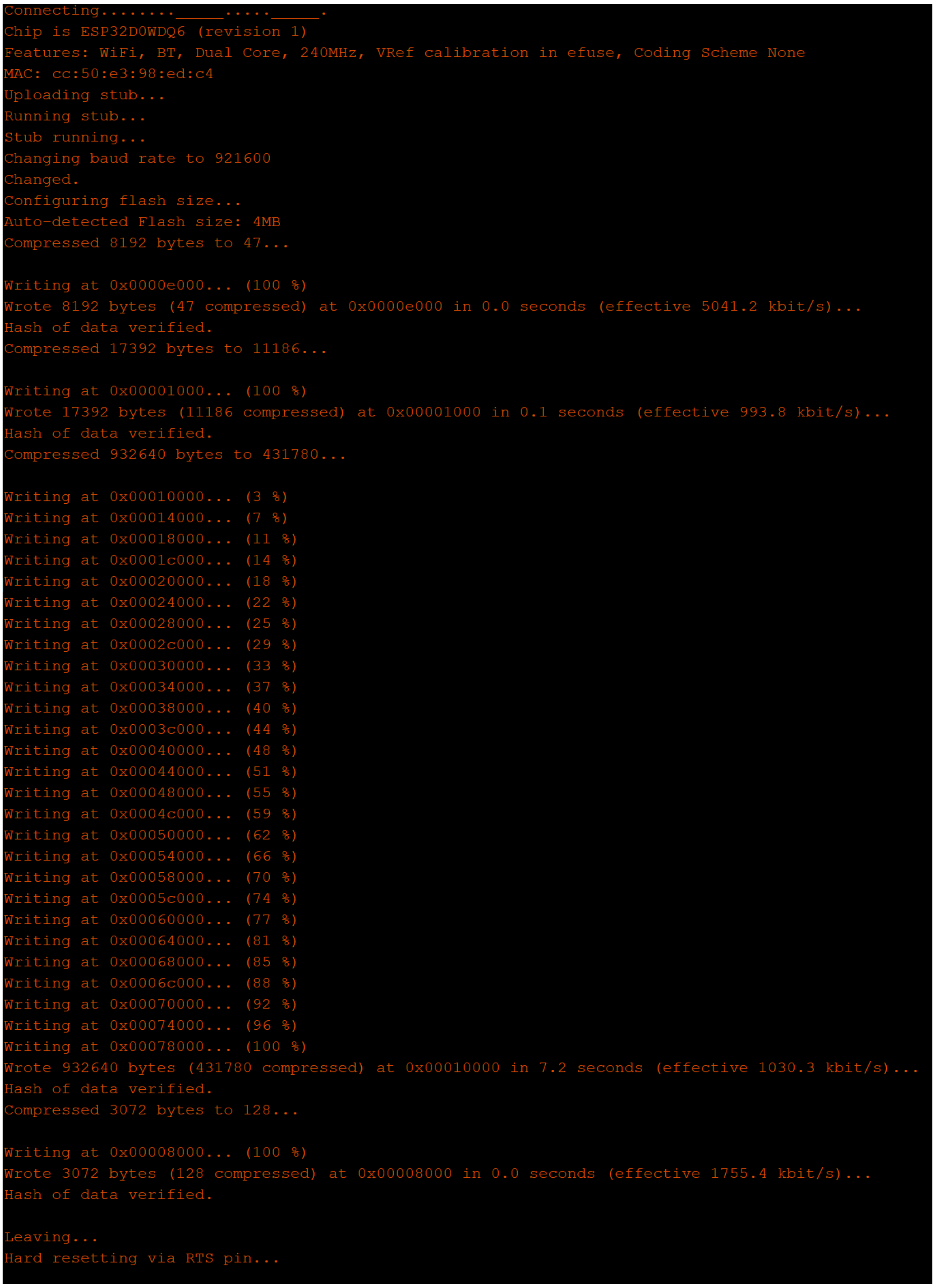
```
Uploading...
Global variables use 39736 bytes (12%) of dynamic memor
esptool.py v2.6
Serial port COM6
Connecting........_____.....
```

```
39    ESP32 Dev Module, Disabled, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32Mb), 921600, None on COM6
```

When this message is in the Arduino IDE you have to press boot button onboard NodeMCU:



Only when output message in the Arduino IDE changes its output to percentages, you can release the boot button. This is the time when the uploading process starts. Percentages show the status of the uploading process. You can find full output message on the image on the next page.



```
Connecting........_____....._____.
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC: cc:50:e3:98:ed:c4
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 921600
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 8192 bytes to 47...

Writing at 0x0000e000... (100 %)
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.0 seconds (effective 5041.2 kbit/s)...
Hash of data verified.
Compressed 17392 bytes to 11186...

Writing at 0x00001000... (100 %)
Wrote 17392 bytes (11186 compressed) at 0x00001000 in 0.1 seconds (effective 993.8 kbit/s)...
Hash of data verified.
Compressed 932640 bytes to 431780...

Writing at 0x00010000... (3 %)
Writing at 0x00014000... (7 %)
Writing at 0x00018000... (11 %)
Writing at 0x0001c000... (14 %)
Writing at 0x00020000... (18 %)
Writing at 0x00024000... (22 %)
Writing at 0x00028000... (25 %)
Writing at 0x0002c000... (29 %)
Writing at 0x00030000... (33 %)
Writing at 0x00034000... (37 %)
Writing at 0x00038000... (40 %)
Writing at 0x0003c000... (44 %)
Writing at 0x00040000... (48 %)
Writing at 0x00044000... (51 %)
Writing at 0x00048000... (55 %)
Writing at 0x0004c000... (59 %)
Writing at 0x00050000... (62 %)
Writing at 0x00054000... (66 %)
Writing at 0x00058000... (70 %)
Writing at 0x0005c000... (74 %)
Writing at 0x00060000... (77 %)
Writing at 0x00064000... (81 %)
Writing at 0x00068000... (85 %)
Writing at 0x0006c000... (88 %)
Writing at 0x00070000... (92 %)
Writing at 0x00074000... (96 %)
Writing at 0x00078000... (100 %)
Wrote 932640 bytes (431780 compressed) at 0x00010000 in 7.2 seconds (effective 1030.3 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 128...

Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.0 seconds (effective 1755.4 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

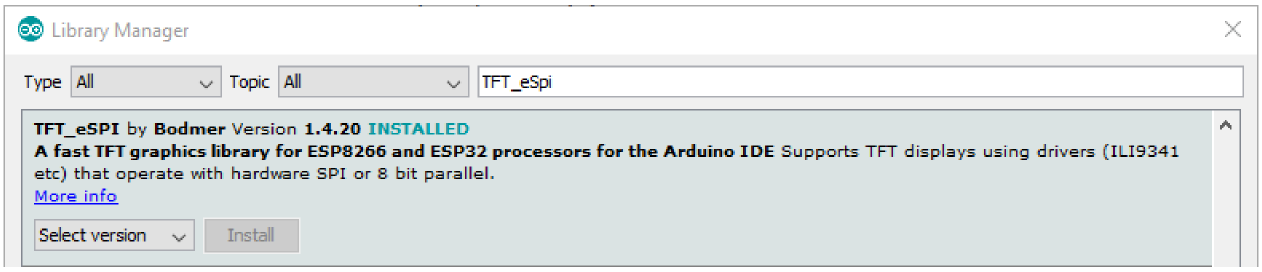
Weather App

In this eBook we will create and explain the app which shows weather data, time and date data, and Moon phases data on the TFT LCD screen.

Libraries

First thing that we need to do is to download external libraries.

First library is for TFT LCD screen, it is called "*TFT_eSPI*". Open Arduino IDE, and go to *Tools > Manage Libraries*. Type the name of the library in the search box and install library "*TFT_eSPI*" by "*Bodmer*", like on the image below:



Library Manager

Type: All | Topic: All | TFT_eSpi

TFT_eSPI by **Bodmer** Version **1.4.20** **INSTALLED**
A fast TFT graphics library for ESP8266 and ESP32 processors for the Arduino IDE Supports TFT displays using drivers (ILI9341 etc) that operate with hardware SPI or 8 bit parallel.
More info

For this library you have to edit one file. Go to directory where Arduino IDE saves sketches and libraries. If you did not change that directory, it is in *Documents > Arduino*. To find *TFT_eSPI* library directory, open *File Explorer*, and go to:
Documents > Arduino > libraries > TFT_eSPI

We have to edit file called “*User_Setup.h*”. Open it in any text editor (for example *Notepad* or *Sublime Text*).

Find following lines of the text (at *156* line number):

```
// ###### EDIT THE PIN NUMBERS IN THE LINES FOLLOWING TO SUIT YOUR ESP32 SETUP
// For ESP32 Dev board (only tested with ILI9341 display)
// The hardware SPI can be mapped to any pins
//#define TFT_MISO    19
//#define TFT_MOSI    23
//#define TFT_SCLK    18
//#define TFT_CS      15  // Chip select control pin
//#define TFT_DC      2   // Data Command control pin
//#define TFT_RST     4   // Reset pin (could connect to RST pin)
//#define TFT_RST     -1  // Set TFT_RST to -1 if display RESET is
                          // connected to ESP32 board RST
//#define TFT_BL      32  // LED back-light
                          // (only for ST7789 with backlight control pin)
//#define TOUCH_CS    21  // Chip select pin (T_CS) of touch screen
//#define TFT_WR      22  // Write strobe for modified Raspberry Pi TFT only
```

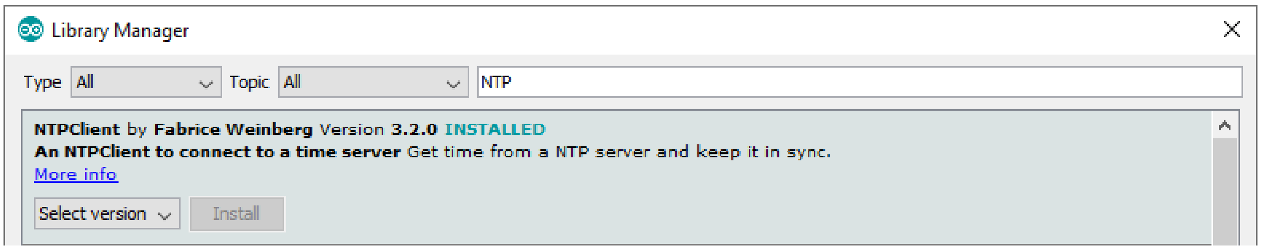
and change it to this:

```
#define TFT_MISO  19
#define TFT_MOSI  23
#define TFT_SCLK  18
#define TFT_CS    5
#define TFT_DC    4
#define TFT_RST   22
//#define TFT_RST -1
//#define TFT_BL  32
#define TOUCH_CS  14
//#define TFT_WR  22
```

After this, save the file. We need to make these changes because pins of the TFT LCD screen are connected to pins of the NodeMCU-32S like in the changed text (you can see it on the connection diagram).

Backlight pin is controlled separately, so we do not set it up here.

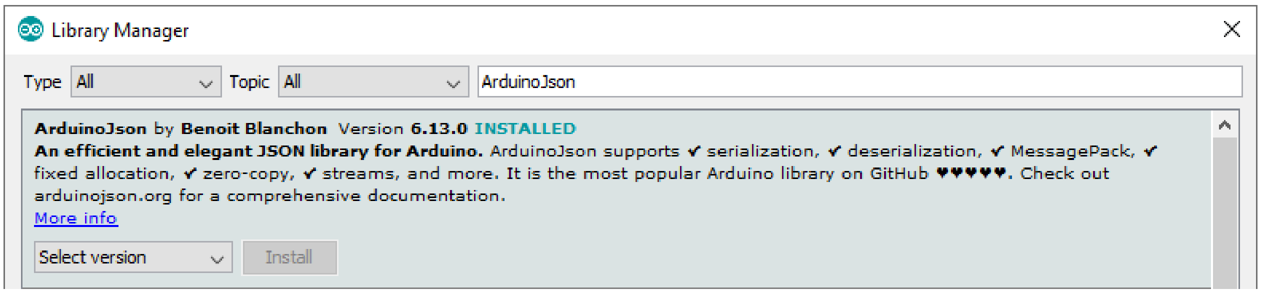
The second library is for Real Time API called "*NTPClient*". Open Arduino IDE, and go to *Tools* > *Manage Libraries*. Type "*NTP*" in the search box and install library "*NTPClient*" by "*Fabrice Wenberg*", like on the image below:



NTP

NTPClient by **Fabrice Weinberg** Version **3.2.0** **INSTALLED**
An NTPClient to connect to a time server Get time from a NTP server and keep it in sync.
More info

The third library is for parsing JSON data, it is called "*ArduinoJson*". Open Arduino IDE, and go to *Tools* > *Manage Libraries*. Type the name of the library in the search box and install library "*ArduinoJson*" by "*Benoit Blanchon*", like on the image below:



ArduinoJson

ArduinoJson by **Benoit Blanchon** Version **6.13.0** **INSTALLED**
An efficient and elegant JSON library for Arduino. ArduinoJson supports ✔ serialization, ✔ deserialization, ✔ MessagePack, ✔ fixed allocation, ✔ zero-copy, ✔ streams, and more. It is the most popular Arduino library on GitHub ♥♥♥♥♥. Check out arduinojson.org for a comprehensive documentation.
More info

The fourth library is for Moon Phase API, called “*moonPhases*”. You can not find this library in the Manage Libraries. We have to download it from the *GitHub*. Go on this link: https://github.com/CelliesProjects/moonPhase
and download *.zip* file. Then, to add it to the Arduino IDE, open Arduino IDE and go to *Sketch > Include Library > Add .ZIP Library…* and add the downloaded *.zip* file.

The fifth library is for converting timestamps, called “*TimeLib*”. You can not find this library in the Manage Libraries. We have to download it from the *GitHub*. Go on this link: https://github.com/PaulStoffregen/Time
and download *.zip* file. Then, to add it to the Arduino IDE, open Arduino IDE and go to *Sketch > Include Library > Add .ZIP Library…* and add the downloaded *.zip* file.

Test sketch

In order to test if everything works perfectly, open Arduino IDE and open a sketch called "*Keypad_240x320*":
File > Examples > TFT_eSPI > 320 x 240 > Keypad_240x320

At the beginning of the `setup()` function add following lines of code:

```
pinMode(15, OUTPUT);
digitalWrite(15, LOW);
delay(1);
```

Upload the sketch to the NodeMCU.

These lines of code are used to turn ON the backlight of the TFT LCD screen. More about this later.

When the sketch is uploaded, you will see the keypad on the TFT LCD screen. Open Serial Monitor (*Tools > Serial Monitor*) and type random number on the TFT LCD screen and press the *Send* button. If everything works well, you should see typed number from TFT LCD screen in the Serial Monitor.

Weather App code

The code for this app can be found and downloaded from the *GitHub* repository: https://github.com/Slaveche90/WeatherAppArduiTouch.git

There are three files on the *GitHub* rapository:
TFT_weatherApp.ino file (main app),
xbm.h (file for weather icon images) and
moonPhs.h (file for moon phase icon images).

TFT_weatherApp.ino File

Code is commented pretty well, but there are few things that need to be explained more thoroughly.

The main app starts with import of libraries. After that we create and initialize several variables and objects.

Clock data is updated every second, that is why `second_interval` variable is used. We store value *1000* in this variable, which is *1000* milliseconds.

Weather data is updated every hour, that is why `weather_interval` variable is used. We store value *3600* in this variable because there are *3600* seconds in one hour. All we have to do is to count *3600* seconds.

Values of char arrays called `ssid[]` and `pass[]` need to be changed, because char arrays contain name and password of the wifi network on which ESP32 connects.

We use `Open Weather Map API` (OWM API) to get weather data. We create an initialize several objects for this. Register on the following link https://openweathermap.org/ in order to get the API code. You can not use the API without this code. API code is a string containing long hexadecimal number. Store it in the string called `apiKey`.

We store OWM API server name in the constant char array called `server[]`.

In the string `nameOfCity`, we store city name and state code in the format "`city_name,CODE`". The state code is world wide state code from the `ISO 3166-2` standard https://en.wikipedia.org/wiki/ISO_3166-2 .

After this come many function definitions. All TFT related functions are recreated from the sketch called "*TFT_graphictest_one_lib*":
File > Examples > TFT_eSPI > 320 x 240 > TFT_graphictest_one_lib

First function is `printText()`. This function accepts *6* arguments and it is used to print text, change the size of text, change the text alignment in the line of the screen. Text is displayed in one text line only that is always the same lenght as the screen.

There are *5* different text sizes, (*1*, *2*, *3*, *4* and *5*). *TFT_eSPI* library has *8* different text sizes, but we did not use text size bigger than *5*. The smallest text size have *5* pixel character length and *1* pixel between characters. With every size, these character sizes increase, character length increases *5* times per text size, and length between characters increases by one pixel per text size.

There are three text alignment values:
1 - left alignment,
2 - center alignment and
3 - right alignment.

If the text exceeds the screen length, there are two `if` blocks to make text size smaller in order to fit the screen.

There is the text alignment function in the *TFT_eSPI* library, which we did not use. You can check it in the example sketch:
File > Examples > TFT_eSPI > 320 x 240 > TFT_String_Align

Next function is called `initializeLCD()` and, as its name says, it is used to initialize TFT LCD touchscreen. Function accepts no argument and returns no value. At the beginning of this function, first we set the pin mode of digital pin *15* as *OUTPUT*, and then drive it *LOW*. This is necessary because backlight pin of the TFT LCD touchscreen is connected to digital pin 15 of the NodeMCU. Digital pin is connected to the backlight pin via *1kΩ* resistor and *BC557* transistor because output pins of ESP32 do not have enough power to drive a backlight. You can even use PWM (Pulse Width Modulation) on this pin, to set the backlight level on specific value other than turned *ON* or *OFF*. In this application we turn *ON* the TFT LCD touch screen, and as long as it is connected to the power supply it stays *ON*.

After that, we initialize display object, fill the screen with black color and set the text color to green.

At the end of this function different font option is created (other than default font), but it is not used. Be careful with using another font, because different font sets new character sizes, that changes overall look of the app!

Next function is called *`initializeTime()`*, and it also does what its name says, it is used to connect to the *NTPClient API* and initialize time. Function accepts one argument and returns no value. The argument represents time offset of the timezone in seconds.

Next function is called *`connectToWiFi()`*, and it is used to make connection to the local wifi network. Function accepts no argument and returns no value. Function of the *`connectToWiFi()`* is connectiong to specified network and printing output. Depending on connection realisations output can be message of successiful or unsuccessful connection.

Next function is called *`printWiFiStatus()`*, and it is used to output the status of wifi connection, ip address and strength of the wifi signal. Function accepts no argument and returns no value.

Next function is called *`makehttpRequest()`* and it is used to establish a connection with OWM API and to request JSON data. Function accepts no argument and returns no value. At the beginning of the function, we clear the display and stop all connections with the OWM API. Then we try to connect to the OWM API server with port 80. If connection is successful, then we send four strings to the server, called *`t1`*, *`t2`*, *`t3`* and *`t4`*. First string is GET request for specific city and state code, with metric units:

```
String t1="GET /data/2.5/forecast?q=" + nameOfCity + "&APPID=" +
               apiKey + "&mode=json&units=metric&cnt=2 HTTP/1.1";
```

If you wish not use *`units`* tag, remove following text from *`t1`* string: `&units=metric`. This way temperature data unit will be in Kelvins. There is an option for imperial units, too; to use it, change the text "*metric*" to "*imperial*".

Second string is the api server name, and the third is user agent:

```
String t2 = "Host: api.openweathermap.org";
String t3 = "User-Agent: ArduinoWiFi/1.1";
String t4 = "Connection: close";
```

After this, we send these strings to the server and wait for response. If server responds successfully, we read the data that server sends back. If the data is as expected, call the *`parseJson()`* function. Data is as expected if JSON file contains an equal number of open and closed curly brackets { }.

Next function is called *`parseJson()`* and it is used to parse JSON data and output the data on the screen. Function accepts no argument and returns no value. The argument is a char array or string containing JSON data. How JSON data is parsed can be found in the sketch examples that come with ArduinoJson library, for example:
File > Examples > ArduinoJson > JsonParserExample

We use following two objects to create an array or a list of JSON data:

```
JsonArray list = doc["list"];
JsonObject current = list[0];
```

All we have to do is to access specific fields of these objects. We do it with following lines of the code:

```
String city         = doc["city"]["name"];
uint32_t timezone   = doc["city"]["timezone"];
float temp          = current["main"]["temp"];
float humidity      = current["main"]["humidity"];
String weather      = current["weather"][0]["main"];
String weatherDescription = current["weather"][0]["description"];
String weatherCode = current["weather"][0]["icon"];
uint32_t sunrise    = doc["city"]["sunrise"];
uint32_t sunset     = doc["city"]["sunset"];
```

You can see how the JSON data looks like on the following link:
https://openweathermap.org/api/hourly-forecast
Search for the title "*Weather parameters in API response*".

At the end of this function, we call the function for displaying the page on the ArduiTouch. This function is called *`page()`*.

Next function is *`page()`* function. Function of *`page()`* is to call *`initializeTime()`* function, read time and date, to clear the screen and to call *`showPage()`* function.

Next function is *`showPage()`* function and it is used to output all data on the TFT LCD screen. Function accepts *9* arguments and returns no value. Arguments are:

- dateTime	epoch timestamp
- city	city name for weather data
- temp	temperature data
- humidity	humidity data
- weather	weather condition name from the OWM API
- weatherDescription	weather condition name, long version
- sunrise	epoch timestamp of sunrise
- sunset	epoch timestamp of sunset
- weatherCode	the code name of image used in the OWM API

Within *`showPage()`* function *`printText()`*, *`printTime()`*, and *`printDayAndDate()`* functions output data on the screen. *`showWeatherIcon()`* and *`showMoonIcon()`* functions output weather and moon icon images on the screen, respectively.

`printTime()` function is used to output time to screen, in format *HH:MM:SS*. This function accepts *6* arguments and return no value. The arguments are: *X* and *Y* position of the cursor, *epoch timestamp*, *textSize*, *textAlign*, *lineLength*. Epoch timestamp is unix timestamp, a number of seconds starting from *January the 1st 1970* up to current *UTC/GMT* time. NTPClient API initializes this timestamp for the timezone of the city used in the app. We did this when we called the *`initializeTime(timezone)`* function.

To convert timestamp into readable format, we used *`TimeLib`* library functions:

`time_t t = dateTime;`	
`hour(t)`	- returns hours from epoch timestamp
`minute(t)`	- returns minutes from epoch timestamp
`second(t)`	- returns seconds from epoch timestamp

In order for the output to be in the format *HH:MM:SS*, we need to check if the numbers are single digits. If this is the case, we have to add leading zero. We do this by calling *`checkZero()`* function.

printDayAndDate() function is used to output specific day and date on the screen. Function accepts *6* arguments, the same arguments as in the *printTime()* function and return no value. Here we use *TimeLib* library functions to convert epoch timestamp into readable date. Date format is *DD. MONTH YYYY*. *TimeLib* library functions *weekday()* and *month()* return numbers. To convert these numbers into words (names), we use the following lines of the code:

```
time_t t = dateTime; // dateTime is epoch timestamp
String daysOfWeek[7]  = { "Sunday", "Monday", "Tuesday",
                         "Wednesday", "Thursday",
                         "Friday", "Saturday"};
String monthNames[12] = { "January ", "February ", "March ",
                         "April ", "May ", "June ", "Jully ",
                         "August ", "September ",
                         "October ", "November ", "Dececember "};
String dayName = daysOfWeek[weekday(t) - 1];
String monthName = monthNames[month(t) - 1];
```

`showWeatherIcon()` and `showMoonIcon()` are used to output weather icon image and moon phase icon image, respectively. Both functions are created from *TFT_eSPI* library sketch, called “*TFT_Flash_Bitmap*”:
File > Examples > TFT_eSPI > Generic > TFT_Flash_Bitmap

In order to use `pushImage()` function of the *TFT_eSPI* library, first we have to create "*.c*" file out of the image we desire to output to the screen. To do so we have to download external app called “*ImagesConverter565.exe*”. You can download it from this link:
https://github.com/mysensors/MySensorsArduinoExamples/blob/master/libraries/UTFT/Tools/ImageConverter565.exe

Open the external app, load the image you wish to convert to "*.c*" file, and click the *Save* button. The output is "*.c*" extension, open it in any text editor, like *Notepad*, and copy the content starting from:

```
const unsigned short nameOfImage[0x1324] PROGMEM ={
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, …….. 
……
```

until the end of the file. Then create new "*.h*" file inside the Arduino IDE, and paste the copied content to the file. This new *.h* file has to start with following lines of the code:

```
#include <pgmspace.h>
const uint16_t image_width = 70;
const uint16_t image_height = 70;
```

After which comes images data from "*.c*" files. We need these two constants because they are used in the `pushImage()` as arguments.

We created two separate "*.h*" files; one for weather icon images, and the second for moon icon images, called *xbm.h* and *moonPhs.h*, respectively. We did this in order to make the app code more readable. All images used in one "*.h*" file are the same width and height, therefore we use constants. If an image differ in size, separate constants need to be created specifically for that image.

All weather icon images that we use are *70* x *70* pixels, and all moon icon images are *60* x *60* pixels. There are *13* weather and *16* moon icon images inside the *WeatherApp*.

`showWeatherIcon()` function accepts one argument and returns no value. The argument is a string containing weather icon code used in OWM API. Depending on the code we output specific image for the specific weather.

`showMoonIcon()` function accepts one argument and returns no value. The argument is unsigned integer containing the angle of the Moon, or phase of the Moon. The angle argument value ranges from *0* to *360.* Depending on this value we output specific image for the moon phase.

And last two functions used in the app are *readTimeDate()* and *updateTime()*.

The first function, *readTimeDate()* accepts no arguments and returns unsigned integer value which is epoch timestamp. This function tries to get data from NTPClient server and returns data.

The second function, *updateTime()* is used to update output that is displayed on the screen. It accepts no arguments and returns no value and *updateTime()* function displays black rectangle over clock data and then outputs new clock data.

The App

You've done it!

Now you can use your module for various projects.

Now is the time to learn and make the Projects on your own. You can do that with the help of many example scripts and other tutorials, which you can find on the internet.

If you are looking for the high quality products for Arduino and Raspberry Pi, AZ-Delivery Vertriebs GmbH is the right company to get them from. You will be provided with numerous application examples, full installation guides, eBooks, libraries and assistance from our technical experts.

https://az-delivery.de

Have Fun!

Impressum

https://az-delivery.de/pages/about-us