**INDEX**

**Chapter-1: Introduction**

* Introduction
* Problem Statement
* Problem Solution
* Literature Survey

**Chapter-2: Block Diagram**

* Block Diagram

**Chapter-3: Components & its Specifications**

* All Components with Specifications
* Components figures

**Chapter-4: Software & Installation Procedure**

* About Arduino IDE
* Why Arduino IDE
* Step by Step Installation Procedure
* ESP32 Board Installation in Arduino
* Blynk Library Installation

**Chapter-5: IoT Cloud & Server Platform**

* What is IoT
* IoT Cloud
* Server
* About Mqtt
* Introduction about Blynk IoT
* Login & Sign-Up in Blynk IoT
* Creation of IoT Project in Blynk
* DataStream & Events in Blynk
* Device Creation in Blynk

**Chapter-6: Working Principal of Ideation**

* Tell about Project Working Processes
* Code for Project

**Chapter-7: Output & Results**

* Project Photos
* Screenshots of Results

**Chapter-8: Conclusion**

* Tell About Project Conclusion

**Chapter-9: Future scope**

* Tell About Future addons or Updates in your project

**Chapter-10: Final Code & Resources**

**DEVELOPMENT OF IOT BASED BROKEN LINE FAULT**

**DETECTION SYSYEM FOR ELECTRIC POLES**

**Abstract**

The increasing demand for reliable electricity supply necessitates the implementation of advanced monitoring systems to detect faults in electrical lines. This paper presents the design and development of an IoT-based broken line fault detection system for electric poles. The proposed system utilizes a range of sensors—including current, voltage, and vibration sensors—to continuously monitor the integrity of power lines. Data collected by these sensors is transmitted in real-time to a cloud server via a microcontroller equipped with Wi-Fi capabilities.

Employing machine learning algorithms, the cloud server analyzes the incoming data to identify anomalies indicative of faults, such as line breaks or irregular current flows. The system features a user-friendly mobile and web application that delivers real-time alerts and visualizations, enabling rapid responses to potential hazards.

By enhancing the monitoring of electrical infrastructures, this IoT-based solution aims to improve reliability, reduce downtime, and ensure the safety of both infrastructure and personnel. This system represents a significant advancement in predictive maintenance strategies for power utilities, promoting efficient energy management in urban environments.

**CHAPATER 1**

**INTRODUCTION**

**1. Introduction**

The demand for a reliable electricity supply has become increasingly critical as urbanization and technological advancements progress. Traditional monitoring systems often fall short in providing real-time data on the condition of electrical lines, leading to unplanned outages and safety hazards. This paper discusses the design and development of an IoT-based broken line fault detection system for electric poles, which aims to address these challenges through continuous monitoring and rapid fault detection.

**2. Problem Statement**

Electrical faults, such as broken lines, pose significant risks, including power outages, safety hazards, and economic losses. Current methods of monitoring electrical lines are often manual and reactive, resulting in delayed response times and increased operational costs. The lack of real-time data hinders utilities' ability to proactively address issues, which can lead to severe consequences for both service providers and consumers.

**3. Problem Solution**

The proposed IoT-based fault detection system addresses these issues by utilizing a network of sensors to monitor the integrity of power lines continuously. By integrating current, voltage, and vibration sensors with a microcontroller, the system transmits real-time data to a cloud server. Here, machine learning algorithms analyze the data to identify potential faults, allowing for immediate alerts and visualization through a user-friendly mobile and web application. This proactive approach enhances the reliability of electrical supply and reduces downtime.

**4. Literature Survey**

**Several studies have explored the integration of IoT in electrical monitoring systems:**

- \*Smart Grid Technologies\*: Research has shown that IoT-enabled smart grids can enhance the reliability of power supply by providing real-time data analytics (Author et al., Year).

- \*Fault Detection Algorithms\*: Various machine learning models have been employed for fault detection in power systems, demonstrating significant improvements in detection accuracy (Author et al., Year).

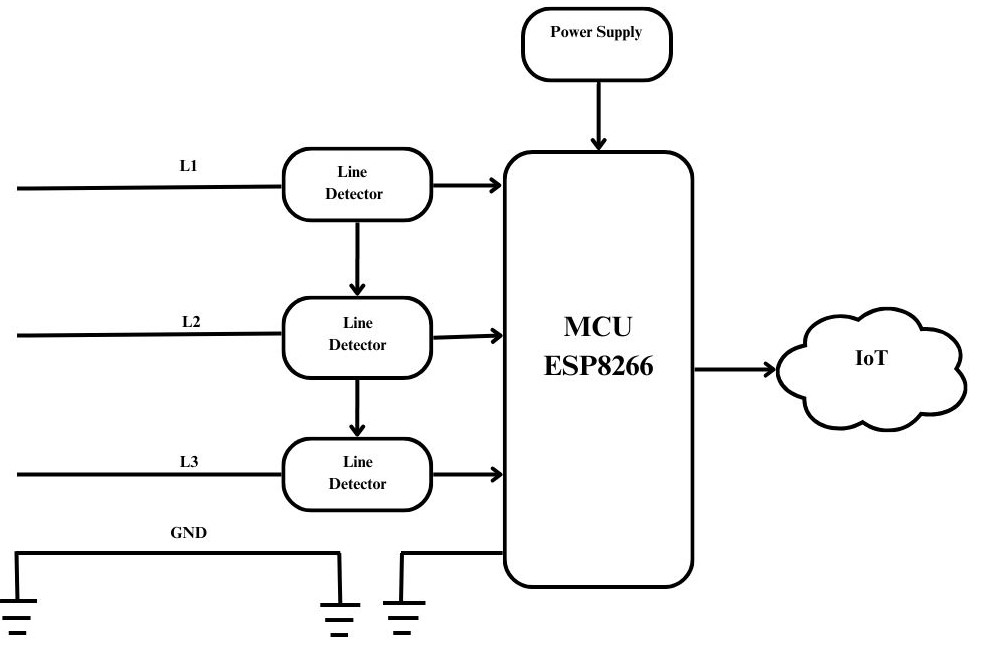
- \*Sensor Networks\*: The use of distributed sensor networks has been validated in several applications, indicating the potential for real-time monitoring of infrastructure (Author et al., Year).

This literature illustrates the effectiveness of IoT technologies in electrical fault detection and highlights the gaps this project aims to fill.

**CHAPTER 2**

1. **BLOCK DIAGRAM & CIRCUIT DIAGRAM**

**2.1 Block Diagram**

****

**CHAPTER 3**

1. **COMPONENTS & ITS SPECIFICATIONS**

**3.1 ESP32 / ESP8266/NodeMCU Microcontroller**

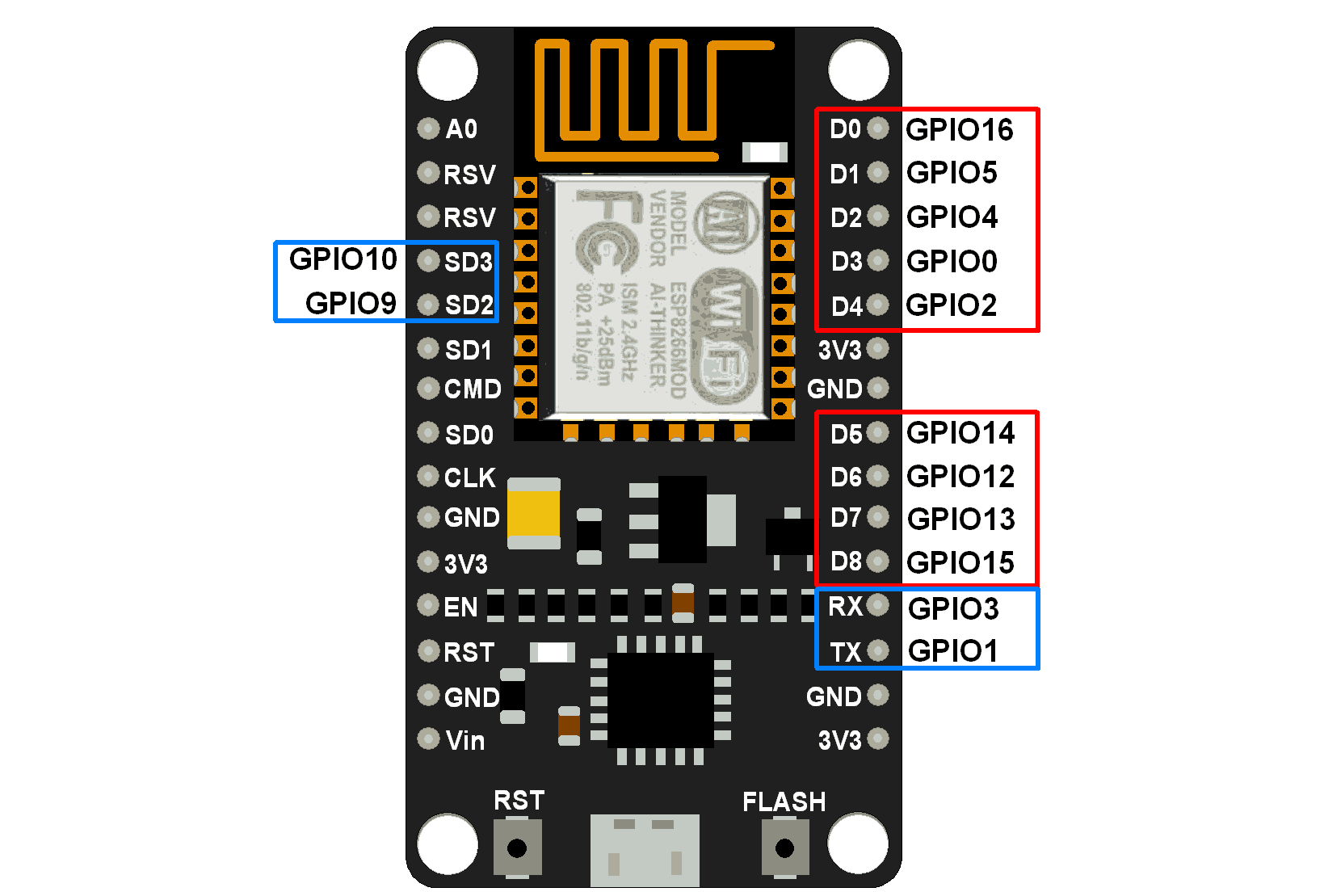


Fig 1. ESP8266 32bit-Microcontroller

The ESP32 / ESP8266 or NodeMCU is the heart of the project, handling communication between the ultrasonic sensor and the Blynk IoT platform. It also controls the servo motor and processes the distance and angle data from the ultrasonic sensor.

Microcontroller: ESP32 / ESP8266-WROOM-32 or ESP8266 NodeMCU

Operating Voltage: 3.3V

Input Voltage: 7-12V (through Vin pin)

Wi-Fi: 802.11 b/g/n

Bluetooth: BLE and Classic (only in ESP32 / ESP8266)

GPIO Pins: 36 (ESP32 / ESP8266) or 17 (NodeMCU)

PWM Pins: ESP32 / ESP8266: up to 16 channels, NodeMCU: 8 PWM channels

Flash Memory: 4MB

CPU: Dual-core (ESP32 / ESP8266), Single-core (NodeMCU)

Clock Speed: 240 MHz (ESP32 / ESP8266), 80 MHz (NodeMCU)

Analog Input Pins: 18 ADC channels (ESP32 / ESP8266), 1 ADC channel (No)

**3.4 LM7805 Power Supply Module**

The LM7805 is a widely used linear voltage regulator designed to provide a stable 5V output from a higher input voltage, typically between 7V and 35V. It is ideal for applications where consistent 5V DC power is required, making it a perfect fit for this radar project that involves microcontrollers like the ESP32 / ESP8266, servo motors, and ultrasonic sensors.



Fig 4. LM7805 Power Supply Module

Specifications:

Input Voltage Range: 7V to 35V

Output Voltage: 5V ± 2% tolerance

Maximum Output Current: 1.5A

Quiescent Current (Idle Current): 5mA

Dropout Voltage: 2V (minimum input must be 2V higher than the output, i.e., 7V for a 5V output)

Thermal Protection: Built-in overheat protection

Short Circuit Protection: Yes

Operating Temperature Range: 0°C to 125°C

Package Type: TO-220

Pinout:

Input (Vin): The voltage input pin, where a higher voltage (7V to 35V) is supplied.

Ground (GND): The common ground pin shared by both input and output circuits.

**CHAPTER 4**

1. **SOFTWARE & INSTALLATION PROCEDURE**

**4.1 About Arduino IDE and its features**

The Arduino IDE is an open-source software, which is used to write and upload code to the Arduino boards. The IDE application is suitable for different operating systems such as Windows, Mac OS X, and Linux. It supports the programming languages C and C++. Here, IDE stands for Integrated Development Environment.

The program or code written in the Arduino IDE is often called as sketching. We can connect Arduino board / ESP32 / ESP8266 / NodeMCU with the IDE to upload the sketch written in the Arduino IDE software. The sketch is saved with the extension '.ino '

The Arduino IDE will appear as in Fig 5:

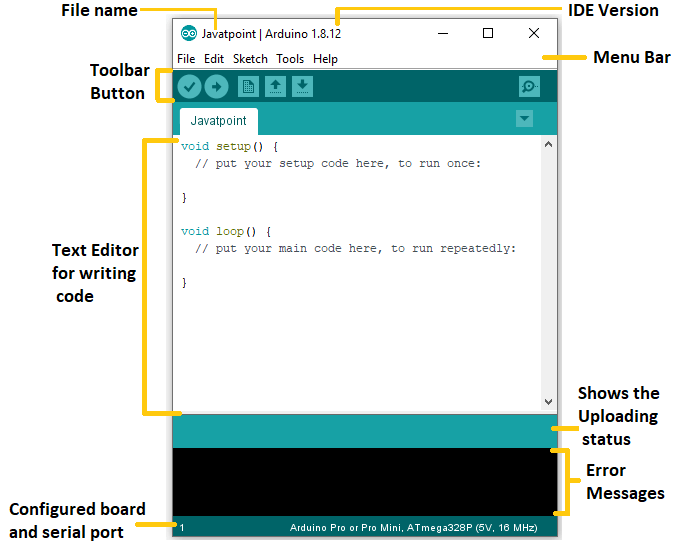


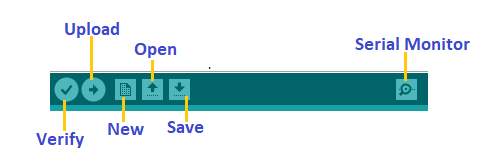
Fig 5. Arduino IDE Screen

Each section of the Arduino IDE display is detailed below.

**Toolbar Button**

The icons displayed on the toolbar are **New, Open, Save, Upload,** and **Verify**.

It is shown below:



**Upload**

The Upload button compiles and runs our code written on the screen. It further uploads the code to the connected board. Before uploading the sketch, we need to make sure that the correct board and ports are selected.

We also need a USB connection to connect the board and the computer. Once all the above measures are done, click on the Upload button present on the toolbar.

The latest Arduino boards can be reset automatically before beginning with Upload. In the older boards / certain ESP32 MCU, we need to press the Reset button present on it. As soon as the uploading is done successfully, we can notice the execution of the program example: The Blink of the LED.

If the uploading is failed, it will display the message in the error window.

We do not require any additional hardware to upload our sketch using the Arduino Bootloader.

A Bootloader is defined as a small program, which is loaded in the microcontroller present on the board.

**Open**

The Open button is used to open the already created file. The selected file will be opened in the current window.

**Save**

The save button is used to save the current sketch or code.

**New**

It is used to create a new sketch or opens a new window.

**Verify**

The Verify button is used to check the compilation error of the sketch or the written code.

**Serial Monitor**

The serial monitor button is present on the right corner of the toolbar. It opens the serial monitor.

It is shown below:

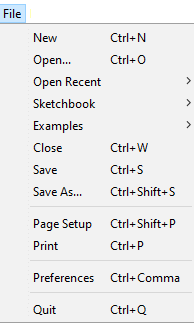
Arduino IDE

When we connect the serial monitor, the board will reset on the operating system Windows, Linux, and Mac OS X. If we want to process the control characters in our sketch, we need to use an external terminal program. The terminal program should be connected to the COM port, which will be assigned when we connect the board to the computer.

**Menu Bar in IDE**

**File Menu**

When we click on the File button on the Menu bar, a drop-down list will appear. It is shown below:



Let's discuss each option in detail.

**New**

The New button opens the new window. It does not remove the sketch which is already present.

**Open**

It allows opening the sketch, which can be browsed from the folders and computer drivers.

**Open Recent**

The Open Recent button contains the list of the recent sketches.

**Sketchbook**

It stores the current sketches created in the Arduino IDE software. It opens the selected sketch or code in a new editor at an instance.

**Examples**

It shows the different examples of small projects for a better understanding of the IDE and the board. The IDE provides examples of self-practice.

**Close**

The Close button closes the window from which the button is clicked.

**Save**

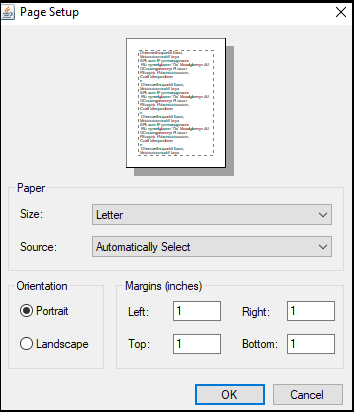
The save button is used to save the current sketch. It also saves the changes made to the current sketch. If we have not specified the name of the file, it will open the '**Save As...'** window.

**Save As**

We can save the sketch with a different name using the '**Save As...'** button. We can also change the name accordingly.

**Page Setup**

It allows setting the page margins, orientation, and size for printing. The '**Page Setup**' window will appear as:



**Print**

According to the settings specified in the 'Page Setup', it prepares the current sketch for printing.

**Preferences**

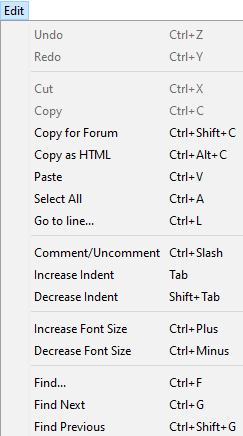
It allows the customization settings of the Arduino IDE.

**Quit**

The Quit button is used to close all the IDE windows. The same closed sketch will be reopened when we will open the Arduino IDE.

**Edit Menu**

When we click on the Edit button on the Menu bar, a drop-down list appears. It is shown below:



Let's discuss each option in detail.

**Undo**

The Undo button is used to reverse the last modification done to the sketch while editing.

**Redo**

The Redo button is used to repeat the last modification done to the sketch while editing.

**Cut**

It allows us to remove the selected text from the written code. The text is further placed to the clipboard. We can also paste that text anywhere in our sketch.

**Copy**

It creates a duplicate copy of the selected text. The text is further placed on the clipboard.

**Copy for Forum**

The 'Copy for Forum' button is used to copy the selected text to the clipboard, which is also suitable for posting to the forum.

**Copy as HTML**

The 'Copy for Forum' button is used to copy the selected text as HTML to the clipboard. It is desirable for embedding in web pages.

**Paste**

The Paste button is used to paste the selected text of the clipboard to the specified position of the cursor.

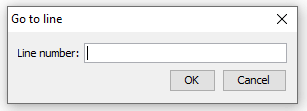
**Select All**

It selects all the text of the sketch.

**Go to line**

It moves the cursor to the specified line number.

The window will appear as:



**Comment/Decomment**

The Comment/ Decomment button is used to put or remove the comment mark (**//**) at the beginning of the specified line.

**Increase Indent**

It is used to add the space at the starting of the specified line. The spacing moves the text towards the right.

**Decrease Indent**

It is used to subtract or remove the space at the starting of the specified line. The spacing moves the text towards the left.

**Increase Font Size**

It increases the font size of the written text.

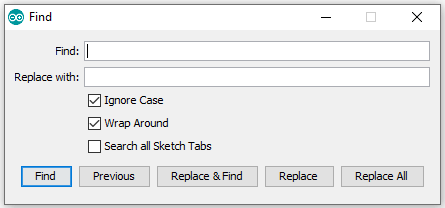
**Decrease Font Size**

It decreases the font size of the written text.

**Find**

It is used to find the specified text. We can also replace the text. It highlights the text in the sketch.

The window will appear as:



**Find Next**

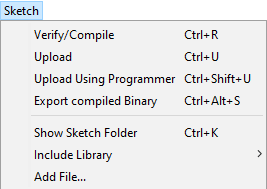
It highlights the next word, which has specified in the '**Find...'** window. If there is no such word, it will not show any highlighted text.

**Find Previous**

It highlights the previous word, which has specified in the '**Find'** window. If there is no such word, it will not show any highlighted text.

**Sketch Menu**

When we click on the Sketch button on the Menu bar, a drop-down list appears. It is shown below:



Let's discuss each option in detail.

**Verify/Compile**

It will check for the errors in the code while compiling. The memory in the console area is also reported by the IDE.

**Upload**

The Upload button is used to configure the code to the specified board through the port.

**Upload Using Programmer**

It is used to override the Bootloader that is present on the board. We can utilize the full capacity of the Flash memory using the '**Upload Using Programmer**' option. To implement this, we need to restore the Bootloader using the **Tools**-> **Burn Bootloader** option to upload it to the USB serial port.

**Export compiled Binary**

It allows saving a .**hex** file and can be kept archived. Using other tools, .hex file can also be sent to the board.

**Show Sketch Folder**

It opens the folder of the current code written or sketch.

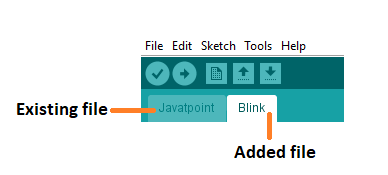
**Include Library**

Include Library includes various Arduino libraries. The libraries are inserted into our code at the beginning of the code starting with the #. We can also import the libraries from .zip file.

**Add File...**

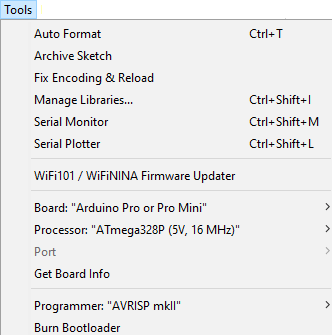
The Add File... button is used to add the created file in a new tab on the existing file.

For example, let's add '**Blink**' file to the '**Javatpoint**' file. The tab will now appear as:



**Tools Menu**

When we click on the Tools button on the Menu bar, a drop-down list appears. It is shown below:



Let's discuss each option in detail.

**Auto Format**

The Auto Format button is used to format the written code. For example, lining the open and closed curly brackets in the code.

**Archive Sketch**

The copy of the current sketch or code is archived in the .zip format. The directory of the archived is same as the sketch.

**Fix Encoding and Reload**

This button is used to fix the inconsistency between the operating system char maps and editor char map encoding.

**Manage Libraries...**

It shows the updated list of all the installed libraries. We can also use this option to install a new library into the Arduino IDE.

**Serial Monitor**

It allows the exchange of data with the connected board on the port.

**Serial Plotter**

The Serial Plotter button is used to display the serial data in a plot. It comes preinstalled in the Arduino IDE.

**WiFi101/WiFiNINA Firmware Updater**

It is used to check and update the Wi-Fi Firmware of the connected board.

**Board**

We are required to select the board from the list of boards. The selected board must be similar to the board connected to the computer.

**Processor**

It displays the processor according to the selected board. It refreshes every time during the selection of the board.

**Port**

It consists of the virtual and real serial devices present on our machine.

**Get Board Info**

It gives the information about the selected board. We need to select the appropriate port before getting information about the board.

**Programmer**

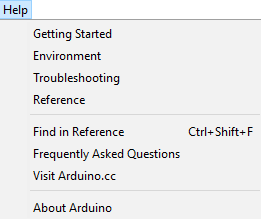
We need to select the hardware programmer while programming the board. It is required when we are not using the onboard USB serial connection. It is also required during the burning of the Bootloader.

**Burn Bootloader**

The Bootloader is present on the board onto the microcontroller. The option is useful when we have purchased the microcontroller without the bootloader. Before burning the bootloader, we need to make sure about the correct selected board and port.

**Help Menu**

When we click on the Help button on the Menu bar, a drop-down list will appear. It is shown below:



The Help section includes several documents that are easy to access, which comes along with the Arduino IDE. It consists of the number of options such as Getting Started, Environment, Troubleshooting, Reference, etc. can be accessed without the internet connection as well. It will directly link us to the official website of Arduino.

**4.2 Why Arduino IDE?**

1. Easy to use: Arduino IDE provides a user-friendly interface for writing, compiling, and uploading code.

2. Cross-platform: Available for Windows, macOS, and Linux.

3. Open source: Free to download, modify, and distribute.

4. Large community: Extensive libraries, tutorials, and forums.

5. Versatile: Supports various microcontrollers, including Arduino, ESP32, and ESP8266.

**4.3 Why Arduino IDE for ESP32/ESP8266?**

1. Simplified development: Arduino IDE streamlines the development process for ESP32/ESP8266.

2. Library support: Access to a vast library collection, including WiFi, Bluetooth, and IoT-specific libraries.

3. Community resources: Leverage ESP32/ESP8266-specific tutorials, examples, and forums.

4. Easy uploading: Upload code directly from Arduino IDE to ESP32/ESP8266 boards.

5. Cost-effective: No need for expensive development tools or software.

**ESP32-specific benefits**:

1. Dual-core processing: Arduino IDE supports ESP32's dual-core architecture.

2. Wi-Fi and Bluetooth: Easily integrate Wi-Fi and Bluetooth capabilities.

3. Low-power consumption: Optimize power usage with Arduino IDE's built-in features.

4. Advanced peripherals: Access ESP32's advanced peripherals, such as SPI, I2C, and UART.

**ESP8266-specific benefits:**

1. WiFi integration: Easily integrate WiFi capabilities.

2. Low-cost: ESP8266 is an affordable board.

3. Small footprint: Ideal for IoT projects requiring compact design.

4. Community support: Extensive ESP8266 community and resources.

**Comparison with other IDEs:**

1. Platform IO: More complex but offers advanced features and multi-platform support.

2. Visual Studio Code: More versatile but requires additional setup and configuration.

3. ESP-IDF: Official ESP32/ESP8266 IDE, but steeper learning curve.

Arduino IDE provides an ideal balance of ease, versatility, and community support, making it a popular choice for ESP32 and ESP8266 development.

**4.4 Step by Step Installation Procedure**

In this section, we will learn in easy steps, how to set up the Arduino IDE on our computer and prepare the board to receive the program via USB cable.

The Arduino IDE 2 is an open-source project. It is a big step from its sturdy predecessor, Arduino IDE 1.x, and comes with revamped UI, improved board & library manager, debugger, autocomplete feature and much more.

**Step 1** – Download and install the Arduino IDE 2 on your Windows. You can easily download the editor by visit the Arduino Software page: <https://www.arduino.cc/en/software>

**Requirements**

* Windows - Win 10 and newer, 64 bits
* Linux - 64 bits
* macOS - Version 10.15: "Catalina" or newer, 64 bits

**For windows installation**: In Arduino IDE 2.3.3 select Win 10 and newer, 64 bits

A screenshot of a computer

Description automatically generated

Fig 6. Arduino Software page

To install the Arduino IDE 2 on a Windows computer, simply run the file downloaded from the software page.

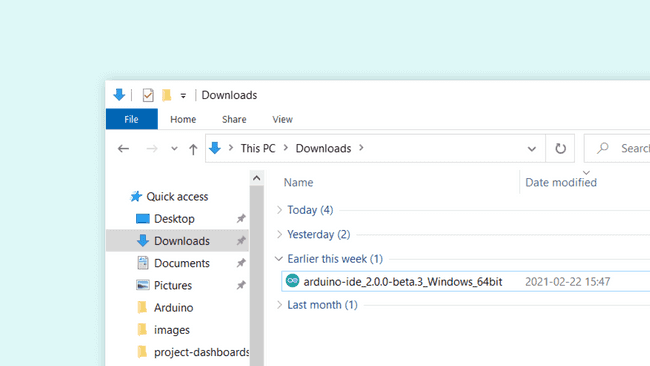
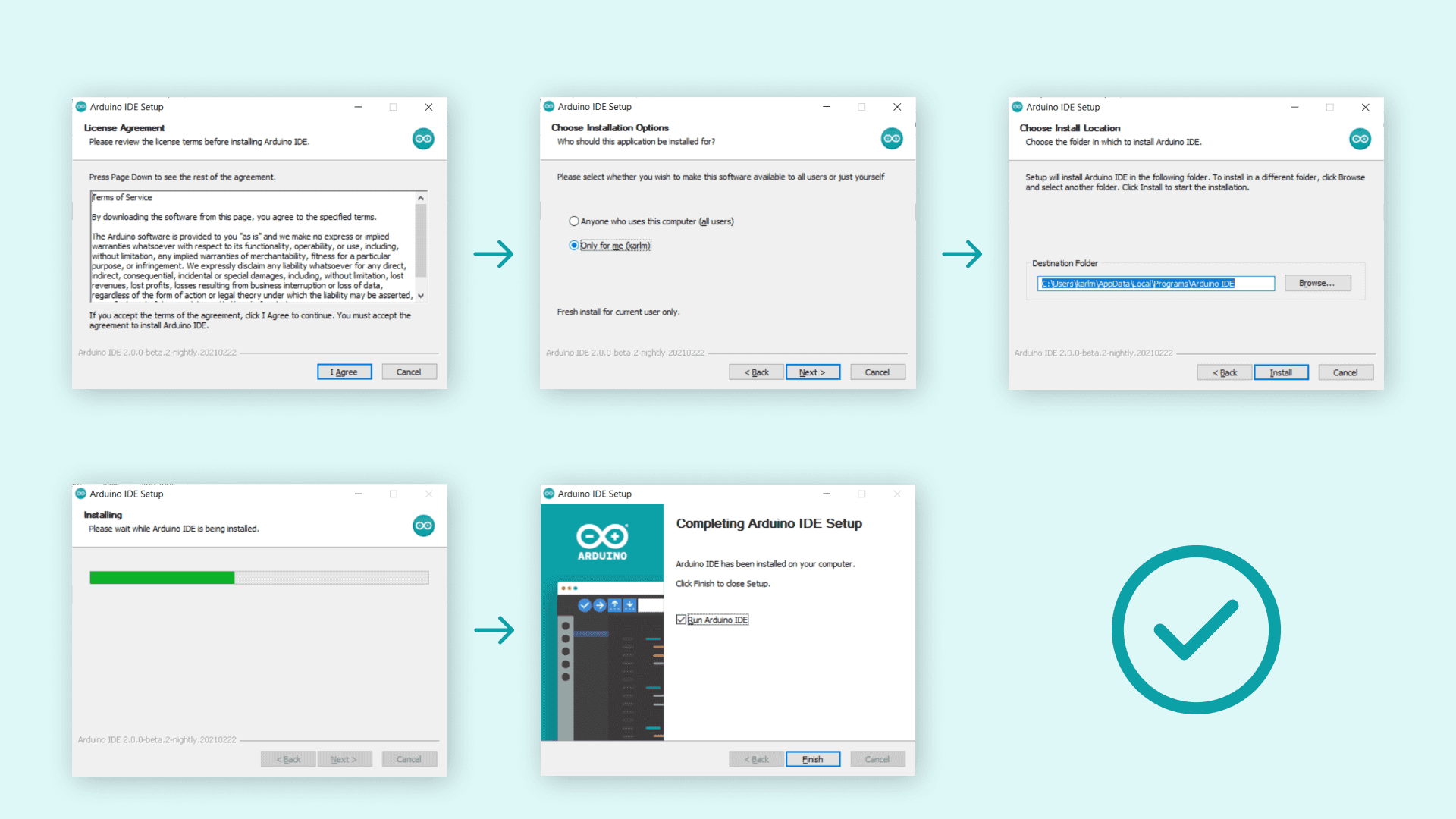


Fig 7. Running the Executable Arduino IDE 2 installation file

Follow the instructions in the installation guide as shown in the image. The installation may take several minutes.



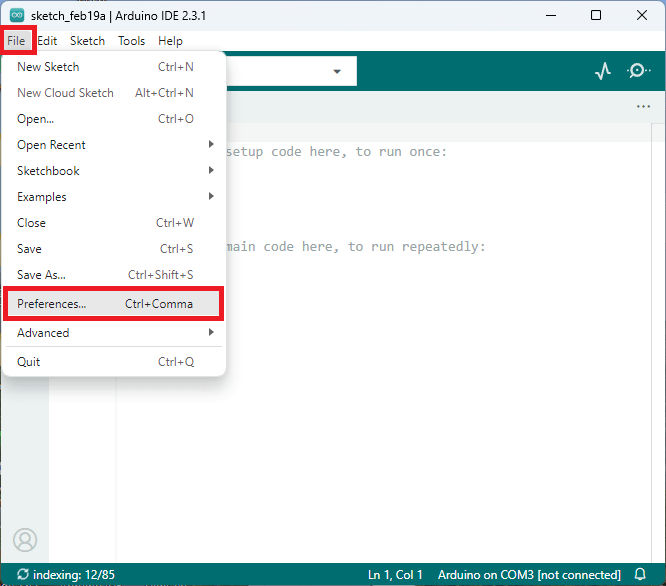
**Installing ESP32 Board in Arduino IDE 2**

Before proceeding make sure you have completed above process that is  [Arduino IDE 2](https://www.arduino.cc/en/Tutorial/getting-started-with-ide-v2" \t "_blank) installed on your computer and follow the instructions given below

**Install ESP32 Add-on in Arduino IDE**

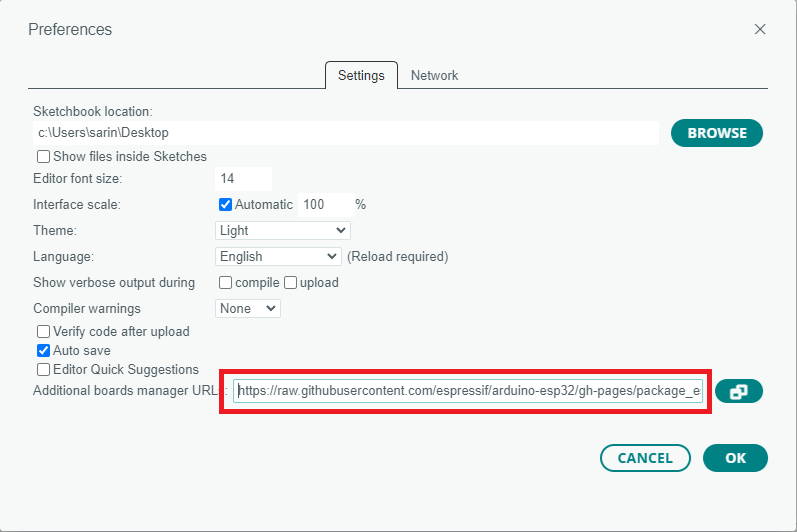
To install the ESP32 board in your Arduino IDE, follow these next instructions:

**1.**In your Arduino IDE 2, go to **File**> **Preferences**.



**2.** Copy and paste the following line to the **Additional Boards Manager** URLs field.

<https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json>



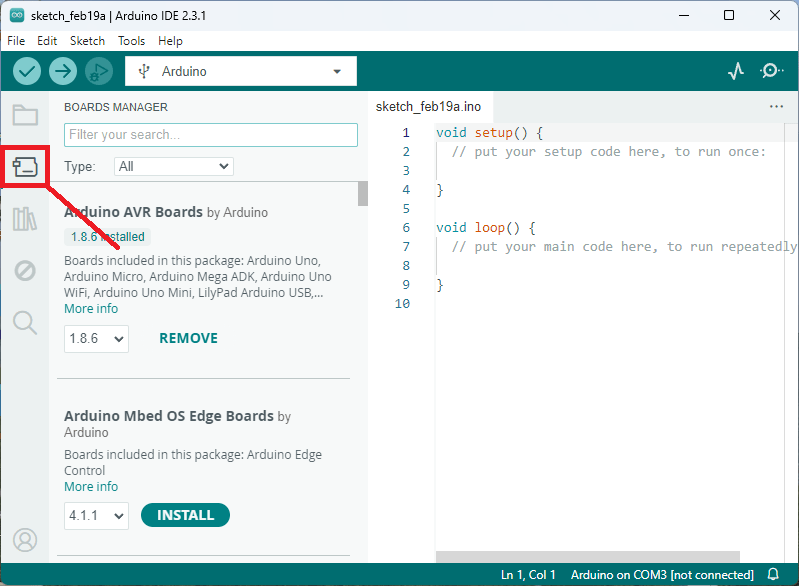
**Note:** if you already have the ESP8266 boards URL, you can separate the URLs with a comma, as follows:

<http://arduino.esp8266.com/stable/package_esp8266com_index.json>,

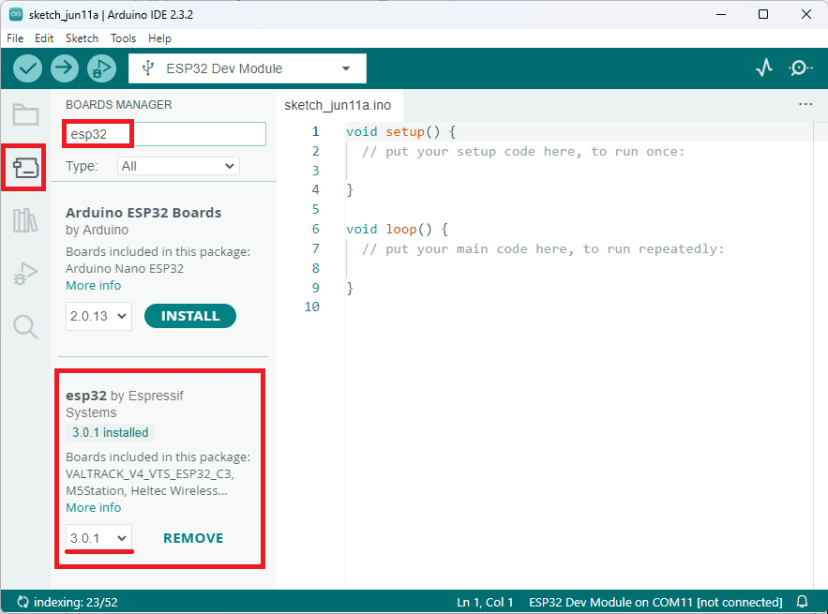
<https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json>

3. Open the Boards Manager. You can go to **Tools**> **Board**> **Boards Manager…** or you

can simply click the **Boards Manager**icon in the left-side corner.



4. Search for **ESP32**and press install button for **esp32 by Espressif Systems version 3.X**.



That’s it. It should be installed after a few seconds.

**Testing the Installation**

To test the ESP32 add-on installation, upload a simple code that blinks the on-board LED (GPIO 2).

Write the following code to your Arduino IDE:

A screenshot of a computer

Description automatically generated

**Uploading the Sketch**

Select your board before uploading the code. On the top drop-down menu, click on “**Select other board and port…**”

[A screen shot of a computer

Description automatically generated](https://i0.wp.com/randomnerdtutorials.com/wp-content/uploads/2024/02/arduino-ide-2-select-board.png?quality=100%26strip=all%26ssl=1)

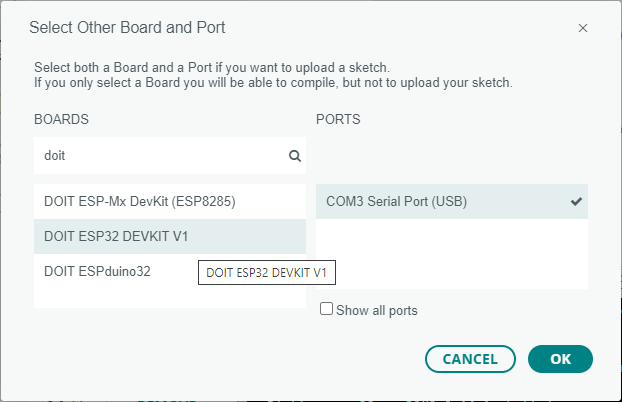
**Uploading the Sketch**

Select your board before uploading the code. On the top drop-down menu, click on “**Select other board and port…**”

[A screen shot of a computer

Description automatically generated](https://i0.wp.com/randomnerdtutorials.com/wp-content/uploads/2024/02/arduino-ide-2-select-board.png?quality=100%26strip=all%26ssl=1)

A new window, as shown below, will open. Search for your ESP32 board model.

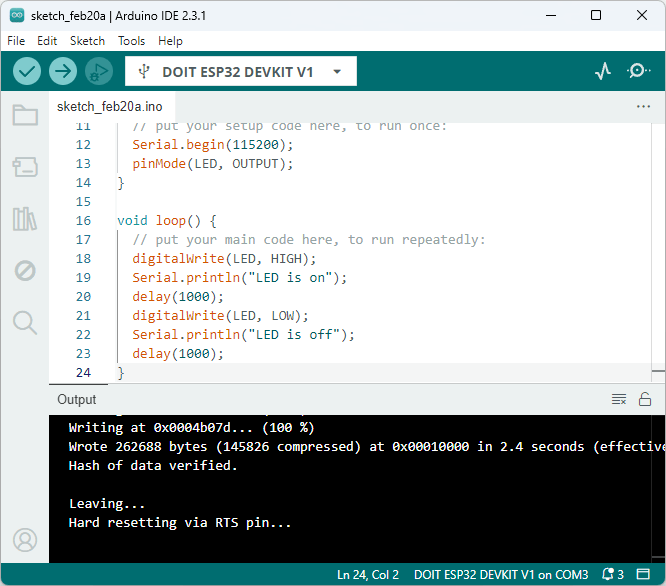
[](https://i0.wp.com/randomnerdtutorials.com/wp-content/uploads/2021/05/arduino-ide-2-select-board-esp32.png?quality=100%26strip=all%26ssl=1)

Select the ESP32 board model you’re using, and the COM port. In our example, we’re using the DOIT ESP32 DEVKIT V1. Click **OK** when you’re done.

Now, you just need to click on the **Upload**button shown below.

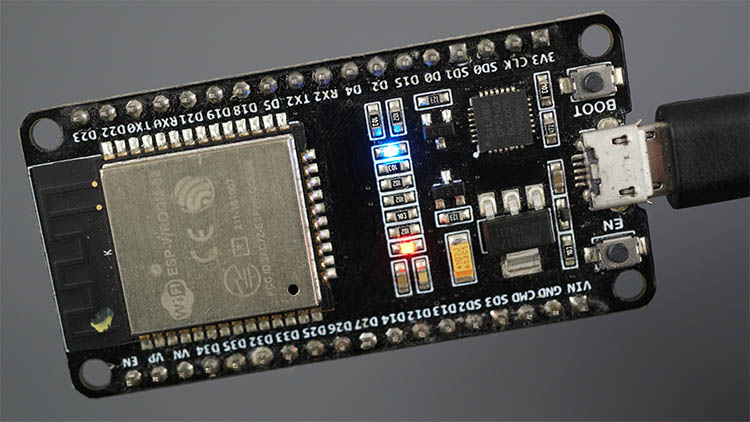
Arduino IDE 2 Upload Button

After a few seconds, the upload should be complete.



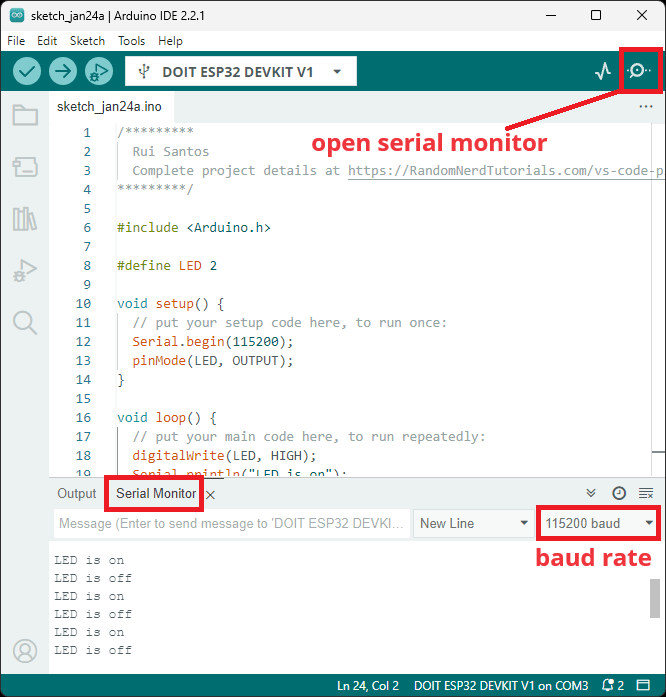
**Note**: some ESP32 development boards don’t go into flashing/uploading mode automatically when uploading a new code and you’ll see a lot of dots on the debugging window followed by an error message. If that’s the case, you need to press the ESP32 BOOT button when you start seeing the dots on the debugging window.

The ESP32 on-board LED should be blinking every second.

[](https://i0.wp.com/randomnerdtutorials.com/wp-content/uploads/2020/04/ESP32-board-Built_in-LED-turned-on-HIGH.jpg?quality=100%26strip=all%26ssl=1)

**Serial Monitor**

You can click on the Serial Monitor icon to open the Serial Monitor tab. Make sure you select the 115200 baud rate.



That’s it! You’ve installed the ESP32 Boards successfully in Arduino IDE 2.

**Troubleshooting**

If you try to upload a new sketch to your ESP32 and you get this error message “*A fatal error occurred: Failed to connect to ESP32: Timed out… Connecting…*”. It means that your ESP32 is not in flashing/uploading mode.

Having the right board name and COM port selected, follow these steps:

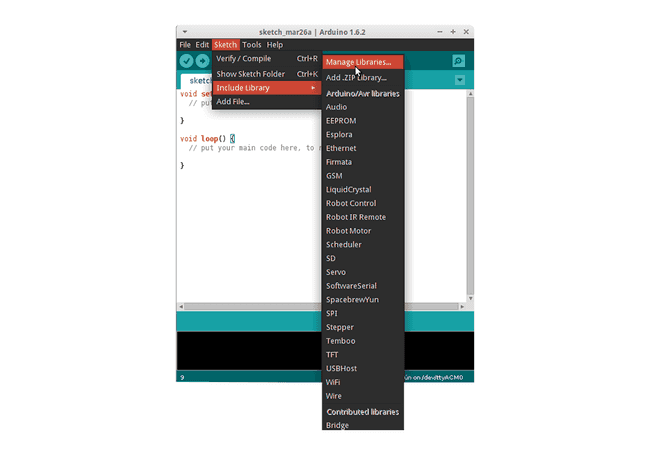
* Hold-down the **BOOT** button in your ESP32 board
* Press the **Upload**button in the Arduino IDE to upload your sketch
* After you see the “*Connecting….*” message in your Arduino IDE, release the finger from the **BOOT**button
* After that, you should see the “*Done uploading*” message

**Installing Libraries**

Libraries are a collection of code that makes it easy for you to connect to a sensor, display, module, etc. For example, the [LiquidCrystal library](https://www.arduino.cc/reference/en/libraries/liquidcrystal/) makes it easy to talk to character LCD displays. There are thousands of libraries available for download directly through the Arduino IDE, and you can find all of them listed at the [Arduino Library Reference](https://www.arduino.cc/reference/en/libraries/).

**Using the Library Manager**

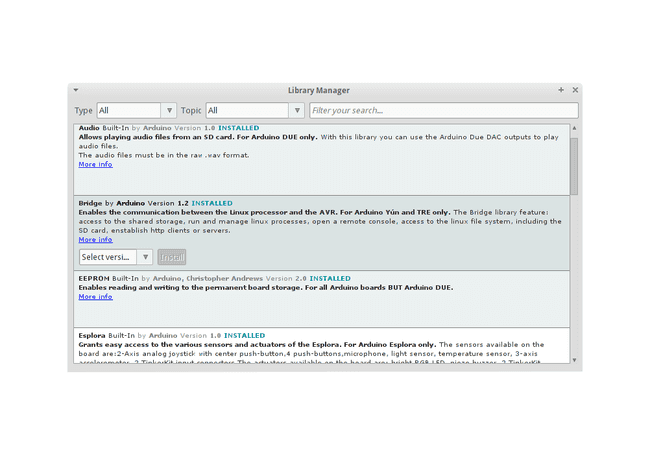
To install a new library into your Arduino IDE you can use the Library Manager (available from IDE version 1.6.2). Open the IDE and click to the "Sketch" menu and then *Include Library > Manage Libraries*.



Then the Library Manager will open, and you will find a list of libraries that are already installed or ready for installation. In this example we will install the Bridge library. Scroll the list to find it, click on it, then select the version of the library you want to install. Sometimes only one version of the library is available. If the version selection menu does not appear, don't worry it is normal.



Finally click on install and wait for the IDE to install the new library. Downloading may take time depending on your connection speed. Once it has finished, an *Installed* tag should appear next to the Bridge library. You can close the library manager.

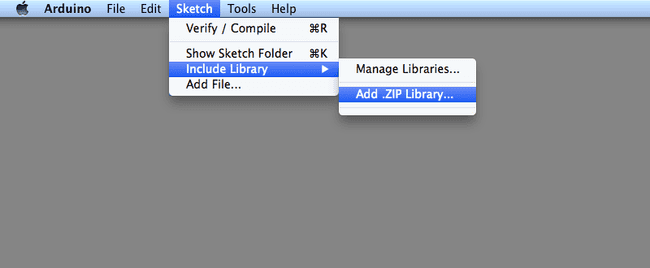


You can now find the new library available in the *Sketch > Include Library* menu. If you want to add your own library to Library Manager, follow [these instructions](https://github.com/arduino/library-registry" \l "adding-a-library-to-library-manager).

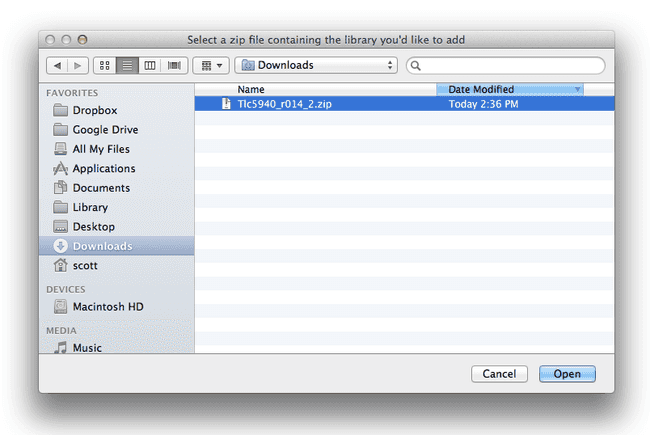
**Importing a .zip Library**

Libraries are often distributed as a ZIP file or folder. The name of the folder is the name of the library. Inside the folder will be a .cpp file, a .h file and often a keywords.txt file, examples folder, and other files required by the library. Starting with version 1.0.5, you can install 3rd party libraries in the IDE. Do not unzip the downloaded library, leave it as is.

In the Arduino IDE, navigate to *Sketch > Include Library > Add .ZIP Library*. At the top of the drop-down list, select the option to "Add .ZIP Library''.



You will be prompted to select the library you would like to add. Navigate to the .zip file's location and open it.



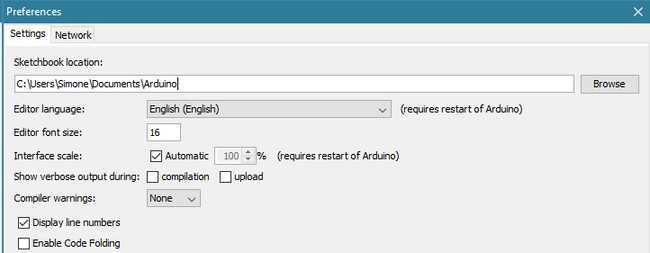
Return to the *Sketch > Include Library menu.* menu. You should now see the library at the bottom of the drop-down menu. It is ready to be used in your sketch. The zip file will have been expanded in the *libraries* folder in your Arduino sketches directory.

Note: the library will be available to use in sketches, but with older IDE versions examples for the library will not be exposed in the *File > Examples* until after the IDE has restarted.

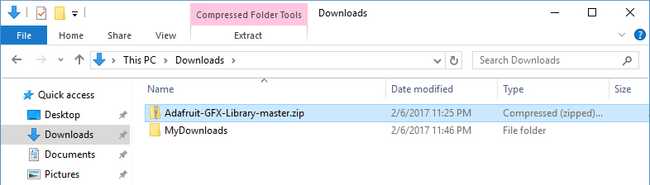
**Manual Installation**

When you want to add a library manually, you need to download it as a ZIP file, expand it and put in the proper directory. The ZIP file contains all you need, including usage examples if the author has provided them. The library manager is designed to install this ZIP file automatically as explained in the former chapter, but there are cases where you may want to perform the installation process manually and put the library in the *libraries* folder of your sketchbook by yourself.

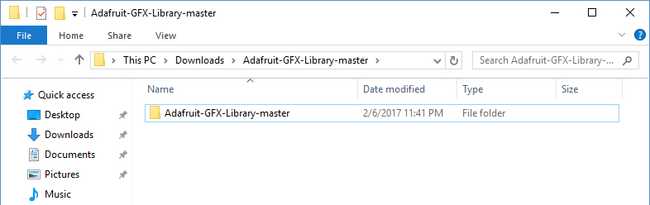
You can find or change the location of your sketchbook folder at *File > Preferences > Sketchbook* location.



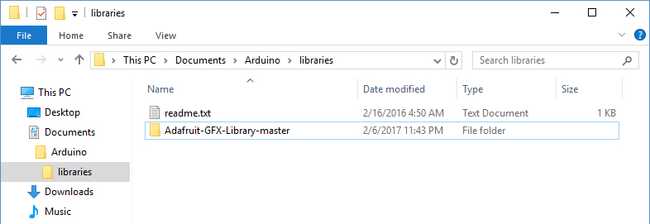
Go to the directory where you have downloaded the ZIP file of the library



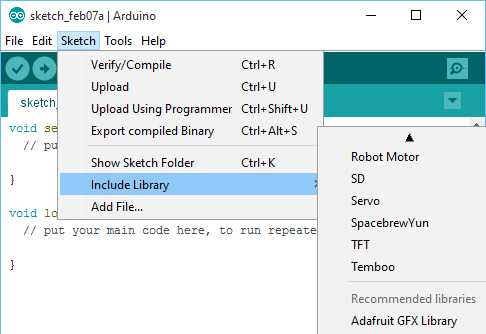
Extract the ZIP file with all its folder structure in a temporary folder, then select the main folder, that should have the library name



Copy it in the "libraries" folder inside your sketchbook.



Start the Arduino Software (IDE), go to *Sketch > Include Library*. Verify that the library you just added is available in the list.



Please note: Arduino libraries are managed in three different places: inside the IDE installation folder, inside the core folder and in the libraries folder inside your sketchbook. The way libraries are chosen during compilation is designed to allow the update of libraries present in the distribution. This means that placing a library in the "libraries" folder in your sketchbook overrides the other libraries versions.

**Blynk Library Installation**

To install a new library Blynk into your Arduino IDE, you can use the Library Manager.

First, connect your computer o the Internet. Open the IDE and click to the “Sketch” menu and then Include Library > Manage Libraries.

**A screenshot of a computer

Description automatically generated**

Then the Library Manager will open and you will find a list of libraries that are already installed or ready for installation. Search for Blynk library and in the version selection choose the latest version to date.

**A screenshot of a computer

Description automatically generated**

Finally click on Install and wait for the IDE to install the new library. Downloading may take time depending on your connection speed. Once it has finished, an Installed tag should appear next to the Bridge library. You can close the library manager.

**CHAPTER 5**

1. **IOT CLOUD & SERVER PLATFORM**

What is the internet of things (IoT)?

The internet of things, or IoT, is a network of interrelated devices that connect and exchange data with other IoT devices and the cloud. IoT devices are typically embedded with technology such as sensors and software and can include mechanical and digital machines and consumer objects.

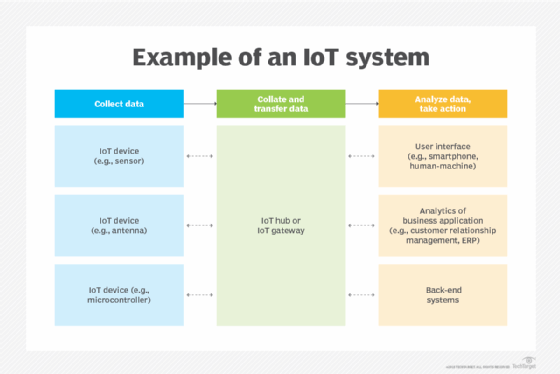
These devices encompass everything from everyday household items to complex industrial tools. Increasingly, organizations in a variety of industries are using IoT to operate more efficiently, deliver enhanced customer service, improve decision-making and increase the value of the business.

With IoT, data is transferable over a network without requiring human-to-human or human-to-computer interactions.

A thing in the internet of things can be a person with a heart monitor implant, a farm animal with a biochip transponder, an automobile that has built-in sensors to alert the driver when tire pressure is low, or any other natural or man-made object that can be assigned an Internet Protocol address and can transfer data over a network.

**How does IoT work?**

IoT systems function by gathering data from sensors embedded in IoT devices, which is then transmitted through an IoT gateway for analysis by an application or back-end system



The following four elements are incorporated into an IoT ecosystem for it to function:

**Sensors or devices**

An IoT ecosystem consists of web-enabled smart devices that use embedded systems, such as processors, sensors and communication hardware to collect, send and act on data acquired from their environments.

**Connectivity**

IoT devices can communicate with one another through a network over the internet. These devices share sensor data by connecting to an IoT gateway, which acts as a central hub where IoT devices can send data. Before the data is shared, it can also be sent to an edge device where it is analyzed locally.

**Data analysis**

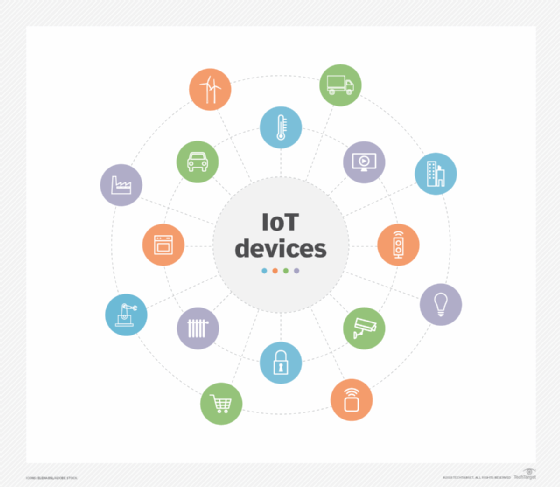
Only the relevant data is used to identify patterns, offer recommendations and identify potential issues before they escalate. Analyzing data locally reduces the volume of data sent to the cloud, which minimizes bandwidth consumption.

Sometimes, these devices communicate with other related devices and act on the information they get from one another. The devices do most of the work without human intervention, although people can interact with the devices. For example, they can set them up, give them instructions or access the data. The connectivity, networking and communication protocols used with these web-enabled devices largely depend on the specific IoT applications deployed.

IoT can also use artificial intelligence and machine learning to make data collection processes easier and more dynamic.

**Graphical user interface**

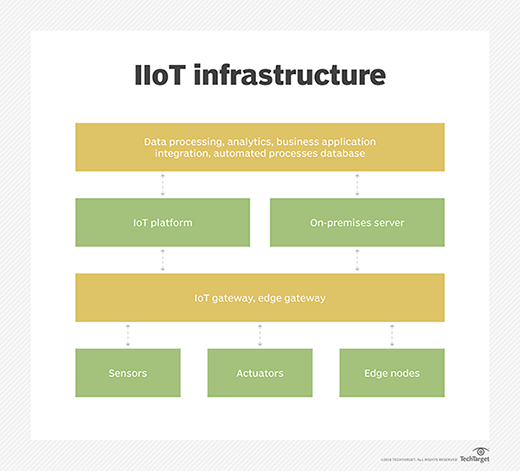
A graphical user interface ([UI](https://www.techtarget.com/searchapparchitecture/definition/user-interface-UI)) is typically used to manage IoT devices. For example, a website or a mobile app can be used as an UI to manage, control and register smart devices.



Consumer, enterprise and industrial IoT devices include smart TVs and smart sensors outfitted for conference rooms and assembly line machines.

**IoT applications include the following**:

* **Agriculture.**IoT can benefit farmers by making their jobs easier. For example, sensors can collect data on rainfall, humidity, temperature and soil content and IoT can help automate farming techniques. Additionally, IoT devices can be used to oversee the health of livestock, monitor equipment and streamline supply chain management.
* **Construction.** IoT can help monitor operations surrounding infrastructure. Sensors, for example, can monitor events or changes within structural buildings, bridges and other infrastructure that could potentially compromise safety. This provides benefits such as improved incident management and response, reduced operations costs and improved service quality.
* **Home automation.** A home automation business can use IoT to monitor and manipulate mechanical and electrical systems in a building. Homeowners can also remotely control and automate their home environment by using IoT devices, including smart thermostats, lighting systems, security cameras and voice assistants such as Alexa and Siri for increased comfort and energy efficiency.
* **Smart buildings and cities.**Smart cities can help citizens reduce waste and energy consumption. They can reduce energy costs using sensors that detect how many occupants are in a room and turning the air conditioner on if sensors detect a conference room is full or lowering the heat if everyone in the office has gone home.
* **Urban consumption systems.** IoT technologies can also be used to monitor and manage urban consumption such as traffic lights, parking meters, waste management systems and public transportation networks.
* **Healthcare monitoring.**IoT devices such as remote patient monitoring systems, smart medical devices and medication trackers let healthcare providers monitor patients' health status, manage chronic conditions and provide timely interventions. IoT gives providers the ability to monitor patients more closely by analyzing the generated data. Hospitals also often use IoT systems to complete tasks such as inventory management for both pharmaceuticals and medical instruments.
* **Retail.** IoT sensors and beacons in retail stores can track customer movement, analyze shopping patterns, manage inventory levels and personalize marketing messages. This enhances the shopping experience for customers and optimizes store operations.
* **Transportation.** IoT devices help the transportation industry by monitoring vehicle performance, optimizing routes and tracking shipments. For example, the fuel efficiency of connected cars can be monitored to reduce fuel costs and improve sustainability. IoT devices can also monitor the condition of cargo, so it reaches its destination in optimal condition.
* **Wearable devices.** Wearable devices with sensors and software can collect and analyze user data, sending messages to other technologies about the users to make their lives easier and more comfortable. Wearable devices are also used for public safety -- for example, by improving first responders' response times during emergencies by providing optimized routes to a location or by tracking construction workers' or firefighters' vital signs at life-threatening sites.
* **Energy management.** IoT-enabled smart grids, smart meters and energy management systems let utility companies and consumers monitor and optimize energy usage, manage demand-response programs and integrate renewable energy sources more efficiently. For example, the data collected by the IoT devices and sensors helps identify patterns, peak usage times and areas of inefficiency.



**What Is IoT Cloud?**

An IoT cloud is a massive network that supports IoT devices and applications. This includes the underlying infrastructure, servers and storage, needed for real-time operations and processing. An IoT cloud also includes the services and standards necessary for connecting, managing, and securing different IoT devices and applications.

**Why IoT Cloud?**

IoT clouds offer an efficient, flexible, and scalable model for delivering the infrastructure and services needed to power IoT devices and applications for businesses with limited resources. IoT clouds offer on-demand, cost-efficient hyperscale so organizations can leverage the significant potential of IoT without having to build the underlying infrastructure and services from scratch.

**Servers**

Servers in IoT connect physical devices to the internet, allowing them to communicate and share data. They can be used to monitor and control devices remotely, and to allow devices to interact with each other autonomously.

Here are some types of servers used in IoT:

* **IoT Data Server**

A data integration controller that can be used in factories, production lines, and cloud systems. It has standard data management functions for collection, processing, saving, noticing, and publishing.

* **IoT web server**

A platform that connects devices to the internet, allowing them to communicate and share data. It can be used to monitor and control devices remotely, and to allow devices to interact with each other autonomously.

* **Windows Server IoT**

A full version of Windows Server that provides enterprise manageability and security to IoT solutions.

* **Embedded web server**

A small, lightweight web server that can be integrated into other software or hardware systems. It's used to provide a web-based interface for monitoring and controlling devices.

* **Network server**

A server that plays a crucial role in IoT ecosystems by managing network resources and device communications. It allows IoT networks to scale up to support thousands or even millions of devices.

**Message Queue Telemetry Transport Protocol (MQTT)**

Message Queuing Telemetry Transport, or MQTT, is a communications protocol designed for Internet of Things devices with extremely high latency and restricted low bandwidth. Message Queuing Telemetry Transport is a perfect protocol for machine-to-machine (M2M) communication since it is designed specifically for low-bandwidth, high-latency settings.

**What is Message Queue Telemetry Transport Protocol (MQTT)?**

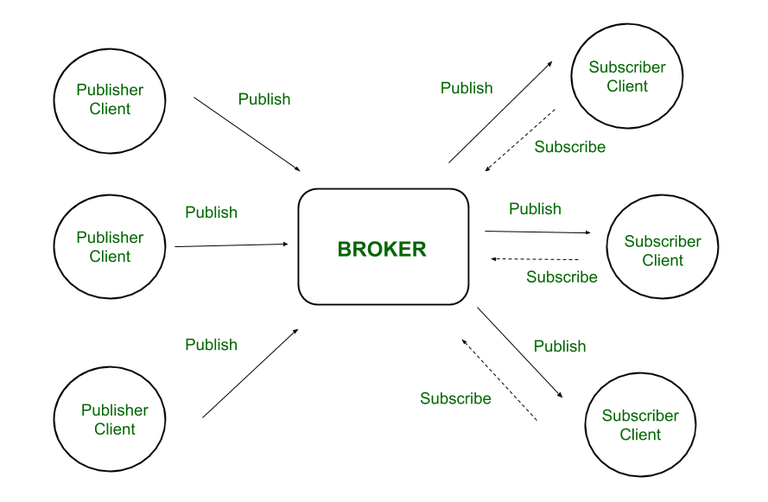
**MQTT** is a simple, lightweight messaging protocol used to establish communication between multiple devices. It is a TCP-based protocol relying on the publish-subscribe model. This communication protocol is suitable for transmitting data between resource-constrained devices having low bandwidth and low power requirements. Hence this messaging protocol is widely used for communication in the IoT Framework.

**Publish-Subscribe Model**

This model involves multiple clients interacting with each other, without having any direct connection established between them. All clients communicate with other clients only via a third party known as a Broker.

**MQTT Client and Broker**

Clients publish messages on different topics to brokers. The broker is the central server that receives these messages and filters them based on their topics. It then sends these messages to respective clients that have subscribed to those different topics. The heart of any publish/subscribe protocol is the MQTT broker. A broker can handle up to thousands of concurrently connected MQTT customers, depending on how it is implemented. All communications must be received by the broker, who will then sort them, ascertain who subscribed to each one, and deliver the messages to the clients who have subscribed. All persistent customers’ sessions, including missed messages and subscriptions, are likewise kept by the Broker.



*Fig. Publish-Subscribe Model*

Hence client that has subscribed to a specific topic receives all messages published on that topic.

Here the broker is central hub that receives messages, filters them, and distributes them to appropriate clients, such that both message publishers, as well as subscribers, are clients.

**Working of MQTT**

MQTT’s publish/subscribe (pub/sub) communication style, which aims to maximise available bandwidth, is an alternative to conventional client-server architecture that communicates directly with an endpoint. In contrast, the client who transmits the message (the publisher) and the client or clients who receive it (the subscribers) are not connected in the pub/sub paradigm. Third parties—the brokers—manage the relationships between the publishers and subscribers because they don’t communicate with one another directly.

Publishers and subscribers, which denote whether a client is publishing messages or has subscribed to receive messages, are examples of MQTT clients. The same MQTT client can be used to accomplish these two features. A publish occurs when a client or device want to submit data to a server or broker.

The term “subscribe” refers to the reversal of the procedure. Several clients can connect to a broker under the pub/sub paradigm and subscribe to subjects that interest them.

A screenshot of a computer

Description automatically generated

When a broker and a subscribing client lose contact, the broker will store messages in a buffer and send them to the subscriber whenever the broker is back up and running. The broker has the right to cut off communication with subscribers and send them a cached message containing publisher instructions if the publishing client abruptly disconnects from the broker.

“Publishers send the messages, subscribers receive the messages they are interested in, and brokers pass the messages from the publishers to the subscribers,” reads an IBM write-up describing the pub/sub paradigm. MQTT clients, such as publishers and subscribers, can only speak with MQTT brokers. Any device or programme that runs a MQTT library can be a MQTT client, ranging from microcontrollers like the Arduino to entire application servers housed in the cloud.

**Characteristics of MQTT**

* **Lightweight:**MQTT is designed to be lightweight, making it suitable for use in aid-restrained environments inclusive of embedded systems and low-strength devices. The protocol minimizes bandwidth and processing overhead, enabling green communication even on restricted networks.
* **Publish-Subscribe Model:**In the publish-subscribe version, clients (publishers) send messages to subjects, and different clients (subscribers) acquire messages from subjects of interest. This decoupling of producers and purchasers permits for flexible and dynamic conversation styles.
* **Quality of Service (QoS) Levels:**MQTT supports exclusive stages of message delivery warranty, referred to as Quality of Service (QoS). QoS levels range from 0 to 2, providing various stages of reliability and message transport guarantees, relying at the utility necessities.
* **Retained Messages:**MQTT lets in agents to store retained messages on topics, making sure that new subscribers acquire the maximum latest message posted on a subject right now after subscribing. This characteristic is beneficial for fame updates and configuration settings.
* **Last Will and Testament (LWT):** MQTT clients can specify a Last Will and Testament message to be posted by way of the broker in the occasion of an sudden consumer disconnect. This function affords a mechanism for detecting patron failures and dealing with them gracefully.
* **Security:**MQTT helps various protection mechanisms, consisting of Transport Layer Security (TLS) encryption and authentication mechanisms which include username/password and consumer certificates. These capabilities make certain the confidentiality, integrity, and authenticity of messages exchanged over MQTT connections.

**Advantages of MQTT**

This model is not restricted to one-to-one communication between clients. Although the publisher client sends a single message on specific topic, broker sends multiple messages to all different clients subscribed to that topic. Similarly, messages sent by multiple such publisher clients on multiple different topics will be sent to all multiple clients subscribed to those topics. Hence one-to-many, many-to-one, as well as many-to-many communication is possible using this model. Also, clients can publish data and at the same time receive data due to this two-way communication protocol. Hence MQTT is considered to be bi-directional protocol. The default unencrypted MQTT port used for data transmission is 1883. The encrypted port for secure transmission is 8883.

* Lightweight protocol that is quick to create and allows for efficient data transport
* Minimal data packet usage, resulting in low network usage
* Effective data dispersion
* The effective use of remote sensing and control
* Prompt and effective message delivery
* Minimizes power consumption, which is beneficial for the linked devices, and maximizes network capacity.
* Data transmission is quick, efficient, and lightweight because MQTT messages have small code footprint. These control messages have a fixed header of size 2 bytes and payload message up to size 256 megabytes.

**Introduction about Blynk IoT**

Blynk is a software platform that allows users to connect their hardware to the cloud and manage connected devices remotely. Blynk is designed for the Internet of Things (IoT) and can be used for personal projects or commercial products.

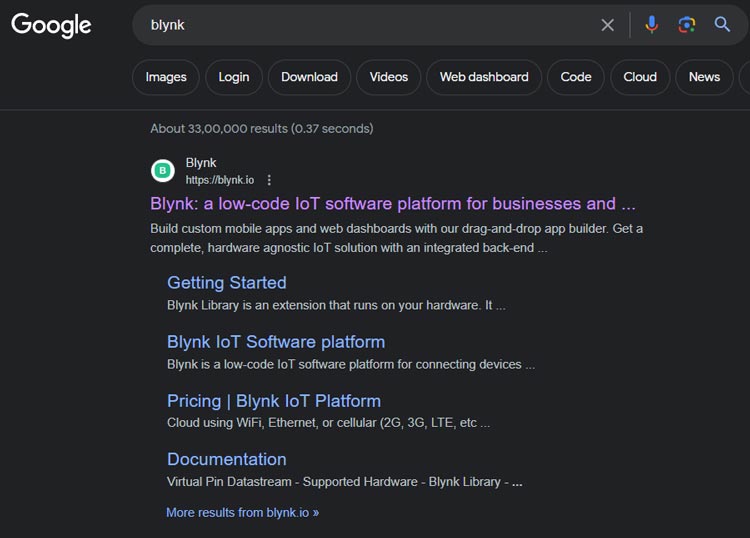
Here are some features of Blynk:

* No-code approach: Blynk allows users to build and scale IoT without coding. For example, users can control an LED or motor from their phone with just a few clicks.
* Remote management: Users can remotely control devices from anywhere, receive notifications, and analyze data.
* Multi-tenancy: Users can configure roles and permissions to control who has access to data.
* Scalability: Blynk can handle thousands of devices and is used by businesses of all sizes.
* Open-source: Blynk's server is open-source and can be run locally or on Blynk Cloud.
* Blynk App: Users can create interfaces for their projects using widgets.
* Blynk Libraries: Users can communicate with the server and process commands for popular hardware platforms.

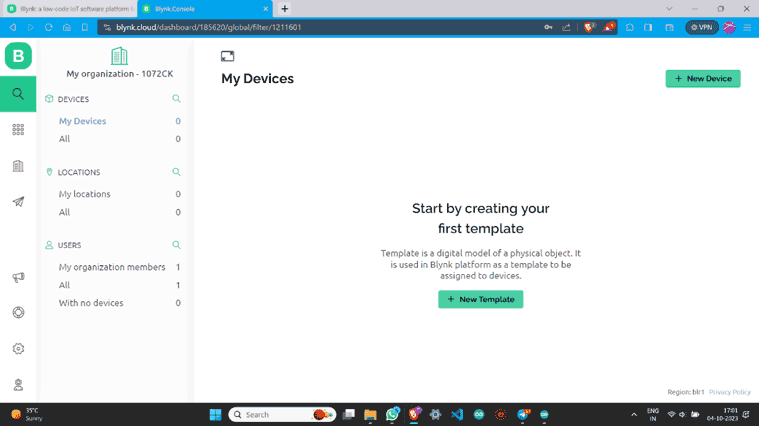
Blynk is free for personal use and prototyping. Businesses can purchase subscriptions to publish Blynk-powered apps for their products.

**Setup Blynk 2.0 App**

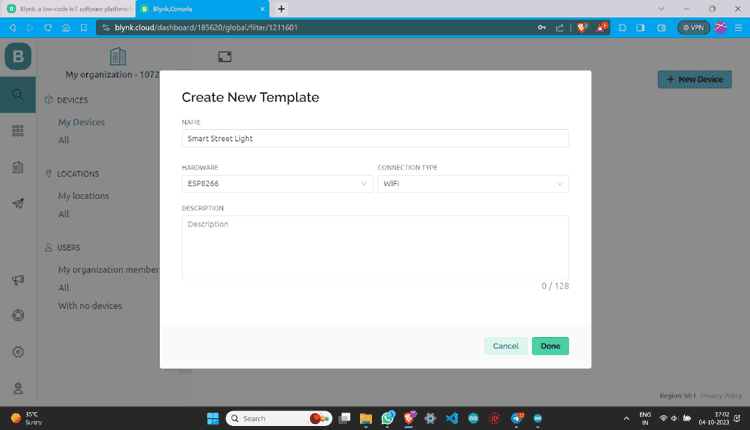
Step 1: Go to the Blynk website and click the "Sign Up" button. Enter your email address and create a password.

****

Step 2: Once you have signed up on the Blynk and logged in, tap the "+ New Template" button to create a new template.

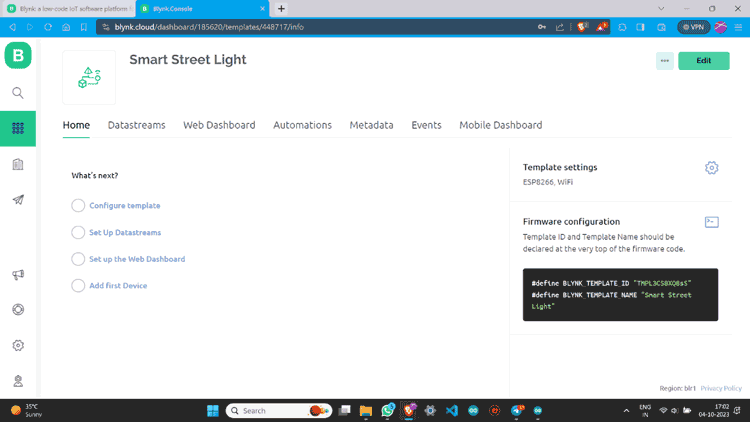
****

Step 3: Name the template, select the Development Board, set WiFi as the connection type and then save this template.

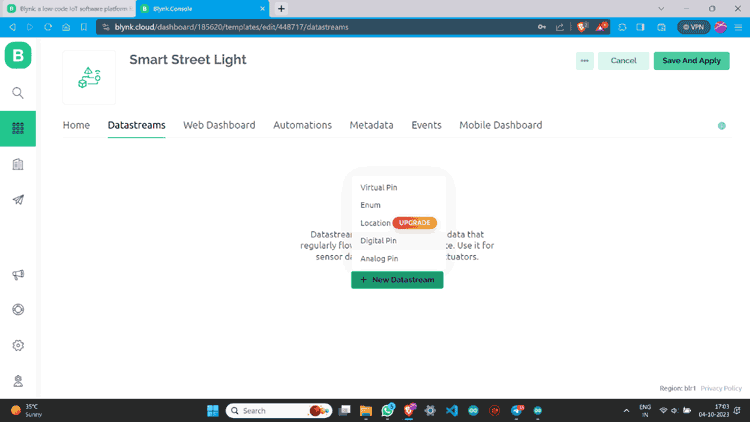
****

Step 4:  Once a new template is created you will see the below screen on your blynk account

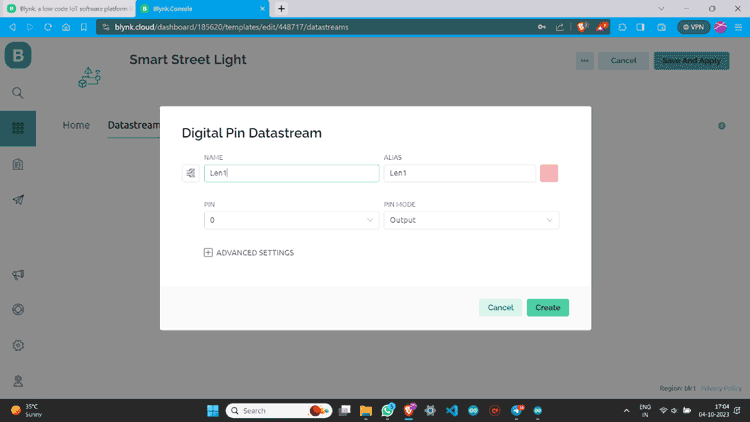
.

****

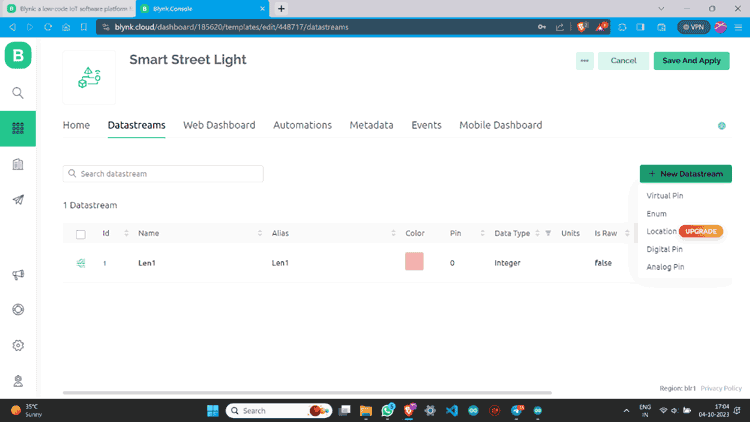
Step 5:  Go to Datastreams and click on “+ New Datastream”. Then Choose “Digital Pin”.

****

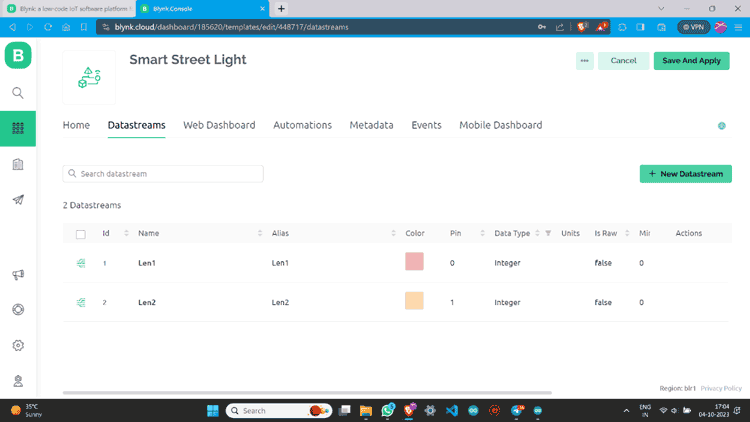
Step 6: Name Datastream and select pin. Assign it’s PIN MODE as Output and Create.

****

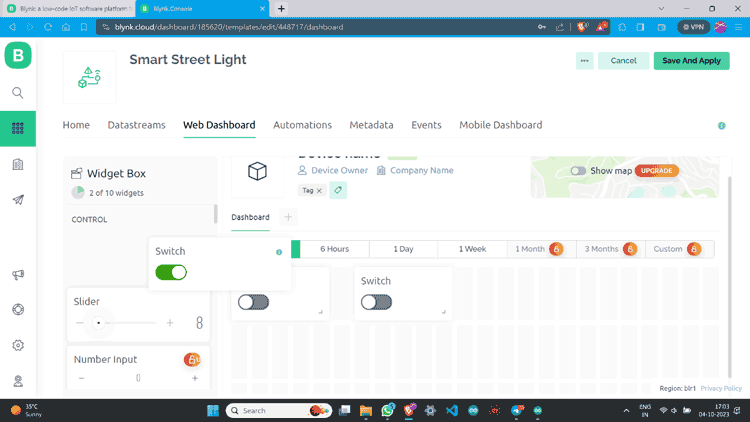
Step 7: Once a datastream is created on the blynk website, it will look something like this below

****

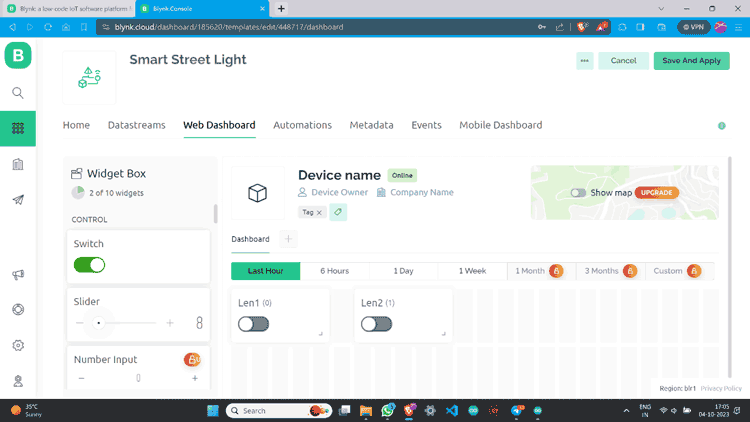
Step 8: I have made one more data stream with the same values.

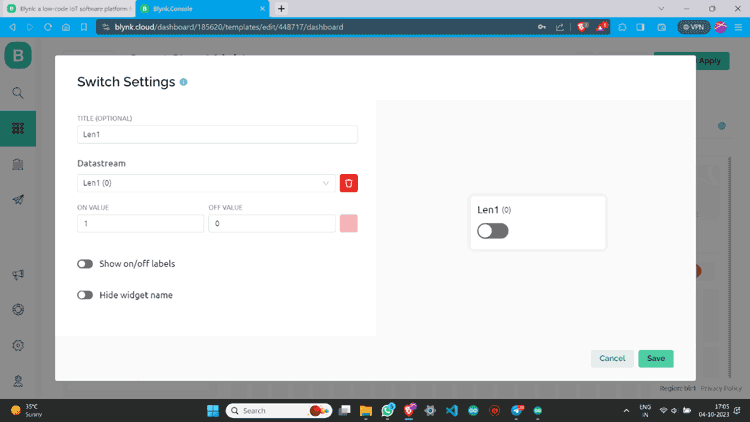
****

Step 9: Go to “Web Dashboard” and add Switch widgets.

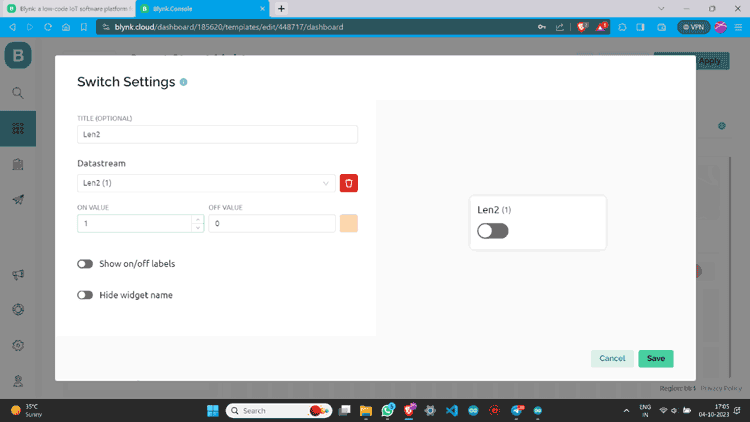
****

Step 10: Click on the Gear button, give values according to your choice, and save.

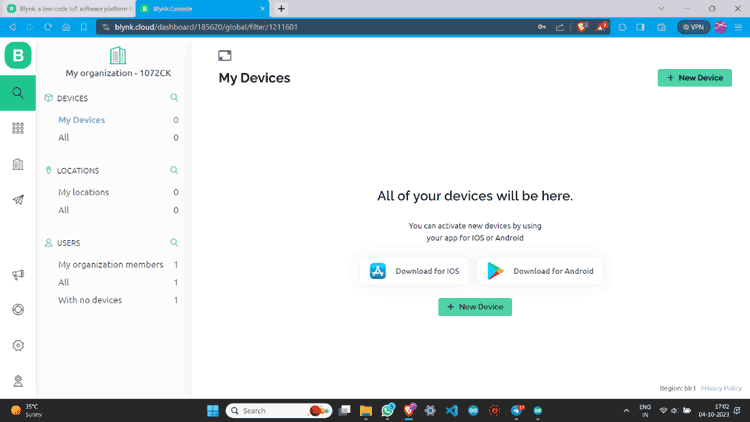
****

****

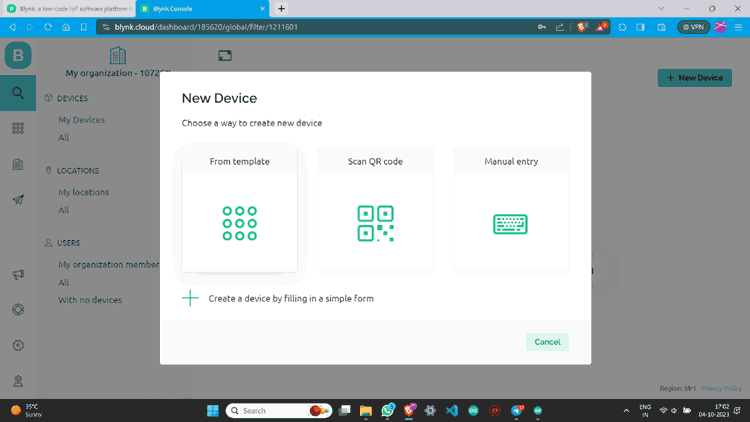
Step 11: I have made one more switch and assigned all values.

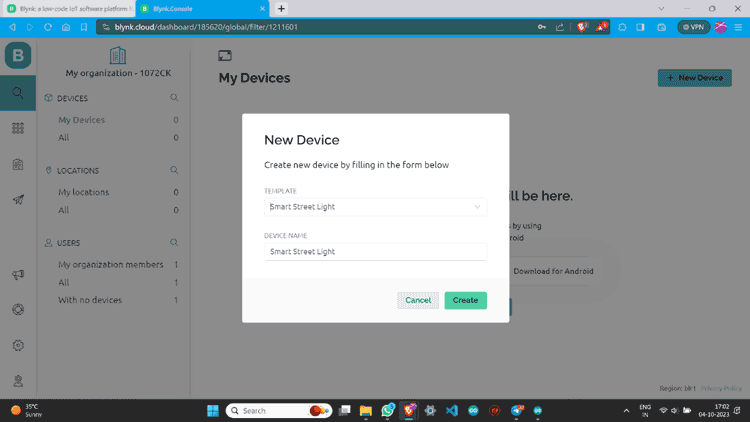
****

Step 12: Go to the lens icon and click on “+ New Device”.

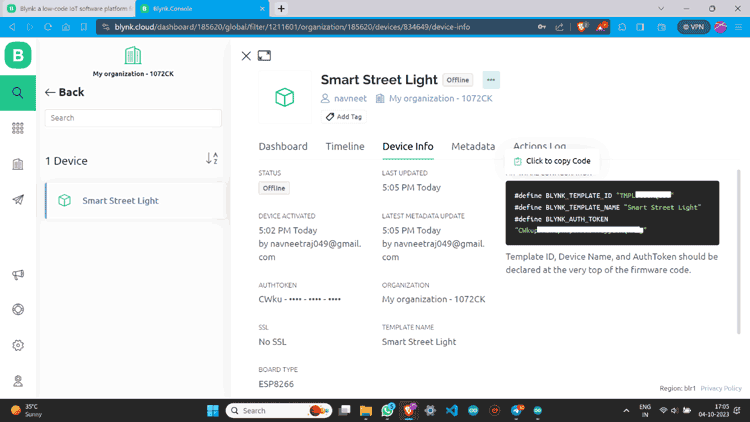
****

Step 13: Click “From template”, select the template, and create.

****

****

Step 14: Now you can see the device. Click on “Device Info” and copy the Auth Token.

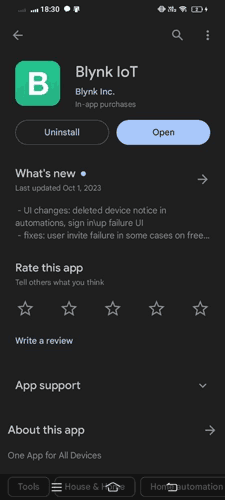
****

**Blynk 2.0 Application Setup**

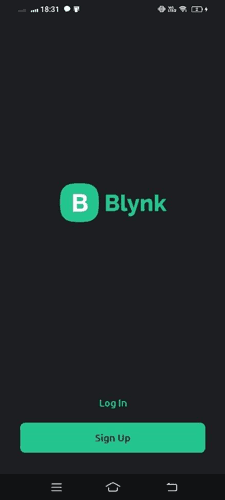
Now after Web setup, we have to make a Mobile App Interface so that we can able to control it from any Android or IOS

Step 1:  Download the Blynk app.

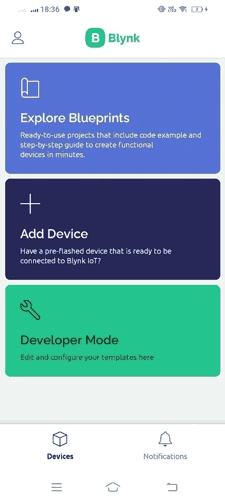
The Blynk app is available for both IOS and Android devices. You can download the app from the App Store or Google Play.

****

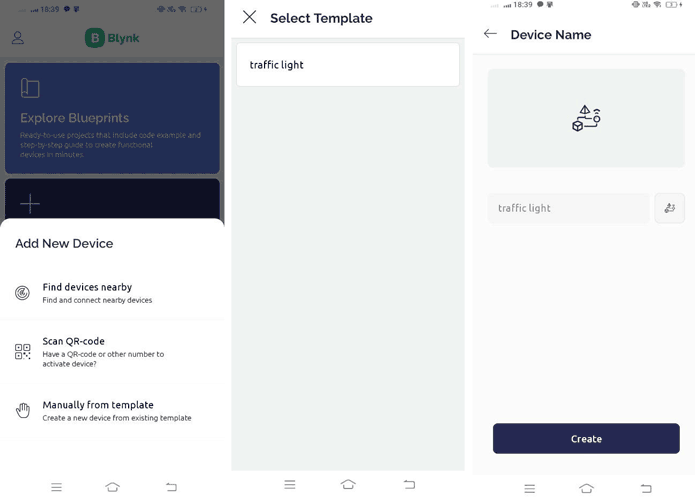
Step 2: Now open the Blynk IOT app and Log in with the credentials.

****

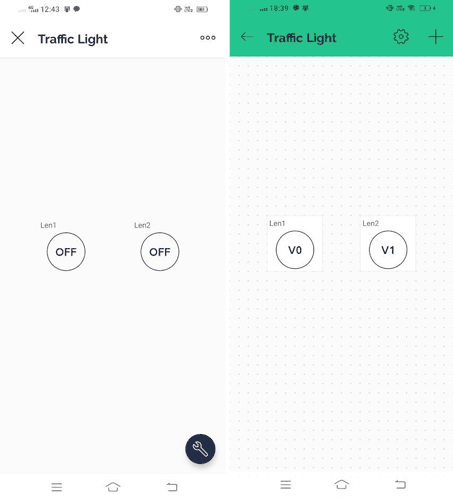
Step 3: After logging in click on “Add Device”.

****

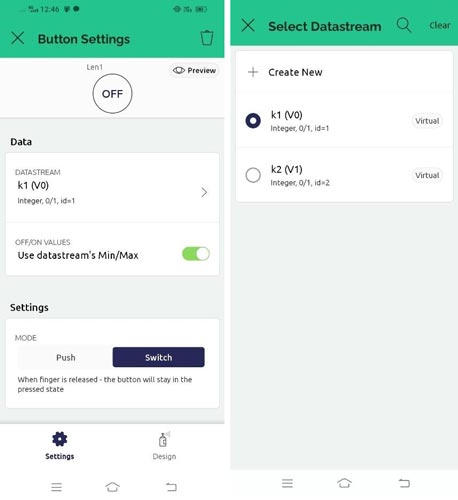
Step 4: Click “Manually from template”, select the device we already created, and then create.

****

Step 5: Now you are on the dashboard of the device then click on the tool button to enter in Developer Mode.

****

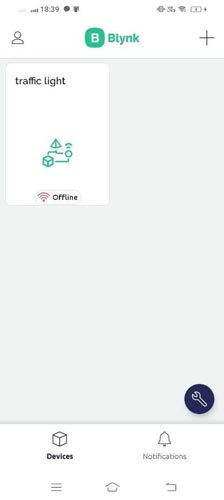
Step 6: Click on the button icons to assign values like Datastream and Mode. You can also change its design and looks.

****

Step 7: You Have successfully created a device dashboard.

****

Step 8: You can check the device status also.

****

**Setup Arduino IDE to NodeMCU ESP8266 board to use Blynk**

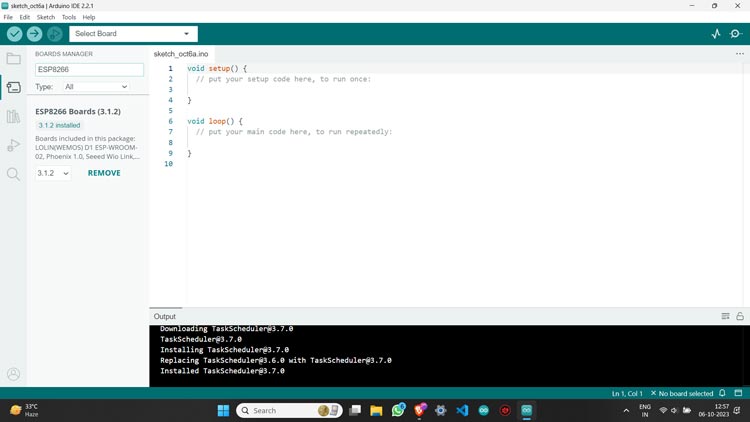
Now we have to set our Arduino IDE to program the NodeMCU ESP8266 board.

Step 1: In Arduino IDE go to the “BOARDS MANAGER” and install ESP8266.

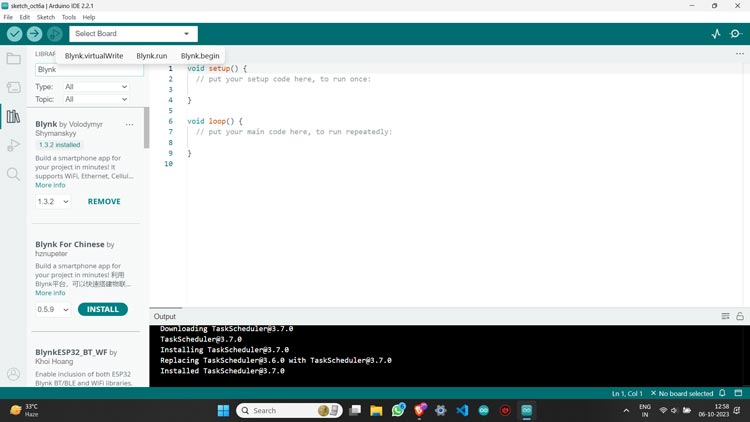
**Install Blynk Library in Arduino IDE**

to set our **Arduino IDE** to program the **NodeMCU ESP8266 board**.

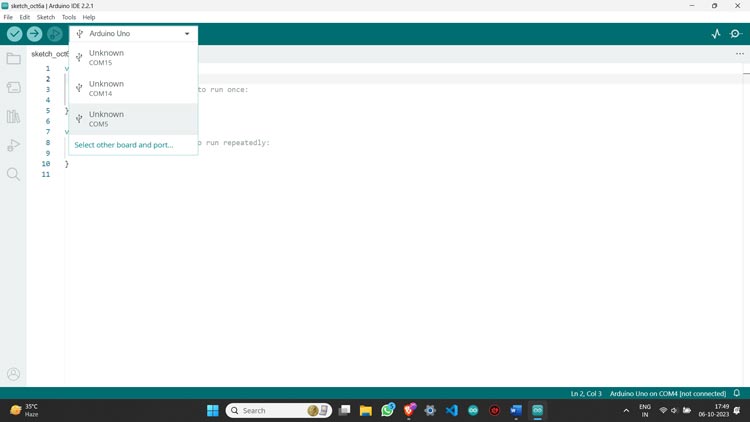
Step 1: In Arduino IDE go to the “**BOARDS MANAGER**” and install **ESP8266**.

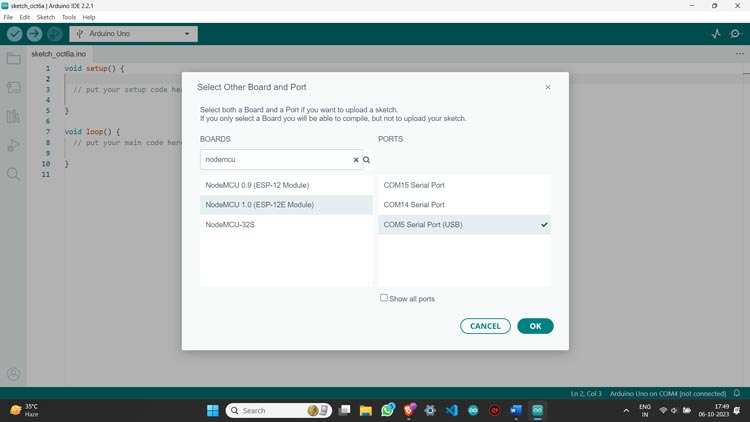


Step 2: Then go to the “**LIBRARY MANAGER**” and install the **Blynk library**.



Step 3: Plug your NodeMCU ESP8266 board into the PC using a **USB cable** and then **select your board and port**.





Step 4: Your Arduino IDE is now **ready**to program your board for using Blynk app

**CHAPTER 6:**

**6. WORKING PRINCIPAL OF IDEATION**

**6.1 Tell about Project Working Processes:**

Working Process

The working process of the IoT-based broken line fault detection system involves several key stages, from design to deployment. Below is a detailed breakdown of each step:

1. System Design

Requirement Analysis: Identify specific needs and constraints of the system, including environmental factors and expected fault scenarios.

Architecture Development: Design the overall architecture, including sensor placements, data flow paths, and system components (sensors, microcontroller, cloud server, and user interface).

2. Hardware Integration

Sensor Selection: Choose appropriate sensors (current, voltage, vibration) based on monitoring requirements and environmental conditions.

Microcontroller Setup: Integrate a microcontroller (e.g., ESP8266 or ESP32) to handle sensor data and facilitate communication with the cloud server.

Circuit Design: Create the necessary circuitry to connect sensors to the microcontroller, ensuring proper power supply and data transmission capabilities.

3. Software Development

Data Acquisition: Develop firmware for the microcontroller to read sensor data at regular intervals and handle error conditions.

Cloud Integration: Set up a cloud server (e.g., AWS, Google Cloud) for data storage and processing. Implement APIs to receive data from the microcontroller.

Machine Learning Implementation: Develop and train machine learning algorithms for anomaly detection. The models will analyze incoming data to identify potential faults based on historical data and predefined thresholds.

4. User Interface Development

Mobile and Web Application: Create a user-friendly interface that displays real-time data, alerts, and historical trends. The application should enable users to configure settings and view system status.

Notification System: Implement a notification system that sends alerts via push notifications, SMS, or email when faults are detected.

5. Testing and Validation

Prototype Testing: Conduct tests on the prototype to evaluate its performance under various conditions, including fault simulations.

Data Accuracy Validation: Ensure the accuracy of the data collected by sensors and the effectiveness of the machine learning algorithms in detecting anomalies.

User Acceptance Testing: Gather feedback from potential users to refine the application and ensure usability.

6. Deployment

Field Installation: Install the system on selected electric poles, ensuring proper calibration of sensors and connectivity to the cloud.

Initial Monitoring: Monitor the system for a trial period to assess performance and make necessary adjustments.

7. Maintenance and Updates

Regular Monitoring: Continuously monitor system performance and data integrity. Address any operational issues promptly.

Software Updates: Periodically update the firmware and machine learning models to improve functionality and adapt to new fault detection scenarios.

User Feedback Integration: Incorporate user feedback into system improvements and enhancements.

Conclusion

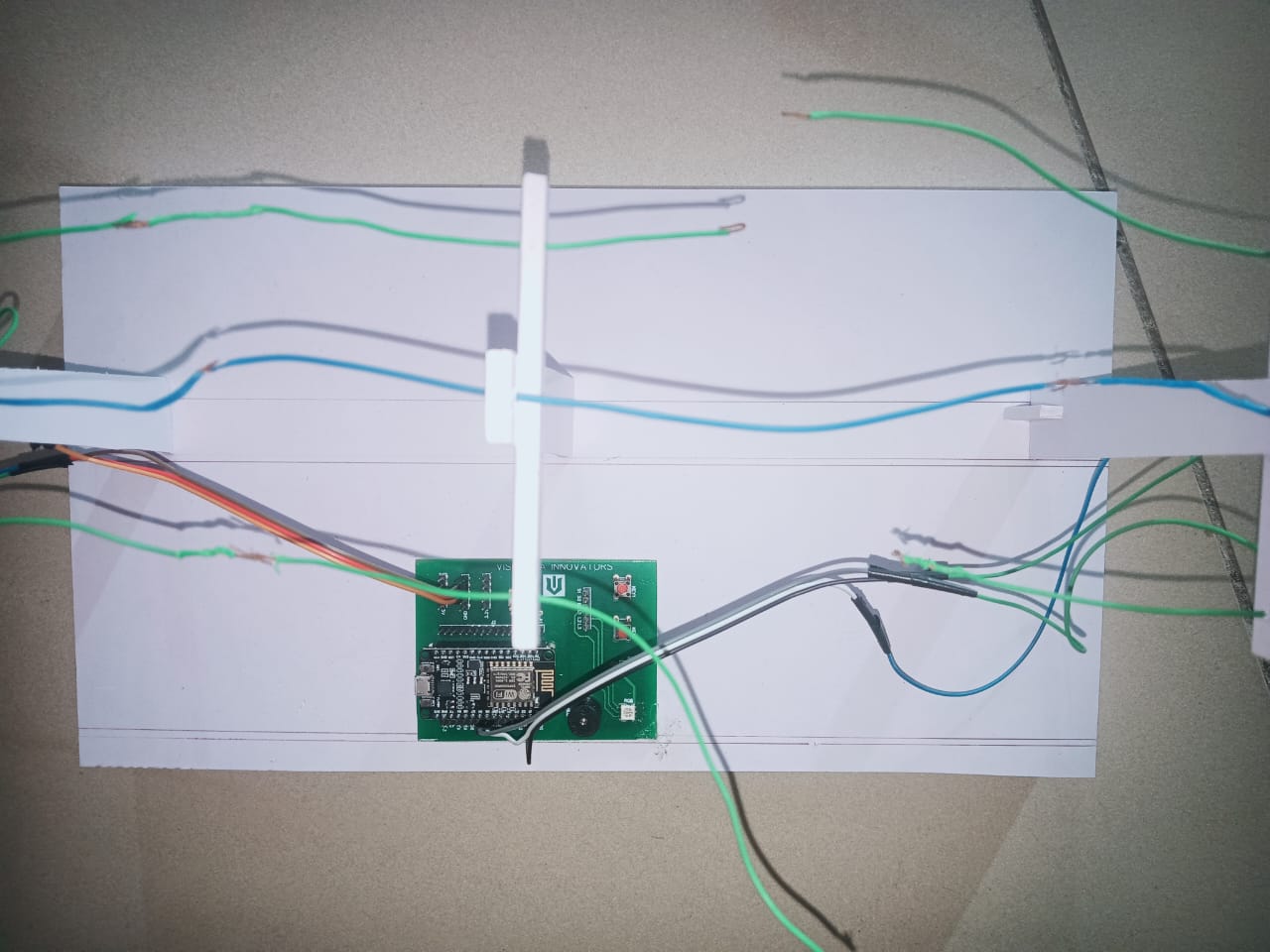
This structured working process ensures that the IoT-based broken line fault detection system is designed, developed, tested, and deployed effectively, providing reliable monitoring and rapid fault detection for electrical infrastructures. By integrating advanced technologies, the system aims to enhance the overall safety and reliability of electricity supply networks.

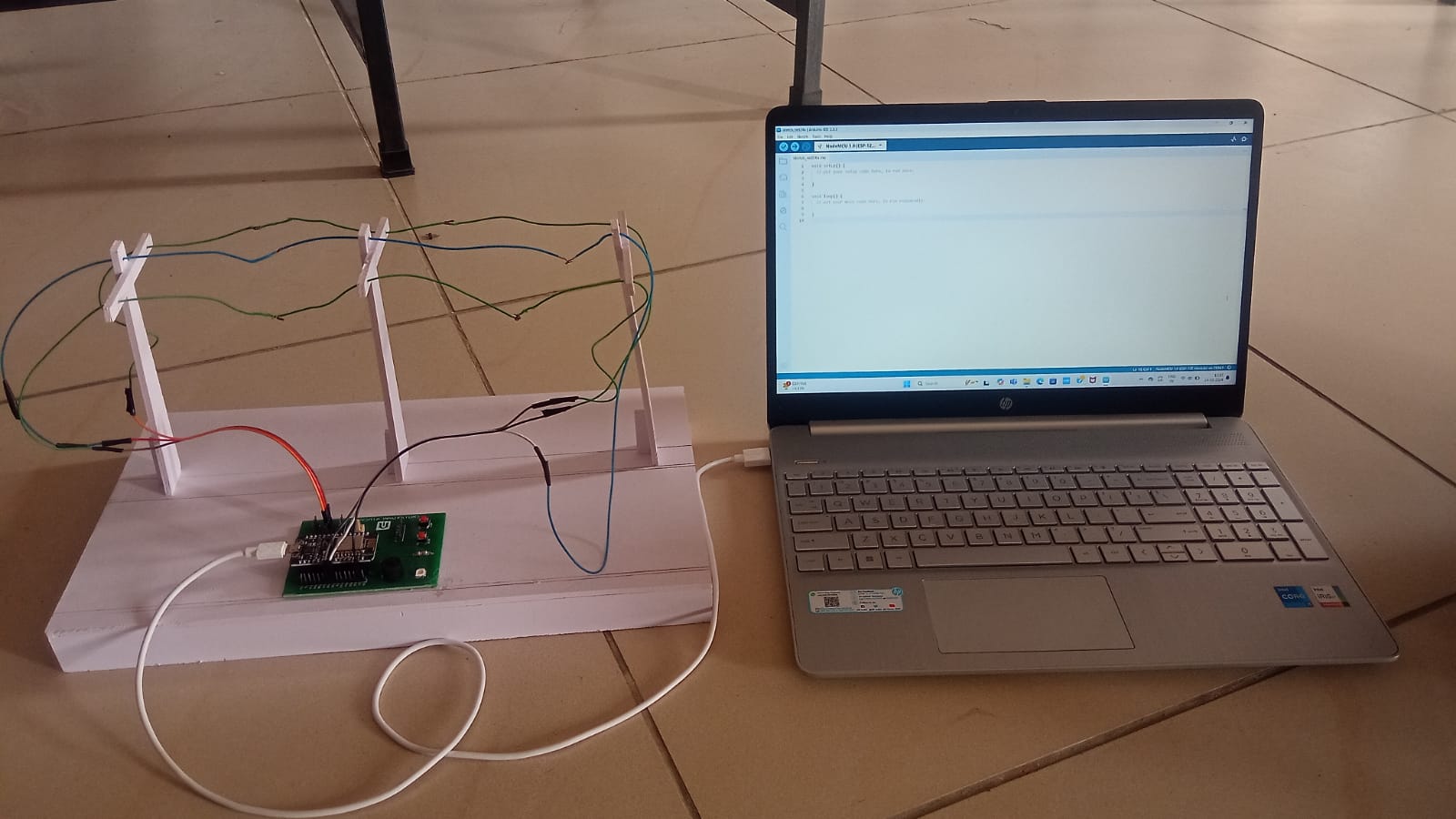
Get smarter responses, upload files and images, and more.

**Chapter 7:**

**OUTPUT & RESULTS**

**7.1 Project Photos**

****

****

**7.2 Screenshots of Results**

|  |  |
| --- | --- |
|  |  |
|  |  |

|  |
| --- |
|  |

**CHAPTER 8:**

**8. CONCLUSION**

**8.1** **Project Conclusion:**

In conclusion, the IoT-based broken line fault detection system for electric poles represents a significant advancement in utility monitoring and maintenance. By integrating real-time data collection through sensors, cloud computing, and advanced analytics, this system enhances the reliability and safety of electrical infrastructure.

Key benefits include:

1. Real-Time Monitoring: Continuous surveillance of electric poles enables immediate detection of faults, minimizing downtime and improving response times.

2. Reduced Maintenance Costs: Predictive maintenance can be implemented, allowing utilities to address potential issues before they escalate, thereby lowering repair costs.

3. Enhanced Safety: Early detection of line breaks helps prevent dangerous situations, protecting both utility workers and the public.

4. Data-Driven Decisions: The system generates valuable data that can inform future infrastructure improvements and resource allocation.

5. Scalability: The IoT framework allows for easy expansion, enabling utilities to implement the system across wider areas without significant additional investments.

Overall, the implementation of an IoT-based broken line fault detection system not only optimizes operational efficiency but also contributes to a more resilient electrical grid, ultimately leading to enhanced service quality for consumers. Future work may focus on further improving sensor technologies, integrating machine learning for predictive analytics, and exploring renewable energy applications.

**CHAPTER 9:**

**9. FUTURE SCOPE**

* 1. **Future addons or Updates in project:**

1. Advanced Sensor Technolog : Development of more sensitive and durable sensors capable of detecting a wider range of faults, including environmental conditions like high winds or ice accumulation.

2. Machine Learning and AI: Implementing machine learning algorithms to analyze data patterns and predict failures before they occur. This can enhance predictive maintenance capabilities.

3. Integration with Smart Grids: Further integration with smart grid technologies to enable real-time communication and coordination between various components of the electrical network.

4. Enhanced Data Analytics: Utilizing big data analytics to process large volumes of data from multiple sources, enabling better decision-making and operational strategies.

5. Remote Monitoring and Control: Developing mobile applications for utility personnel to monitor and manage systems remotely, improving response times and operational efficiency.

6. Sustainability Initiatives: Exploring the use of renewable energy sources to power IoT devices, contributing to greener utility practices.

7. Standardization and Interoperability: Establishing industry standards for IoT devices and communication protocols to ensure interoperability across different systems and manufacturers.

8. Public Safety and Awareness: Creating platforms to inform the public about outages and maintenance schedules, enhancing community engagement and safety.

9. Expanded Applications: Investigating additional use cases beyond fault detection, such as load monitoring, environmental impact assessments, and wildlife protection measures near power lines.

**CHAPTER 10:**

**10.FINAL CODE & RESOURCES**

The final code for the IoT-based Broken line fault detection system for electric poles with the ESP8266, and sending data to the Blynk platform. Additionally, resources such as libraries and references are provided for further development.

**10.1 CODE**

// Required Libraries

#include <ESP8266WiFi.h>

#include <BlynkSimpleEsp8266.h>

char auth[] = BLYNK\_AUTH\_TOKEN;

char ssid[] = "Robotics";

char pass[] = "makeindia";

#define line1 D1

#define line2 D5

#define line3 D6

int l1, l2, l3;

void setup() {

// put your setup code here, to run once:

pinMode(line1, INPUT\_PULLUP);

pinMode(line2, INPUT\_PULLUP);

pinMode(line3, INPUT\_PULLUP);

Serial.begin(9600);

Blynk.begin(auth, ssid, pass);

}

WidgetLED S1(V1);

WidgetLED S2(V2);

WidgetLED S3(V3);

void loop() {

// put your main code here, to run repeatedly:

Blynk.run();

l1=digitalRead(line1);

l2=digitalRead(line2);

l3=digitalRead(line3);

//Serial.println(l1);

//Serial.println(l2);

//Serial.println(l3);

if ((l1 == 0) && (l2 == 0) && (l3 == 0))

{

S1.off();

S2.off();

S3.off();

Serial.println("all are normal");

}

if(l1==1)

{

S1.on();

Serial.println("fault at line1");

Blynk.logEvent("notify","fault occured at line1");

delay(6000);

}

else

{

S1.off();

Serial.println("line1 normal");

}

if(l2==1)

{

S2.on();

Serial.println("fault at line2");

Blynk.logEvent("notify","fault occured at line2");

delay(6000);

}

else

{

S2.off();

Serial.println("line2 normal");

}

if(l3==1)

{

S3.on();

Serial.println("fault at line3");

Blynk.logEvent("notify","fault occured at line3");

delay(6000);

}

else

{

S3.off();

Serial.println("line3 normal");

}

}

**10.2 Resources:**

1. **Libraries**:
   * **ESP8266WiFi.h**: Handles Wi-Fi communication for NodeMCU.
   * **BlynkSimpleEsp8266.h**: Blynk lib
   * rary to interface with the Blynk IoT platform.
2. **Hardware Components**:
   * **NodeMCU (ESP8266)**: Microcontroller for Wi-Fi communication.
   * **Power Supply**: Ensure the NodeMCU has a stable 5V power supply.
3. **Blynk Platform**:
   * Virtual pins **V1**, **V2** and **V3** are used to send the error lines to the Blynk app for real-time monitoring.
4. **Tools**:
   * **Arduino IDE**: For writing and uploading code to the NodeMCU.
   * **Blynk App**: To create a user-friendly dashboard for broken line fault detection.