# Connecting an ESP8266 Dev Board

By Leighche Jaikarran

## Installing ESP8266 Board Package in Arduino IDE

1. **Open the Arduino IDE:**
    - Launch the Arduino IDE.
2. **Add ESP8266 Board URL:**
    - Go to **File** > **Preferences**.

In the "Additional Boards Manager URLs" field, paste the following URL:
bash

http://arduino.esp8266.com/stable/package_esp8266com_index.json

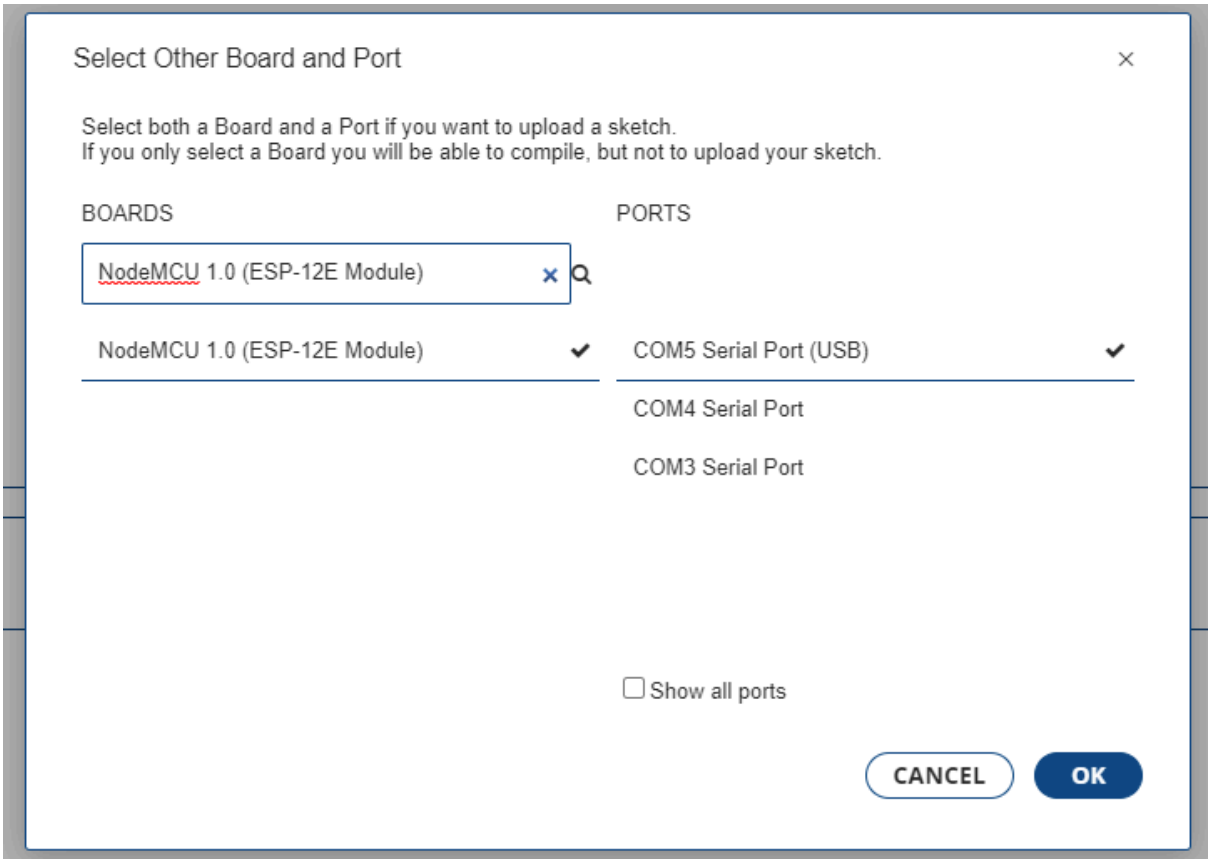
If you already have other URLs listed, separate them with a comma.

3. **Open Boards Manager:**
    - Go to **Tools** > **Board** > **Boards Manager**.
4. **Install the ESP8266 Platform:**
    - In the Boards Manager window, search for **ESP8266**.
    - Click on the entry labeled **"esp8266 by ESP8266 Community"**.
    - Click **Install.** This will install the latest version of the ESP8266 platform.
5. **Select Your ESP8266 Board:**
    - Once the installation is complete, go to **Tools** > **Board** and scroll down to select your ESP8266 board model (e.g., **NodeMCU 1.0 (ESP-12E Module)**).
6. **Restart Arduino IDE (Optional):**
    - If the boards still don't appear, try closing and reopening the Arduino IDE.

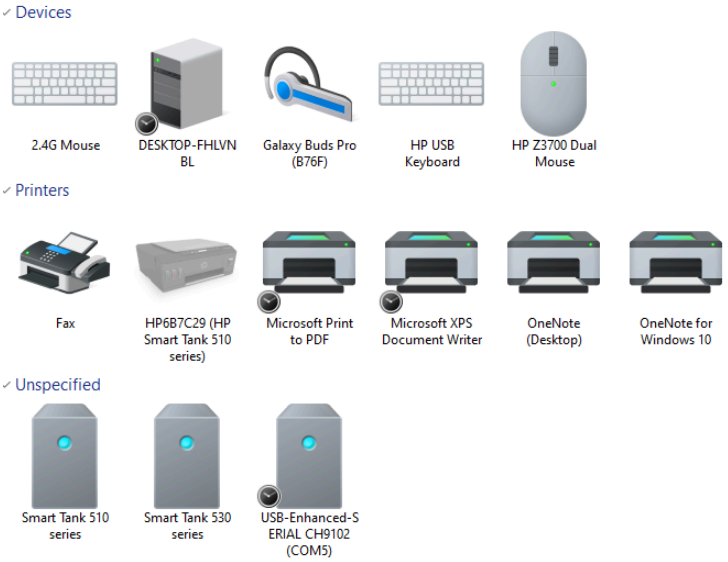
## Verifying the Installation:

- After installing the ESP8266 board package, go to **Tools** > **Board** and ensure you can now select an ESP8266 board from the list.

If your experiencing errors ensure your on port COM5




You can check this in your device manager by going to Action> Devices & Printers:




And seeing the COM port under the connected devices properties

**Find the Arduino Code here:**

https://github.com/AutoGrow-Solutions/Auto-ConnectAPI.git

Remember to change the thing speak API Write Key and the wifi SSIDand password according to yours when testing

**Limited Space on ESP8266-12F:**

The ESP8266-12F has limited memory. It has two main types of memory:

- **Flash Memory (Large, Read-Only):** Stores the program code and doesn't get erased when power is cycled.
- **RAM (Small, Read-Write):** Stores temporary data used by the program during operation. This gets cleared when power is cycled.

While flash memory is larger, code execution is slower from it compared to RAM. The limited RAM on the ESP8266-12F becomes a bottleneck when you have complex code or want to store a lot of data during program execution.

As seen in compiler below:

Output

```
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 310112 bytes to 225345...
Writing at 0x00000000... (7 %)
Writing at 0x00004000... (14 %)
Writing at 0x00008000... (21 %)
Writing at 0x0000c000... (28 %)
Writing at 0x00010000... (35 %)
Writing at 0x00014000... (42 %)
Writing at 0x00018000... (50 %)
Writing at 0x0001c000... (57 %)
Writing at 0x00020000... (64 %)
Writing at 0x00024000... (71 %)
Writing at 0x00028000... (78 %)
Writing at 0x0002c000... (85 %)
```

**How PROGMEM Helps:**

The `PROGMEM` keyword is a special modifier used with the ESP8266 (and other Arduino-like environments) to store constant data like strings in flash memory instead of RAM. This frees up valuable RAM for your program's running needs.

**Explanation of the Code:**

1. **Include Libraries:**
    - `ESP8266WiFi.h`: Provides functions for connecting to WiFi networks.
    - `ESP8266WebServer.h`: Enables creating a web server on the ESP8266.
    - `ESP8266HTTPClient.h`: Allows making HTTP requests to the internet.
2. **Store Credentials in PROGMEM:**
    - `const char ssid[] PROGMEM = "Room Wifi";`: Declares a constant character array `ssid` to hold the WiFi network name. The `PROGMEM` keyword instructs the compiler to store this string in flash memory.
    - Similar declaration for `password`.
3. **Define Web Server and LED Pin:**
    - `ESP8266WebServer server(80);`: Creates a web server object on port 80 (standard HTTP port).
    - `const int ledPin = 5;`: Defines the pin connected to an LED.
4. **Setup Function:**
    - **Serial Communication:** Initializes serial communication for debugging output.
    - **LED Pin Setup:** Sets the LED pin as an output and turns it off initially.
    - **Load Credentials from PROGMEM:**
        - Allocates buffers (`ssid_buf` and `password_buf`) in RAM large enough to hold the WiFi credentials.
        - `strcpy_P` function copies the string data from flash memory (`ssid` and `password`) to the RAM buffers (`ssid_buf` and `password_buf`).
    - **Connect to WiFi:** Attempts to connect to the WiFi network using the credentials loaded from flash.
    - **Server Start:** Starts the web server.
5. **Loop Function:**
    - `server.handleClient()`: Continuously checks for incoming requests to the web server and handles them.

6. **sendToThingSpeak Function:**
    - Checks if connected to WiFi.
    - Creates an HTTP client object and a WiFi client object.
    - Builds the URL for sending data to ThingSpeak, including the API key and LED state (`value`).
    - Makes an HTTP GET request to the ThingSpeak URL.
    - Parses the response from ThingSpeak (if successful) and prints it for debugging.

**Benefits of Using PROGMEM:**

- **Frees Up RAM:** By storing strings and other constant data in flash memory, you can significantly reduce the amount of RAM used by your program. This allows you to run more complex code on the ESP8266-12F.
- **Improved Performance:** Accessing data in RAM is faster than accessing data in flash memory. However, the small amount of RAM on the ESP8266 can become a bottleneck. By using `PROGMEM`, you can store less frequently accessed data in flash without significantly impacting performance and free up RAM for critical operations.

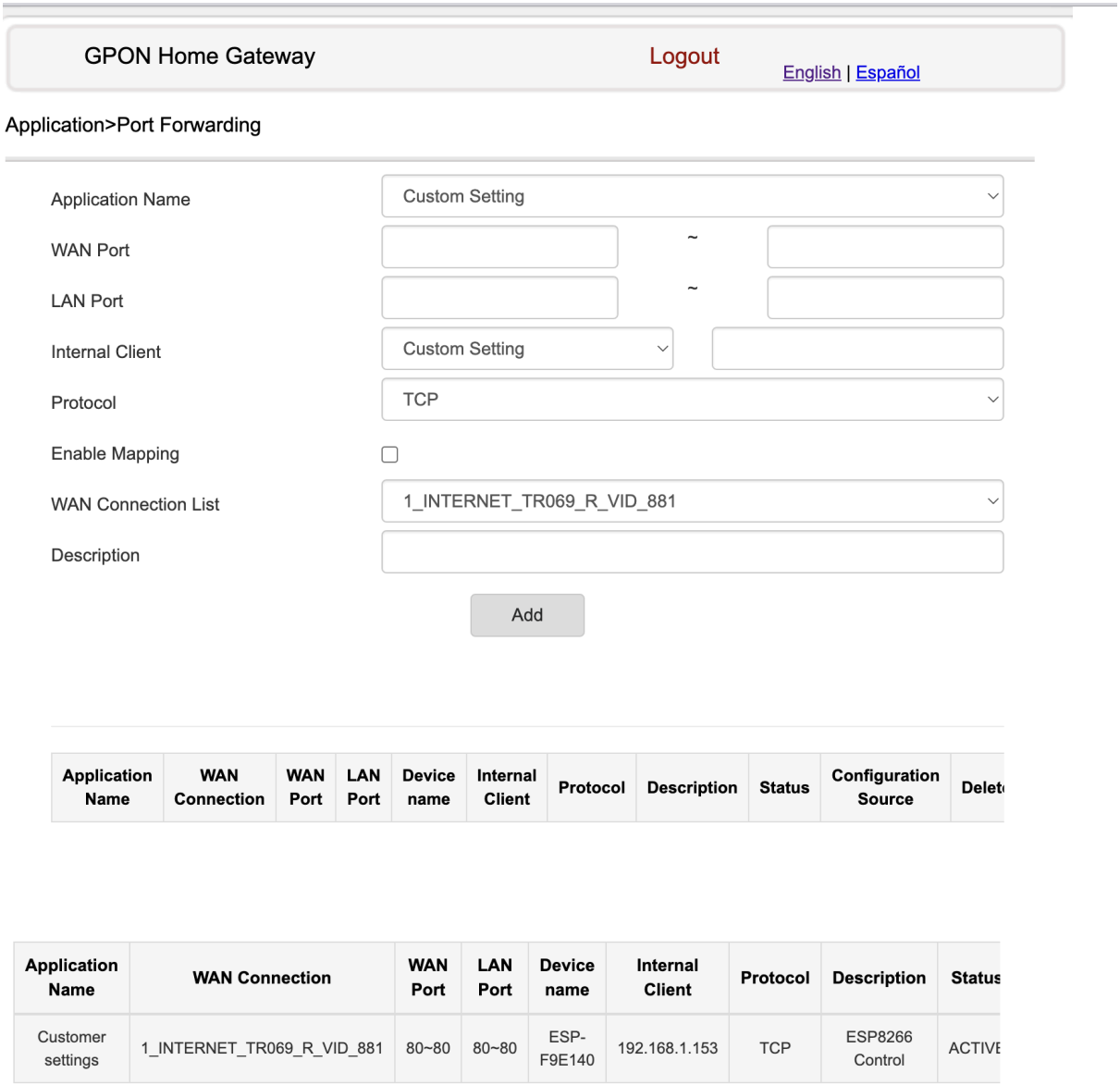
# Moving from sending request over your home Wifi to any connection

To access and control your ESP8266 from anywhere on the internet, you need to expose your device to the web securely. This usually involves setting up port forwarding on your router, using a dynamic DNS service, or employing cloud services that facilitate remote access. Here's a simplified guide to achieve this:

## 1. Port Forwarding

1. **Access Your Router Settings**: Open your router's configuration page. Typically, you can access it by typing `192.168.1.1` or `192.168.0.1` in your browser's address bar.
2. **Find Port Forwarding Section**: Look for the port forwarding section in your router settings. This could be under Advanced Settings, NAT, or similar.
3. **Create a New Port Forwarding Rule**:
    - **Service Name**: Enter a name (e.g., `ESP8266`).
    - **Port Range**: Set this to `80` (the port your server is running on).
    - **Local IP Address**: Enter the IP address of your ESP8266 (the one printed in the Serial Monitor).
    - **Local Port**: Set this to `80`.
    - **Protocol**: Choose `TCP` (or `Both` if `TCP` is not an option).
4. **Save and Apply**: Save your changes and apply them. This will forward external requests on port `80` to your ESP8266.

See below my setup, i have fibre and this is my screen: |




| Application Name | WAN Connection | WAN Port | LAN Port | Device name | Internal Client | Protocol | Description | Status |
|---|---|---|---|---|---|---|---|---|
| Customer settings | 1_INTERNET_TR069_R_VID_881 | 80~80 | 80~80 | ESP-F9E140 | 192.168.1.153 | TCP | ESP8266 Control | ACTIVE |

Port forwarding allows you to access devices on your local network from outside your network through the internet. Here's how it applies to your ESP8266 and the API you set up for controlling the LED:

## How Port Forwarding Works with Your ESP8266 API:

1. **Local Access**:
    - Inside your home network, you access the ESP8266's API by using its local IP address (e.g., `http://192.168.1.153/led/on`).
2. **External Access with Port Forwarding**:
    - With port forwarding set up, you can access your ESP8266 from anywhere on the internet by using your public IP address or a Dynamic DNS domain name (e.g., `http://<Your_Public_IP>/led/on` or `http://<Your_Domain_Name>/led/on`).

## What Port Forwarding Does:

1. **Maps External Requests to Local IP**:
    - Port forwarding tells your router to direct incoming traffic on a specific external port to the internal IP address and port of your ESP8266.
2. **Allows Remote Access**:
    - When you send a request to your public IP address on the forwarded port (port 80 in this case), the router forwards that request to your ESP8266's local IP address and port. This enables you to control the LED from anywhere with internet access.
3. **Enables API Functionality Remotely**:
    - Your ESP8266's API endpoints (`/led/on` and `/led/off`) become accessible from outside your home network. This means you can use a web browser or an HTTP client to send requests to turn the LED on or off, no matter where you are, as long as you have internet access.

## Example Workflow:

1. **Set Up Port Forwarding**:
    - Configure your router to forward requests on port 80 to your ESP8266's local IP address (e.g., `192.168.1.153`) on port 80.
2. **Find Public IP Address**:
    - Obtain your public IP address by searching "what is my IP" or use a Dynamic DNS service if you prefer a domain name.
3. **Access API Remotely**:
    - From anywhere with internet access, enter your public IP address or Dynamic DNS domain name followed by the API endpoint in a web

browser (e.g., `http://<Your_Public_IP>/led/on` or `http://<Your_Domain_Name>/led/on`).

## Security Considerations:

1. **Expose Only Necessary Ports**:
    - Only forward the ports you need. In this case, you're forwarding port 80 for HTTP access.
2. **Use Secure Authentication**:
    - Consider adding authentication to your API to prevent unauthorized access. You can do this by requiring a username/password or using API keys.
3. **Regularly Monitor and Update**:
    - Keep your firmware and router software updated to protect against vulnerabilities.
4. **Dynamic DNS**:
    - If your public IP address changes frequently, using Dynamic DNS ensures that you always have a domain name pointing to your current IP address.

By setting up port forwarding, you effectively make your ESP8266's API accessible from outside your local network, enabling remote control of your LED through HTTP requests.

One this is done you can now control the LED remotely from outside your network:

# Control the LED from any network

1. **Find your public IP address**: You can do this by searching "What is my IP" on Google. This will show the public IP assigned to your router by your Internet Service Provider (ISP).
2. **Access the ESP8266**: In a browser, enter your public IP address followed by `:80` (if using the default HTTP port 80). For example:
    - To turn the LED on: `http://<your_public_ip>/led/on`
    - To turn the LED off: `http://<your_public_ip>/led/off`
3. Ensure that your firewall settings allow traffic on port 80 and your router is properly configured for this connection.

If you're testing this from within your local network, you should use the internal IP `http://192.168.1.153/led/on` or `http://192.168.1.153/led/off` directly, but from outside your network, you'll need the public IP.

Please get the ipV4 address if you can't find it tap here:

https://whatismyip.com/

### Test Remote Access:

Use your browser or an HTTP client (like Postman or `curl`) to test the connection:
arduino:
Example ipv4 address : `http://102.32.222.22:80`

- This should reach the ESP8266 web interface or control page (assuming your ESP8266 is set up to handle web requests).

### Security Considerations:

- **Verify Your Firewall:** Ensure that your router's firewall is not blocking port 80. Some ISPs also block common ports (like 80), so you might need to test another port like 8080.
- **Use Strong Authentication:** Since you're exposing a device to the internet, it's good practice to secure access with authentication if not already done.
- **Consider Using HTTPS:** If you're sending any sensitive data or controlling devices, using HTTPS (with a self-signed certificate) can help secure your connection.

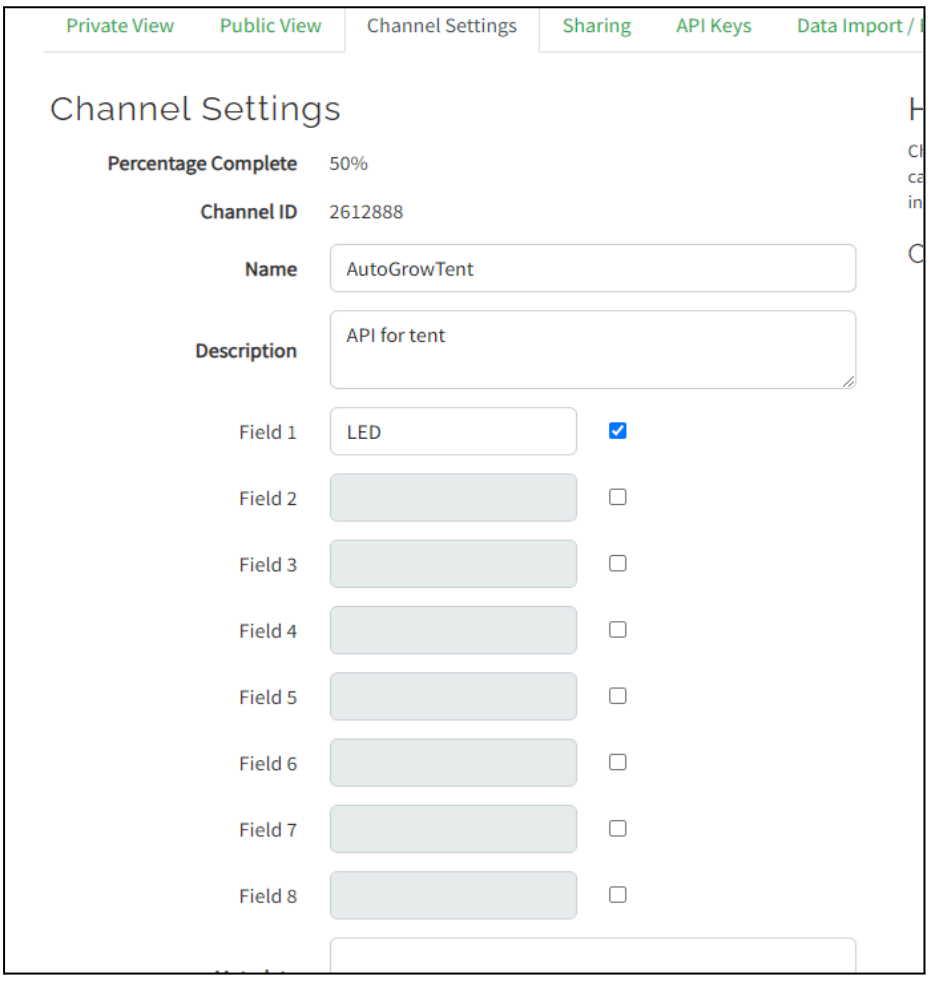
## 2. Dynamic DNS (Optional but Recommended)

Since your public IP address from your ISP might change periodically(unless you have a static IP from your ISP), you can use a dynamic DNS service to map a domain name to your changing IP address,This way, you won't have to keep checking and updating your public IP. You'll access your device via something like `http://yourname.ddns.net`.

1. **Register with a Dynamic DNS Provider**: Services like No-IP or DynDNS offer free and paid plans.
2. **Set Up Dynamic DNS on Your Router**: Some routers have built-in support for Dynamic DNS. Configure it with the details provided by your dynamic DNS provider.

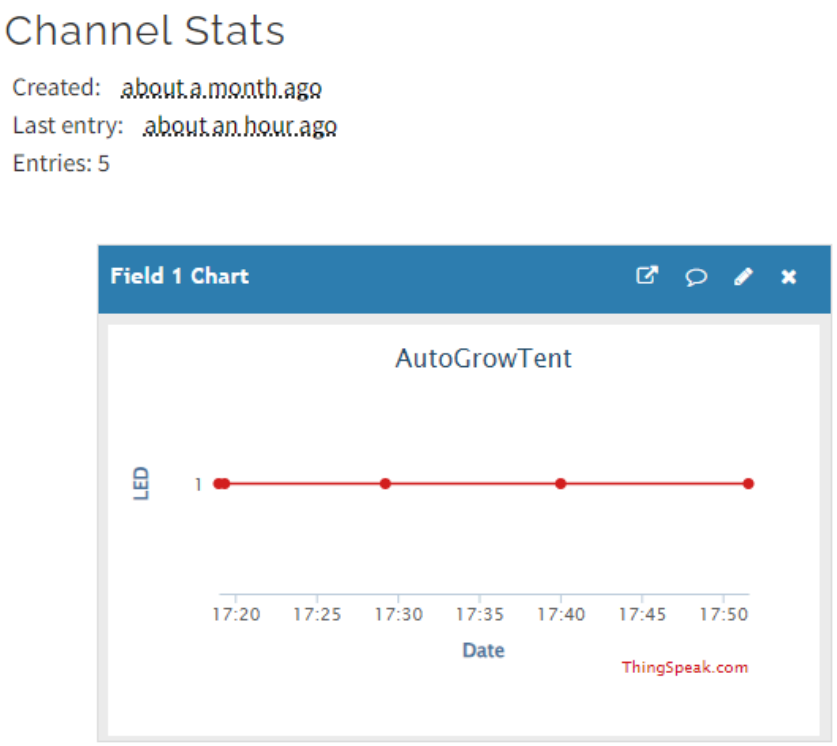
# ThingSpeak

link:




When on things speak create fields for whatever values you trying to map and tick it.

After you turned your lights on and off things speak maps them here:




The read key can be used in your android or Swift application to generate graphs of historical data as well as display the readings in the app itself, constantly.