JOHANNES KEPLER
UNIVERSITÄT LINZ

Erstellt von

Tomislav Dodig

Emin Mehic

Hannah Fehringer

Angefertigt am

Institut für Wirtschaftsinformatik – Communications Engineering

Betreuer / Betreuerin

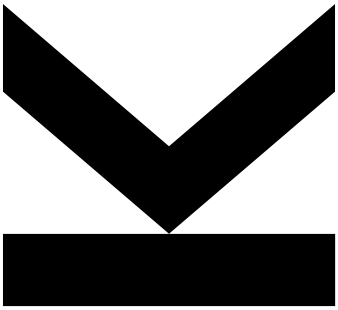
o.Univ.-Prof. Dipl.-Ing. Dr. Christian Stary

Claudia Kaar, BSc MSc

August 2020

SMART HOME MIT DEM INTERNET-OF-THINGS

Eine M5Stack- und Blockly-Anwendung



Diese Anleitung zeigt den Einsatz von Internet-of-Things-Technologien zur Kontrolle von Zugängen. Ziel dabei ist die anschauliche, be-greifbare Wissensvermittlung im Rahmen handlungsgeleiteter Lernprozesse. Die Ausführungen richten sich an Verantwortliche in Aus- und Weiterbildungseinrichtungen zur Integration von IoT-Szenarien in ihre Bildungsarbeit.

JOHANNES KEPLER

UNIVERSITÄT LINZ

Altenberger Straße 69

4040 Linz, Österreich

INHALTSVERZEICHNIS

1. Einführung 3
1.1. M5Go IoT Starter Kit und Komponenten 4
1.2. Smart-Home Kategorien 8
1.2.1. Kategorie Home (ENV + RGB) 8
1.2.2. Kategorie Plant (Earth + Light) 8
1.2.3. Kategorie Security (PIR + Laser TX/RX) 9
1.2.4. Kategorie Control (Angle + Dual Button) 9
2. Vorbereitung, Einrichtung & Installation 10
2.1. Notwendige Downloads 10
2.2. Einrichtung und Installation M5Stack & UIFlow 11
2.2.1. M5Burner 11
2.2.2. UIFlow 13
2.2.3. Variablen 17
2.2.4. Labels 17
2.2.5. Units 18
3. Realisierung des Smart-Home-Systems 19
3.1. Kommunikation zwischen den Geräten 19
3.1.1. MQTT 19
3.1.2. MQTT-Broker 20
3.1.3. Publish & Subscribe 20
3.1.4. MQTT Realisation für den Anwendungsfall 21
3.1.5. MQTT Webclient 21
3.2. Programmieren am M5Stack 24
3.2.1. Erstellung einer Verbindung zwischen M5Stack & MQTT Broker 24
3.2.2. Smart-Home Implementierung am M5Stack 27
3.2.2.1. Home 27
3.2.2.2. Plant 34
3.2.2.3. Security 37
3.2.2.4. Control 42
3.3. Steuerung über Mobilgerät 48
3.3.1. Eigene Handy-App entwickeln 48
4. Abbildungsverzeichnis 50

1. Einführung

Die Entwicklungen rund um Internet-Technologien halten in allen Bereichen der Gesellschaft Einzug. Dieser Anwendungsfall setzt sich mit dem sogenannten Internet-of-Things (IoT) und einigen damit verbundenen Methoden und Werkzeugen auseinander sowie der Erstellung einer Mobilen Applikation. Die in der Folge vorgestellten und genutzten Komponenten und Software können eingesetzt werden, um das eigene Zuhause mit einem Smart Home System zu erweitern . Die in der Anleitung gezeigten Schritte erlauben die Konstruktion eines Systems, dass dazu dient, Abläufe und Zustände im Haus mittels einer simplen Handy Applikation zu beobachten, aber auch einfach zu steuern. Auch einige Sicherheitsaspekte werden dabei berücksichtigt. Einige Beispiele zu den Funktionalitäten

- Temperatur regeln
- Licht regeln (pro Zimmer, Lichtstärke einstellbar)
- Pflanzenzustand messen (Licht & Feuchtigkeit der Erde)
- Bewegung im Raum erkennen über
    - Bewegungsmelder
    - Laserstrahl

Zur Umsetzung des Smart Home Systems mit dem Internet-of-Things Anwendungsfalles wird das IoT-System **M5Stack** verwendet. Es besteht aus mehreren angreifbaren Elementen wie Sensoren, Bedienungsgeräte mit Tasten, Kabelverbindungen, mehreren M5Stacks die mit einer Bildschirmoberfläche ausgestattet sind. Zur Umsetzung praxistauglicher Anwendungen ist zusätzlich eine Programmierumgebung erforderlich. In unserem Fall wird die Programmierumgebung UIFlow mit der graphischen Programmiersprache Blockly eingesetzt. Der Vorteil dieser Entwicklungsumgebung ist, dass zum Arbeiten mit dem M5Stack und Blockly von UIFlow kein Vorwissen im Bereich digitaler Systeme sowie Programmieren an sich notwendig ist. Zusätzlich wird eine reine Code Programmierung durch die Programmiersprache Python unterstützt.

Der M5Stack ist ein Microcontroller-Entwicklungsboard, ausgestattet mit einem Gehäuse, Grafikdisplay, microSD Steckplatz, USB-C und Lautsprechern sowie Erweiterungssteckern. An einen M5Stack können mehrere zusätzliche Komponenten mit unterschiedlichen Fähigkeiten, wie z.B. dem Erkennen von Bewegungen, angeschlossen werden.

- Diese Komponenten werden als ***Units*** bezeichnet.
- Die Anschlüsse am M5Stack für diese Units werden als ***Ports*** bezeichnet.

Mit Hilfe dieser Elemente kann ein einfaches Programm entwickelt werden. Der M5Stack ist darauf ausgelegt, Units zu verbinden. Durch stufenweise Erweiterung lassen sich an- und begreifbar, und damit in überschaubaren Schritten, komplexe Systeme entwickeln. Da der M5Stack auch mit Lego-Bausteinen kompatibel ist, bietet das System auch für

Kinder und Jugendliche die Möglichkeit, Hemmschwellen zur Beherrschbarkeit von digitalen Systemen, und somit auch zur Programmierung, zu überwinden.

UIFlow ist gezielt darauf ausgelegt, den M5Stack zu programmieren, und zwar mittels der visuellen Programmiersprache ***Blockly*** oder textbasiert in Python. Blockly ist ein visuelles Drag-and-Drop Block-Stecksystem, mit dem die Funktionalität des M5Stack bearbeitet werden kann. Mit Blockly ist für das Programmieren wenig Schreibaufwand verbunden, sodass die Entwicklung durch einfaches Verschieben und sachorientiert erfolgen kann.

Die Steuerung der einzelnen Komponenten des Smart-Home-Systems übernimmt ein M5Stack mit dem man über dem Bildschirm alles im Überblick hat und mit den Tasten gewisse Bereiche auch steuern kann, wie z.B. Licht ein- und ausschalten. Für Fortgeschrittene würde sich die Entwicklung einer eigenen mobilen Applikation z.B. als Handy Applikation anbieten. In dieser Anleitung wird auch eine fertige Handy Applikation vorgestellt, wo eine Kommunikation mit den M5Stacks ermöglicht wird, sowie aber auch Tipps und Anleitungen gezeigt, wie man eine eigene Handy App selbst machen kann.

Diese Anleitung ermöglicht einen einfachen Einstieg in die Verwendung des M5Stack und seiner Komponenten, sowie der Programmierumgebung UIFlow. Sie hilft, einen Anwendungsfall mit den erwähnten Bestandteilen (M5Stack, UIFlow) zu erstellen und bietet Hilfestellungen um eine eigene Handy-Applikation zu erstellen.

Das Smart Home System wird somit nicht nur benutzerfreundlich entwickelt, sondern weist alle Eigenschaften auf, wie IoT-Systeme praktisch genutzt und im individuellen Umfeld eingesetzt werden können.

1.1. M5Go IoT Starter Kit und Komponenten

In diesem Abschnitt werden das zur Realisierung des Anwendungsfalls erforderliche M5Go IoT Starter Kit ESP32 sowie zusätzliche Komponenten erläutert und beschrieben.

Das Starter Kit von M5 enthält 6 verschiedene Units, einen M5Stack sowie Lego Bausteine. Dieses Starter Kit erleichtert, mit dem Konzept des Internet-of-Things (IoT) vertraut zu werden und wesentliche Komponenten kennenzulernen. Jede Komponente kann separat angeschafft werden, sodass der Einsatz von Komponenten in mehreren Etappen erfolgen kann.

Im Starter Kit enthalten ist:

M5Stack:

Diese Komponente ist die zentrale physische Komponente jedes IoT-Systems. An den M5Stack lassen sich alle Units anschließen. Dies ermöglicht die Kombination von mehreren Units.

Motion Sensor:

Diese Unit kann Bewegung erkennen.

ANGLE Sensor:

Diese Unit erlaubt die Erkennung ausgewählter Werte.

Grove Hub:

Diese Unit erlaubt es, bis zu drei Units an einen Port anzuschließen.

IR Remote:

Diese Unit ermöglicht die Kommunikation mit Geräten über Infrarot-Strahlen.

RGB LED:

Diese Unit besitzt drei Lampen, die jeweils sämtliche Farben der RGB Skala annehmen können.

Environment Sensor:

Diese Unit kann die Temperatur, Luftfeuchtigkeit und den Luftdruck messen.

Abbildung 1 gibt einen Überblick über die beschriebenen Komponenten (Units) des Starter Kit.

Abbildung 1: M5Go IoT Starter Kit ESP32

Für den Anwendungsfall werden jedoch noch weitere Units benötigt:

pb.Hub:

An diesen Hub lassen sich bis zu sechs Units anschließen, jedoch mit der Einschränkung, dass diese nur für den Port B gedacht sind. Eine Besonderheit von diesem Hub ist es, dass er an Port A angesteckt werden muss.

Abbildung 2: Pb.Hub Unit Abbildung

Earth:

Dieser Unit ist speziell für Pflanzen bzw. für die Erde, in der sie steckt, gedacht. Sie misst den Feuchtigkeitsgrad der Erde und mithilfe dieser Information, soll man benachrichtigt werden, wann die Pflanze wieder zu gießen ist.

Abbildung 3: Earth Unit Abbildung

Light:

Diese Unit misst die Lichtstärke, die auf den Sensor einwirkt. So kann man generell die Lichtstärke im Raum messen. In unserem Anwendungsfall würde man so ermitteln können, ob die Pflanze auch ausreichend Licht bekommt.

Abbildung 4: Light Unit Abbildung

Dual Button:

Diese Unit besteht aus einem blauen und einer roten Taste. Diese wird genutzt um die Temperatur im Haus ein zu stellen.

Abbildung 5: Dual Button Unit Abbildung

Laser RX:

Diese Unit funktioniert nur in Kombination mit der Unit Laser TX. Sie empfängt den Infrarotstrahl von Laser TX.

Abbildung 6: Laser RX Unit Abbildung

Laser TX:

Diese Unit funktioniert nur in Kombination mit der Unit Laser RX. Diese Unit sendet einen Infrarotstrahl, der von der Laser RX Unit empfangen werden kann.

Abbildung 7: Laser TX Unit Abbildung

1.2. Smart-Home Kategorien

Der Anwendungsfall besitzt 4 Kategorien wie man den M5Stack mit den Units verbindet und sind wie folgt unterteilt:

1.2.1. Kategorie Home (ENV + RGB)

Diese Kategorie ist gedacht als Herzstück eines jedes Raumes. Diese beinhaltet ein ENV-Sensor Unit zum Messen von Temperatur, Luftfeuchtigkeitswerte usw. sowie auch RGB Units für die Beleuchtung des Zimmers. Von dieser Kategorie werden so viele Exemplare benötigt, wie viele Räume man mit dem Smart-Home-System ausstatten will.

1.2.2. Kategorie Plant (Earth + Light)

Diese Kategorie ist für die Überwachung von Pflanzen zuständig. Dafür wird zusätzlich zum M5Stack das **Earth** und das **Light Unit** benötigt. Mit denen wird die Feuchtigkeit der Erde sowie die Lichtstärke gemessen. So soll man erfahren, ob die Pflanze ausreichend Wasser und Licht bekommt. Die Light Unit wird 1x pro Zimmer benötigt, während hingegen die Earth Unit so oft benötigt werden, wie viele Pflanzen man beobachten will. Beispiel 2 Zimmer mit jeweils 3 Pflanzen würde bedeuten: zwei M5Stacks an der jeweils 1 Light und 3 Earth angeschlossen sind.

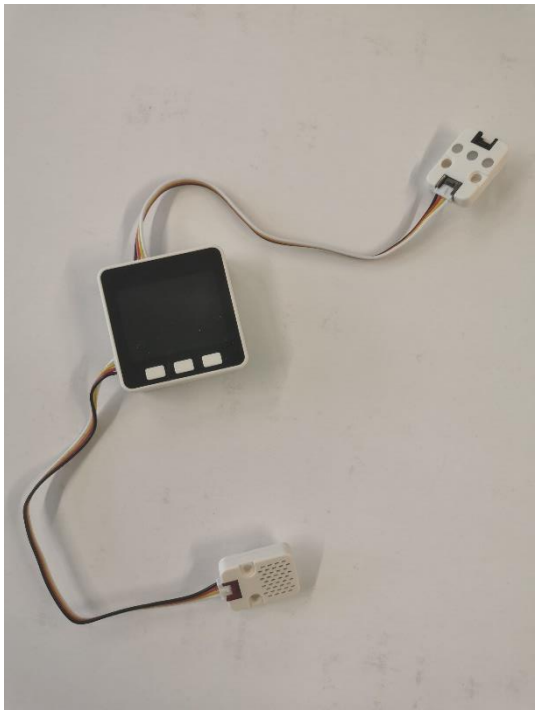


Abbildung 9: M5Stack und die dazugehörigen Units für das Sicherheit mit dem Internet-of-Things System

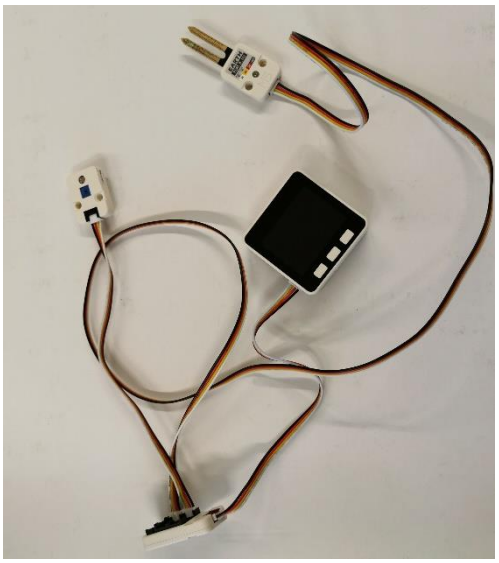


Abbildung 8: M5Stack und die dazugehörigen Units für das Sicherheit mit dem Internet-of-Things System

1.2.3. Kategorie Security (PIR + Laser TX/RX)

Ähnlich zu einem anderen Anwendungsfall, soll diese Kategorie das ganze Smart-Home-System auch mit einem Sicherheitsaspekt ausgestattet werden. Dieser wird in der Regel nur einmal gebraucht pro Haus. Je nach dem, wie stark man sein Zuhause „schützen“ und ausstatten möchte, kann man das mit beliebig vielen Units erweitern (*Hinweis: Anwendungsfall Security bietet viele Ideen bezüglich der Units*) aber für unseren Anwendungsfall haben wir uns auf einen einfachen **Bewegungsmelder (PIR)** und einer **Laserschranke** beschränkt.

1.2.4. Kategorie Control (Angle + Dual Button)

Diese Kategorie ist das Steuerelement für das Smart-Home-System. Mit dem **Angle Unit**, welches ein Drehschalter ist, soll die Lichtstärke im Haus geregelt werden. Mit dem **Dual Button** soll die Wunsch Temperatur im Haus eingestellt werden. Mit der Handyapplikation ist es auch möglich, diese Einstellungen im Haus vorzunehmen, deswegen ist die Kategorie nicht unbedingt erforderlich, jedoch trotzdem zu empfehlen, da man so die Wunscheinstellungen auch lokal im Haus vornehmen kann, wenn das Mobiltelefon grad nicht verfügbar ist.

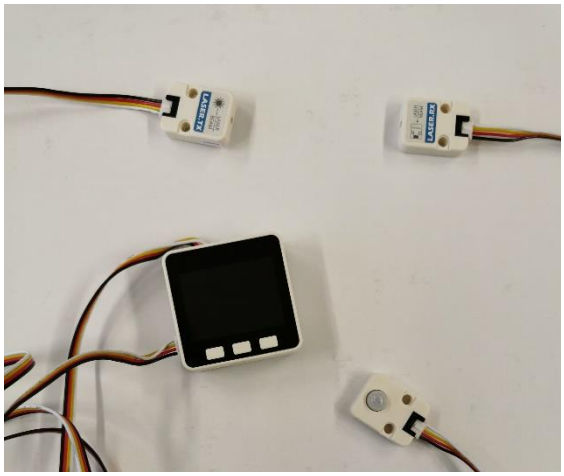


Abbildung 11: M5Stack und die dazugehörigen Units für das Sicherheit mit dem Internet-of-Things System

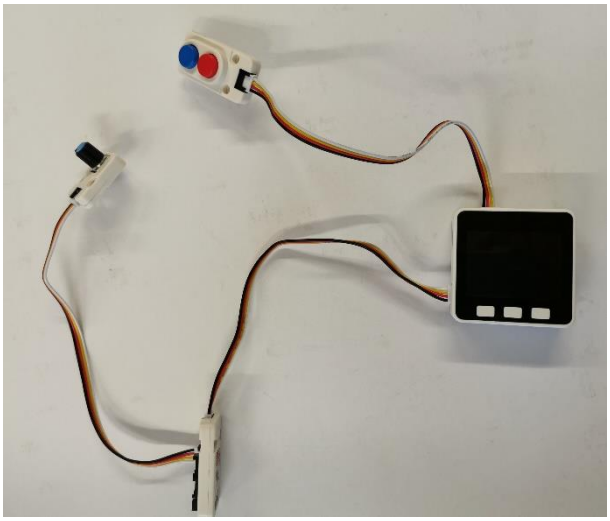


Abbildung 10: M5Stack und die dazugehörigen Units für das Sicherheit mit dem Internet-of-Things System

2. Vorbereitung, Einrichtung & Installation

2.1. Notwendige Downloads

Die folgende Liste beinhaltet Informationen und Links, um die notwendigen Werkzeuge zu beschaffen, um den Anwendungsfall zu realisieren

- ☐ Installation von M5Burner auf Computersystem - M5Burner ist unter https://m5stack.com/pages/download zu finden
- ☐ Verwendung des M5Burner um auf dem M5Stack die aktuellste Firmware zu installieren (mehr Info unter M5Burner)
- ☐ Installieren von Treiber CP2104 Driver des M5Stack – dieser ist unter https://m5stack.com/pages/download zu finden
- ☐ Installation von lokaler UIFlow-Programmierumgebung auf Computersystem (Mehr Info unter UIFlow)
- ☐ (Optional für Fortgeschrittene): Installation von Android Studio für die Entwicklung der Android Applikation für das Handy. (https://developer.android.com/)

2.2. Einrichtung und Installation M5Stack & UIFlow

Der M5Stack bildet gemeinsam mit den in diesem Abschnitt vorgestellten Software-Entwicklungskomponenten das Internet-of-Things-System. Zur Entwicklung und zum Betrieb wird der M5Stack mit dem Computersystem über ein USB-C Kabel angeschlossen. Sobald der M5Stack eingeschaltet wird, ist es notwendig, den richtigen Modus auszuwählen, um eine Verbindung mit dem Computersystem herzustellen. Dafür drückt man Knopf C, um in die Setup Kategorie zu gelangen. Von dort wählt man „Switch

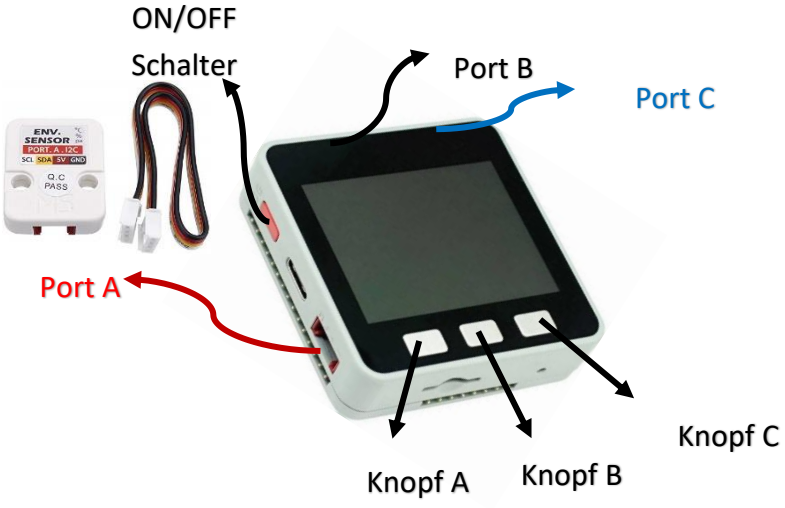


Abbildung 12: Beschriftung vom M5Stack

to USB mode“ um das Gerät richtig einzustellen. In Abbildung X wird der M5stack mit seinen verschiedenen Ports und Knöpfen gezeigt.

2.2.1. M5Burner

Bei der allerersten Verwendung des M5Stack muss man den M5Burner verwenden. Den M5Burner kann man von der Seite https://m5stack.com/pages/download laden. Durch den M5Burner kann man die Firmware auf dem M5Core installieren. Dieser muss dieselbe Version haben wie UIFlow (wenn man damit arbeitet).

Den M5Burner kann man nach dem Download über die Datei „M5Burner.exe“ ausführen, wie in der nächsten Abbildung gezeigt.

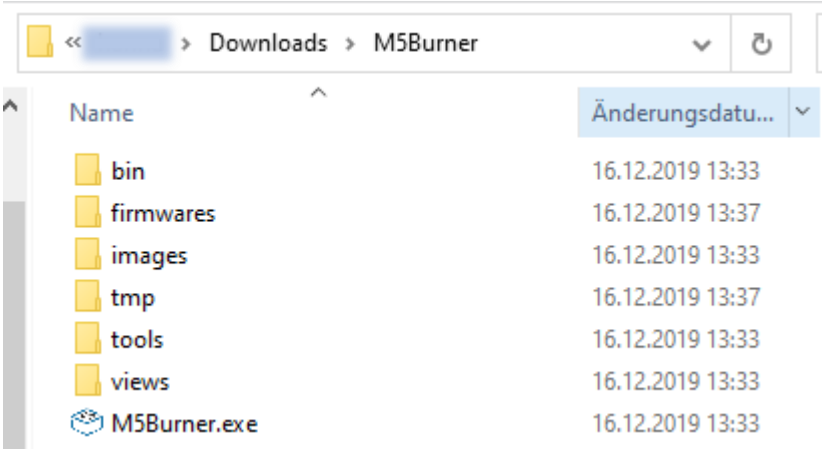


Abbildung 13: Ordnerstruktur vom M5Burner

Sobald man den M5Burner geöffnet hat, muss man die richtige UIFlow-Version laden. Dafür muss man auf das Download-Symbol neben der ausgewählten UIFlow-Version klicken. Danach sind der M5Stack über das USB-C Kabel an den Laptop/PC anzuschließen und der richtige COM-Anschluss auszuwählen. Empfehlenswert ist es hier die Version „UIFlow-v1.4.5“ auszuwählen, weil diese die Möglichkeit erlaubt innerhalb der grünen Felder den WLAN-Name sowie das dazugehörige Passwort einzugeben. Für den Anwendungsfall ist es besonders wichtig, dass der M5Stack als auch das dazugehörige Smartphone (oder Tablet) im gleichen Netzwerk ist.

Anschließend kann man auf Burn klicken und die Firmware wird auf dem M5Stack installiert. Die Darstellung vom M5Burner wird in Abbildung 5 gezeigt.

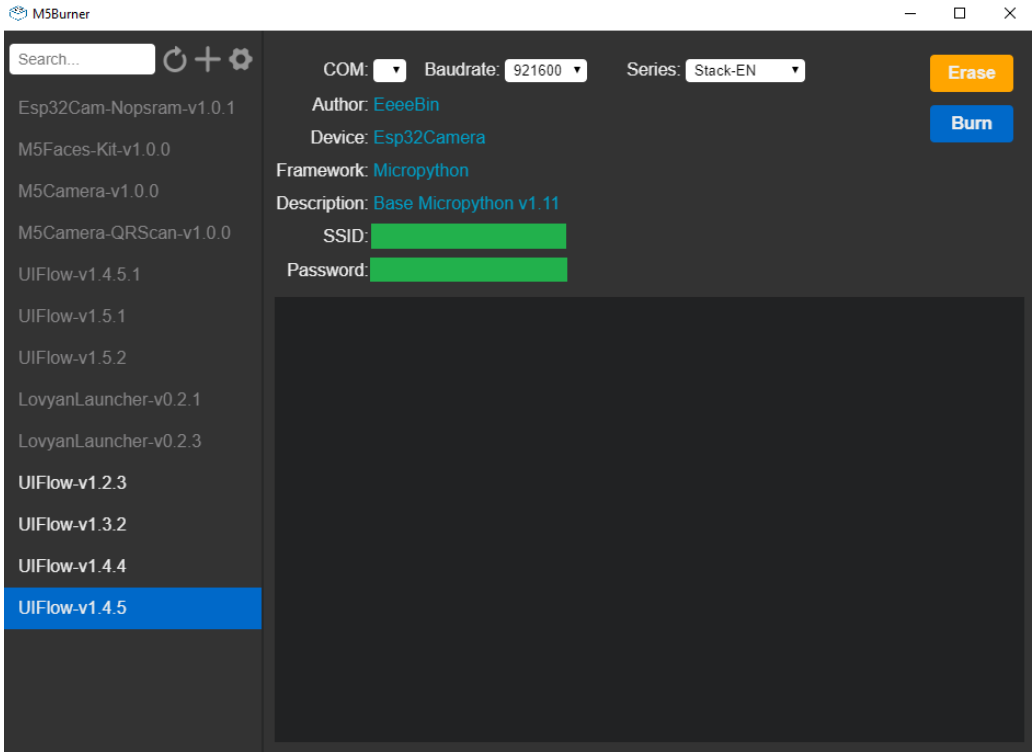


Abbildung 14: Benutzeroberfläche vom M5Burner

Es kann vorkommen, dass das Computersystem den M5Stack nicht erkennt. Sollte dies geschehen, kann der Treiber „CP2104 Driver“ für den M5Stack, ebenfalls von der Seite https://m5stack.com/pages/download geladen werden.

2.2.2. UIFlow

Die lokale Programmierumgebung von UIFlow kann von https://m5stack.com/pages/download geladen werden. Innerhalb der UIFlow-Oberfläche wird beim Starten der lokalen Programmierumgebung nach einem verbundenen M5Stack gefragt, welcher in der COM: Schaltfläche ausgewählt wird, wie in Abbildung 15 zu sehen ist.

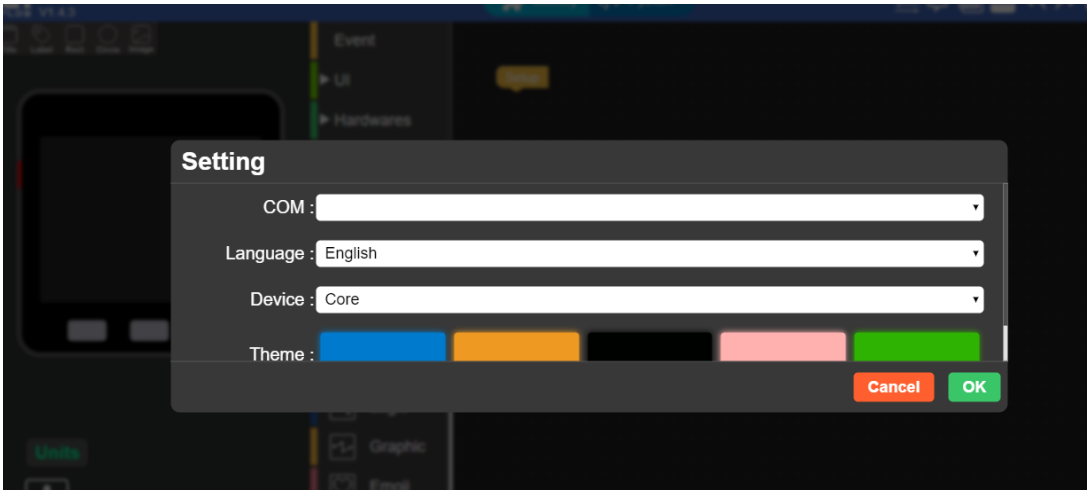


Abbildung 15: Setting Auswahl vom UIFlow

Üblicherweise wird der korrekte COM-Anschluss automatisch angezeigt. Sollte dies jedoch nicht der Fall sein, so kann man über den Gerätemanager des Computersystems herausfinden, welcher COM-Anschluss in UIFlow zu verwenden ist. Dafür klickt man mit der rechten Maustaste auf das Windows Symbol im unteren linken Bereich des Desktop und anschließend wählt man „Geräte-Manger“ aus, wie in der folgenden Abbildung zu sehen ist.

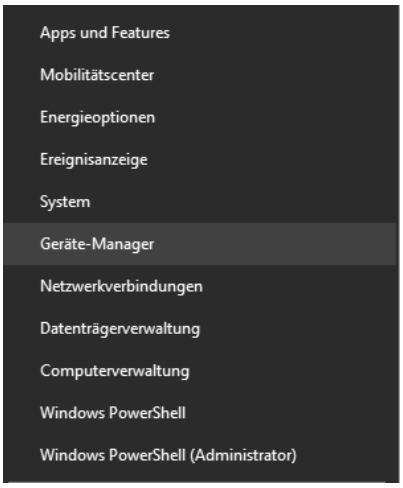


Abbildung 16: Die Ansicht, nachdem man auf das Windows Symbol einen Rechtsklick ausführte

Unter „Anschlüsse (COM & LPT)“ kann man dann herausfinden, über welchen COM der M5Core angesprochen werden kann. Diese Information findet man, sobald „Silicon Labs CP210x USB to UART Bridge (…)“ unter den Anschlüssen aufscheint.

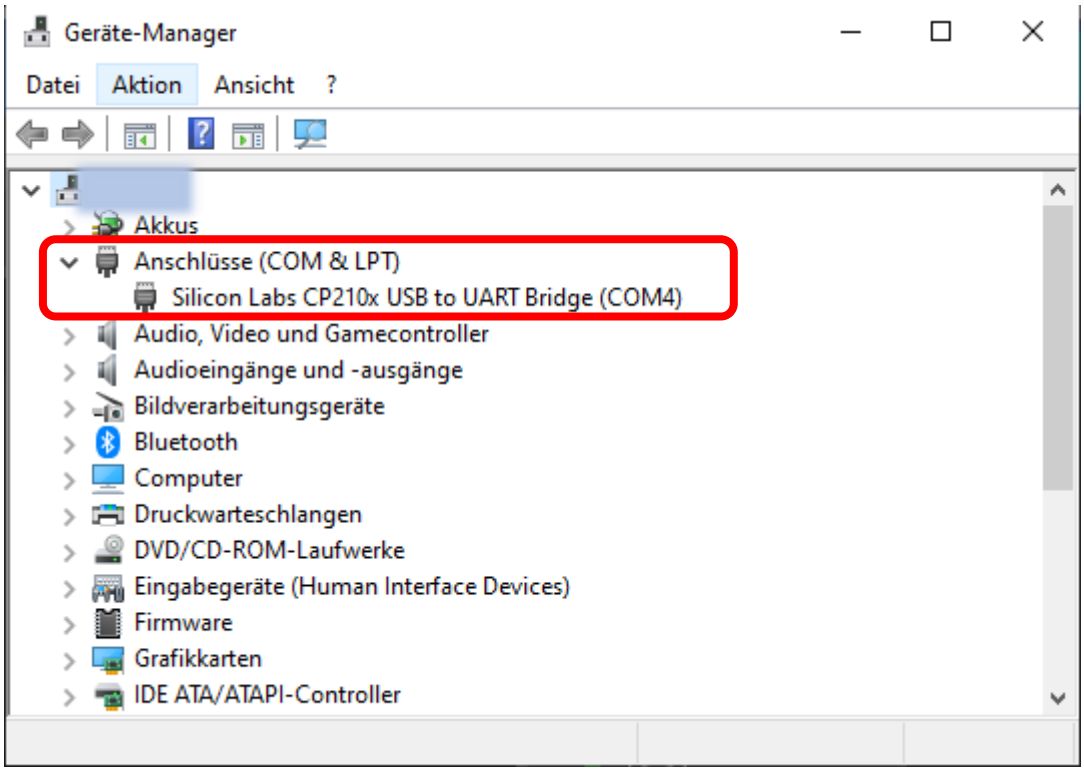


Abbildung 17: Anzeige der verschiedenen Anschlüsse am Computer

Dieser COM-Anschluss muss dann auf der Startseite von UI-Flow eingegeben werden.

Nach einer erfolgreichen Verbindung ist die Nutzung vom M5Stack und UIFlow vorbereitet. Dies erkennt man, wenn am unteren linken Rand des Bildschirms der Status „Connected“ sichtbar ist. Diese Anzeige ist in der nächsten Abbildung zu sehen.

Abbildung 18: M5Stack-Verbindung wird erfolgreich angezeigt.

Danach ist die Nutzung der UIFlow-Plattform möglich – die ersten Schritte für die Entwicklung einer eigenen Anwendung sind gesetzt. In Abbildung 19 werden die Benutzungsschnittstelle dargestellt und die wichtigsten Komponenten hervorgehoben. Diese Komponenten werden unterhalb des Bildes noch genauer erklärt.

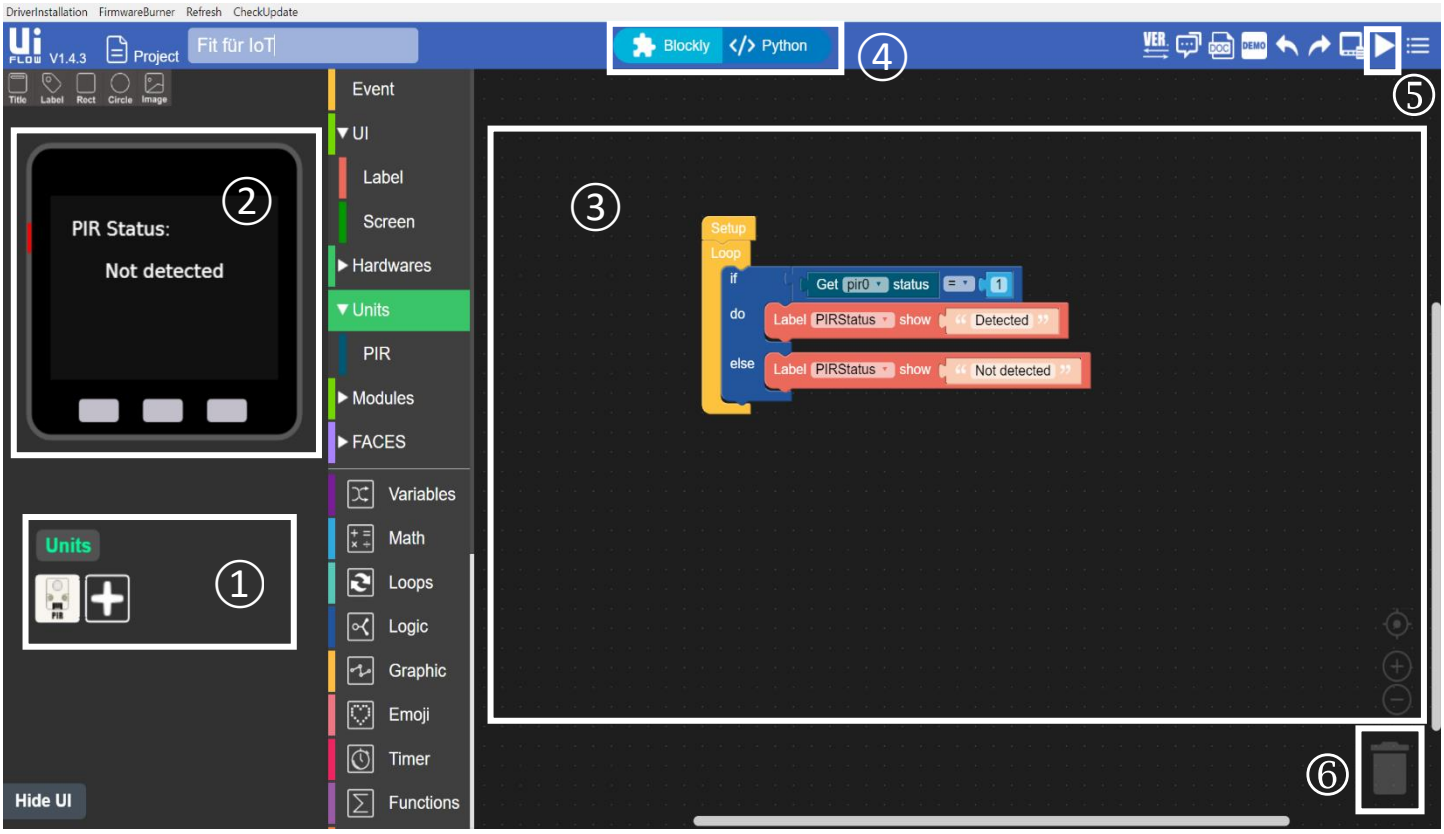


Abbildung 19: Erklärung zur UIFlow Oberfläche (anhand eines PIR Beispiel).

1. Hier können jene Units hinzugefügt werden, die man programmieren möchte. Nähere Informationen zu Units siehe Abschnitt Units.
2. Die Oberfläche des M5Core kann hier kreiert werden. Kreise, Texte, Quadrate und Bilder können per Drag and Drop eingefügt werden.
3. Dies ist der Programmierbereich (Coding Area). Hier erstellt man per Drag and Drop seinen Code, den man später ausführen möchte, zusammen.
    a. Blöcke sind Bausteine in verschiedensten Farben mit verschiedensten Funktionen, die benutzt werden, um eine Ablauflogik zu programmieren. Abhängig davon, welche Funktion man benötigt, kann man einen Block per Drag und Drop auf der Oberfläche einfügen. Im Beispiel werden Blöcke aus dem Event-, PIR- und Label-Block verwendet.
4. Hier kann man zwischen den Programmiersprachen Blockly und Python wechseln.
5. Damit spielt man den Code temporär auf den M5Core, welcher diesen ausführt, sofern er korrekt zusammengestellt wurde.
6. Per Drag and Drop kann man zu diesem Papierkorb Blöcke hinziehen, die man nicht mehr benötigt, damit sie gelöscht werden.

Die Blöcke, die man zum Programmieren verwendet, befinden sich in dem Panel auf der linken Hälfte der UIFlow-Oberfläche. Dort kann zuerst eine Metakategorie ausgewählt werden und anschließend der Block, den man benötigt, per Drag and Drop in die Coding Area gezogen werden. Zur Veranschaulichung wird der Event-Block in Abbildung 20 verwendet, welcher besonders wichtig für den Start eines Programms ist.

Abbildung 20: Event-Block in UIFlow

Um mit dem Programmieren sinnvoll starten zu können, sollte man jedoch davor noch ein die in der Folge beschriebenen Konzepte kennen.

2.2.3. Variablen

Variablen stellen sozusagen das Gedächtnis eines Programms dar. Sie können Werte speichern, die im Zuge der Ausführung eines Programms geändert bzw. verwendet werden können, und zwar durch den Aufruf des Namens der Variable. Beispiele unterschiedlicher Variablen sehen wir in der nächsten Abbildung.

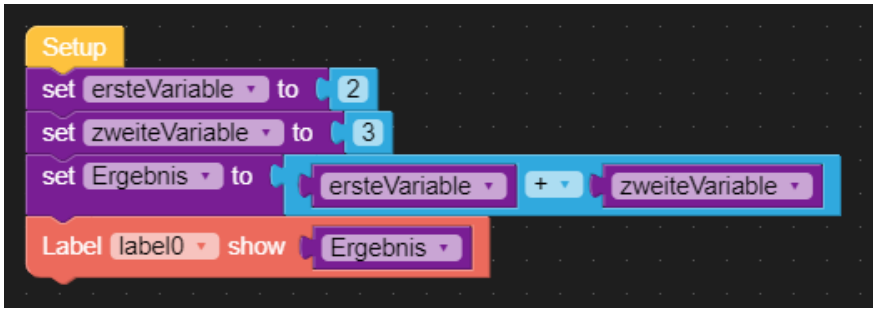


Abbildung 21: Beispiel für Variablen – Berechnung von 2 + 3 inklusive Ausgabe

2.2.4. Labels

Als Label werden die Textplatzhalter auf dem M5Core bezeichnet, mit denen man gewisse Werte oder Strings auf dem LCD-Display des M5Core ausgeben kann.

Um ein Label auf dem M5Core anzuzeigen, muss man dieses von der linken oberen Ecke in UIFlow per Drag and Drop auf die Oberfläche des M5Core ziehen, wie in der Abbildung 22 zu sehen ist.

Abbildung 22: Drag und Drop von einem Label auf den M5Core

Danach erscheint in der Auswahlleiste für die Blöcke unter UI eine Subkategorie Label. Somit kann man jetzt die Blöcke auswählen, mit denen man die Ausgabe auf dem M5Core verändern kann. Ein solcher Block wird in der Abbildung 23 gezeigt.

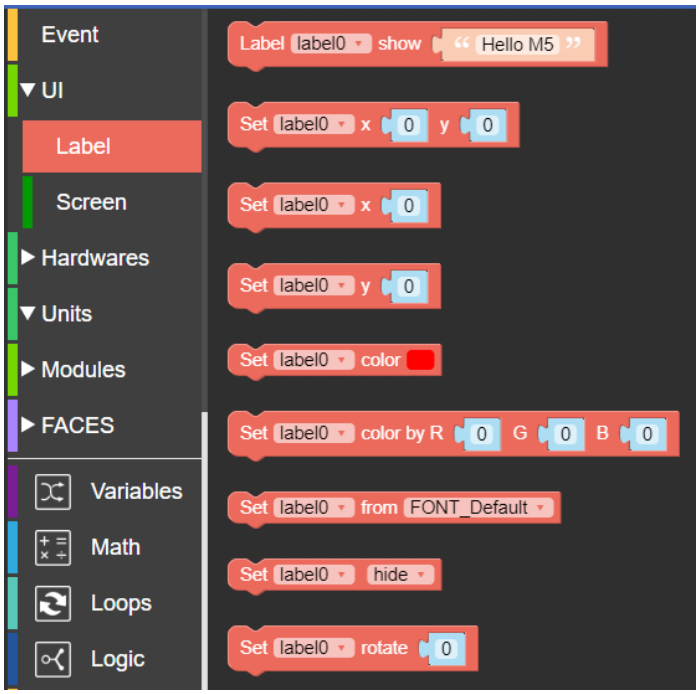


Abbildung 23: Block Ansicht für "Label"

Das Prinzip für Title, Rect, Circle und Image ist das gleiche wie für Label, mit Ausnahme der Subkategorien unter „UI“, die dann Title, Rect, Circle oder Image heißen, abhängig davon, welche Felder per Drag and Drop auf den Screen des M5Stack gezogen wurden.

2.2.5. Units

Units sind Sensoren, die man an den M5Core anschließen kann. Diese liefern Daten, die weiterverarbeite werden können. Ein Beispiel ist der Environment-Sensor, der die Umgebungstemperatur misst und weiterleiten kann.

Um eine Unit auf einfache Art und Weise in UIFlow zu verwenden, muss sie zu allererst unter Units über das „+“ Symbol, im linken unteren Eck von UIFlow hinzugefügt werden. Hierfür einfach auf das „+“-Symbol klicken, welches in der Abbildung 24 abgebildet wird.

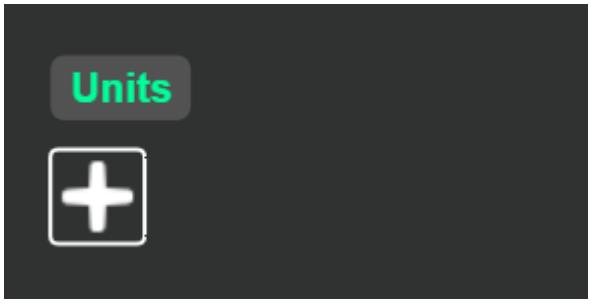


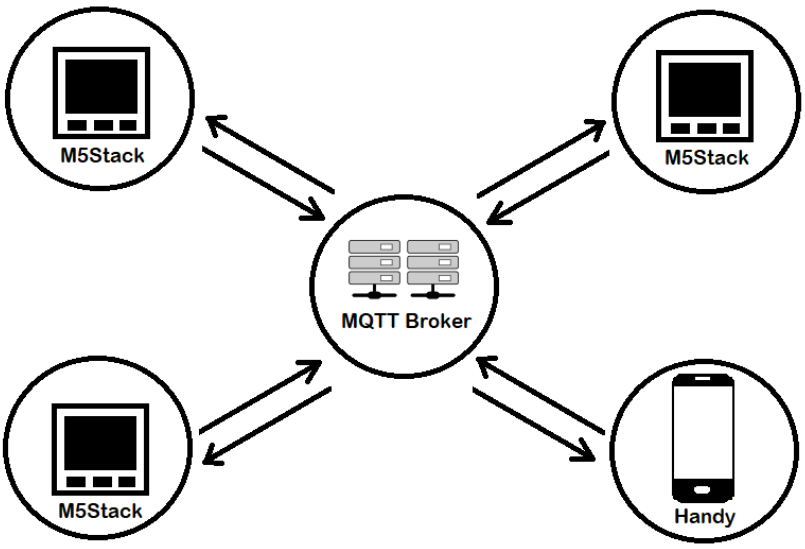
Abbildung 24: Hinzufügen einer Unit in UIFlow

3. Realisierung des Smart-Home-Systems

Nachdem alle notwendigen Werkzeuge runtergeladen worden sind und man alles korrekt installiert hat, kann man nun mit der Implementierung beginnen.

Jedoch, wie so für alle Software Projekte, muss man sich im Vorhinein überlegen, wie die ganze Architektur und Struktur aller Komponenten und Geräten am Ende aussehen soll. In einem Vorherigen Kapitel wurden bereits die „Kategorien“ vorgestellt für das Smart-Home-System. Jede Kategorie wird durch ein M5Stack Gerät repräsentiert und übernimmt eine gewisse Aufgabe im Haus. Wie erfährt jedoch ein M5Stack, was gerade bei dem anderen passiert? Welche Information benötigt eine Kategorie von den anderen, und welche soll diese an andere weiterschicken? Dies ist eines der wichtigsten Aspekte für diesen Anwendungsfall, und bevor man mit dem Programmieren beginnt, müssen zuerst diese Fragen beantwortet werden.

3.1. Kommunikation zwischen den Geräten



Die oben gezeigte Grafik beschreibt im groben wie die Kommunikation zwischen den Geräten funktioniert. Die Geräte kommunizieren nicht direkt miteinander, sondern der ganze Datenaustausch findet über den „MQTT Broker“ statt. Was genau ein MQTT Broker, bzw. was MQTT generell ist, wird im folgenden Kapitel erklärt.

3.1.1. MQTT

MQTT oder auch „Message Queuing Telemetry Transport“ ist ein Netzwerkprotokoll, welche die Übertragung von Nachrichten zwischen Geräten ermöglicht. Da es bei dem Datenaustausch um simple und kleine Datenmengen handelt, ist dieser ideal für den Internet-of-Things Bereich, da das ganze System somit auch bei einer schwachen Internetleitung funktioniert und durch die geringen Datenmengen der Austausch sehr schnell stattfindet.

3.1.2. MQTT-Broker

Wie man in der Grafik erkennt, ist in der Mitte der MQTT-Broker die zentrale Schnittstelle für alle Geräte im System. Dieser MQTT-Broker ist entweder selbst gehostet auf einem eigenen Computer oder man greift über eine Cloud-Lösung auf einen zu. Somit kann man sagen, dass ein MQTT-Broker einen Server darstellt, und alle Geräte, die mit diesem Kommunizieren, sind Clients. Für mehr Informationen, was genau Client bzw. Server sind, findet man unter folgenden Link mehr dazu: https://de.wikipedia.org/wiki/Client-Server-Modell.

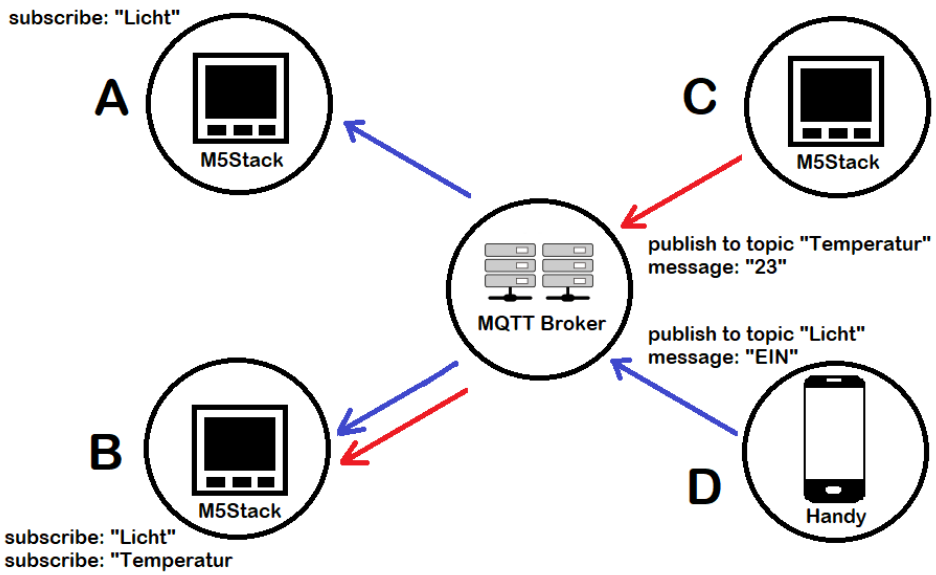
3.1.3. Publish & Subscribe

Nachdem man verstanden hat, wie die Kommunikation grob aussieht, stellt sich nun trotzdem die Frage, wie es detaillierter aussieht und wie genau man auf solche MQTT-Broker zugreift bzw. Daten an diesen schickt und von diesem bekommt.

Im Gegensatz zu herkömmlichen Netzwerkprotokollen wie TCP/IP, gibt es keine Adressen an wen Nachrichten bzw. Daten gesendet werden. Stattdessen funktioniert MQTT über ein **Publish** & **Subscribe**-Modell basierend auf **Topics**.

Topics sind Kanäle, die man abonnieren kann. Ein Client kann Nachrichten nur empfangen, wenn er einen topic abonniert hat. Schickt ein Gerät eine Nachricht, so empfangen alle Clients diese Nachricht, sofern sie die gleiche topic abonniert haben, in der sie gesendet worden sind. Dieses abonnieren ist das „subscribe“, und Nachrichten auf einem bestimmten Kanal senden ist das „publish“. Eine MQTT-Nachricht besteht aus dem Inhalt der Nachricht und dem Topic, wo die Nachricht „ankommen“ soll.

Beispiel:



An diesem Beispiel erkennt man, dass der M5Stack **C** eine Nachricht mit dem Inhalt „23“ an den topic „Temperatur“ sendet. Diese Nachricht gelangt zuerst zu dem Broker, wo sie dann nur von den Geräten empfangen wird, die topic „Temperatur“ abonniert haben, in diesem Fall M5Stack B. Die Nachricht „EIN“ vom Handy **B** wird von M5Stack **A** und **B** empfangen, da beide Geräte die topic „Licht“ abonniert haben.

3.1.4. MQTT Realisation für den Anwendungsfall

Es gibt grundsätzlich zwei Möglichkeiten, wie man zu einem MQTT-Broker gelangt bzw. wie man einen nutzen kann. Die beste und sicherste Methode wäre es, einen eigenen MQTT-Broker zu erstellen, der im Idealfall, im gleichen Haushalt als Server dient, wo die Geräte sich befinden. Das Problem: einen so zu konfigurieren, dass dieser stabil läuft und alles so funktioniert, wie es soll ist sehr zeitaufwändig und man braucht einen Computer der als Server ständig eingeschaltet ist. Die andere Variante ist es, einen externen MQTT-Broker zu nutzen auf dem man über das Internet zugreift. Hier muss man für private Broker, so dass der Zugriff auf diesen nur einem selbst gewährleistet wird, etwas zahlen. Jedoch für Testzwecke, und auch für unseren Anwendungsfall rein ein öffentlicher MQTT-Broker. Diese sind ebenfalls externe, über das Internet aufrufbare Broker, die von jedem Client zugänglich sind. Man kann durch die selbstgewählten topics dafür sorgen, dass ein Zugriff kaum möglich ist, bietet jedoch keinen 100% Schutz, reicht für unseren Fall komplett aus.

Für unseren Fall verwenden wir den öffentlichen Broker von „HiveMQ“. Die Zugangsdaten für diesen sind

Broker:	`broker.hivemq.com`
TCP Port:	`1883`
Websocket Port:	`8000`

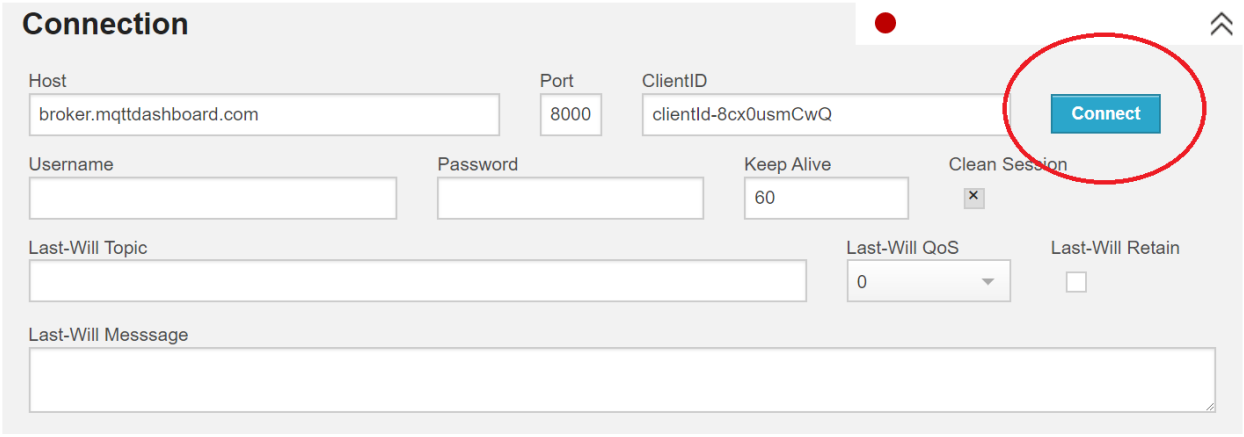
Diese Daten werden später wichtig, da wir unsere M5Stacks und Geräte so konfigurieren, sodass sie sich mit diesem Broker verbinden und an diesen die Nachrichten senden und von diesem empfangen.

3.1.5. MQTT Webclient

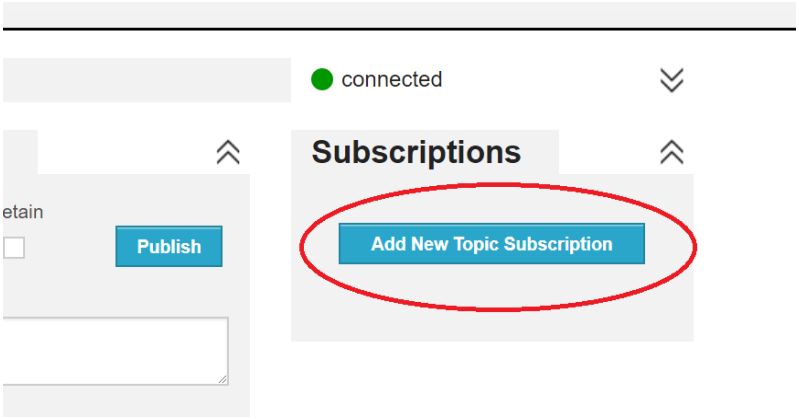
Unter diesem Link gelangt man zu einem WebClient. Mit diesem kann der eigene PC bzw. der Browser als Client fungieren, wo man ebenfalls topics abonnieren kann. Dies stellt somit ein gutes Werkzeug dar, um zu testen, ob die Kommunikation der eigenen Implementation funktioniert. Obendrauf kann man dies auch nutzen, um das Smart-Home-System zu steuern, da man auch Nachrichten damit senden kann. Bietet somit eine sehr einfache Alternative zu einer mobilen Applikation auf dem Handy.

Um mit MQTT generell vertraut zu werden, versuchen wir nun über diesen WebClient mehrere Topics zu abonnieren, und eine Nachricht an einer dieser zu senden und somit wiederum diese zu empfangen.

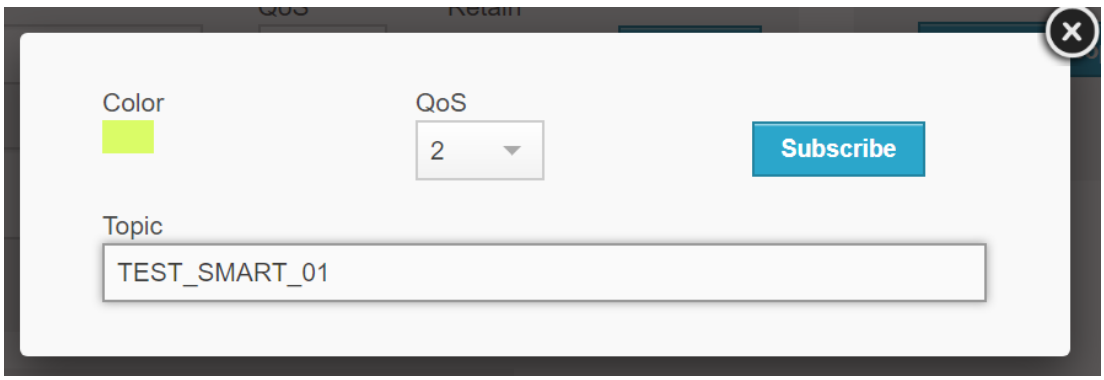
Schritt 1: Sich mit dem Client zu verbinden -> Beim Aufruf der Seite genügt es auf „Connect“ zu drücken, sodass der rote Punkt grün wird. Die anderen Felder sind vorerst nicht von Bedeutung.



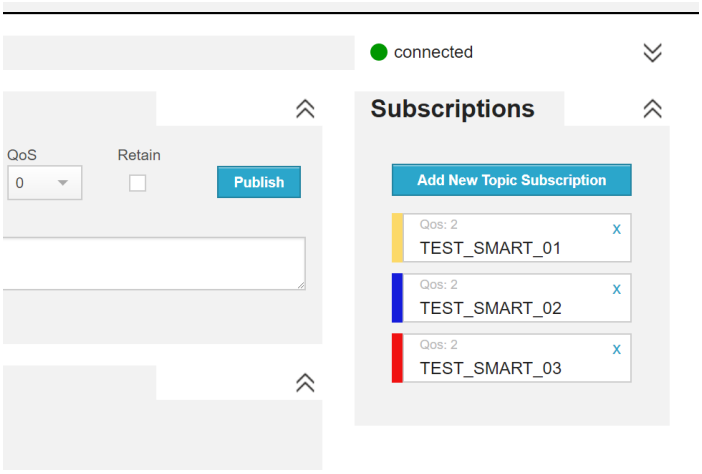
Schritt 2: Bei einem Topic „subscriben“ indem man auf „Add New Topic Subscription“ drückt



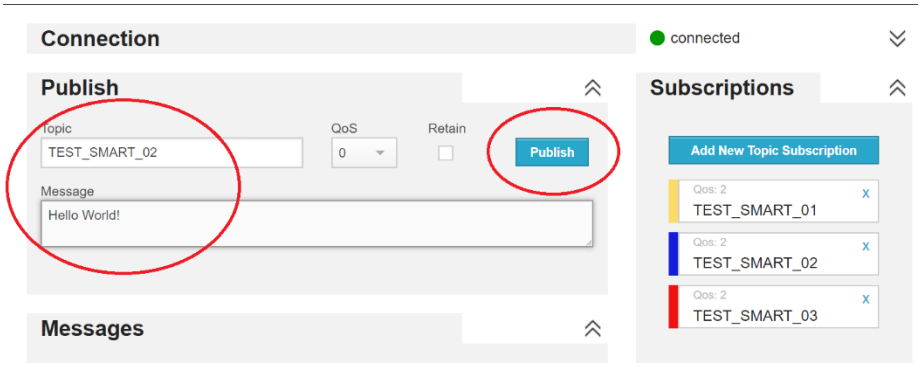
Schritt 3: einen eigenen Topic-Namen wählen und auf „Subscribe“ drücken.



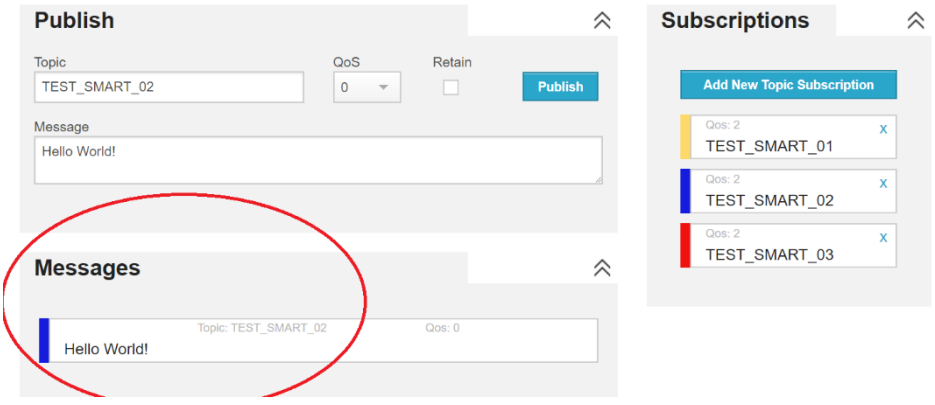
Schritt 4: gleich mehrere Topics subscriben, damit man eine bessere Übersicht der Resultate hat.



Schritt 5: Eine Nachricht an eine der abonnierten Topics senden. Nach dem Eingeben der Nachricht (Topic + Message) dann auf „Publish“ drücken



Schritt 6: Die Nachricht mit korrektem Inhalt und Topic erhalten. Ersichtlich durch die Farbe und auch durch die Topicbezeichnung bei der Nachricht selbst.



Diese kurze Demonstration zeigt uns, wie MQTT in der Praxis funktioniert. Dieses Wissen bezüglich der Kommunikation wenden wir nun für unseren Fall an, in dem wir im nächsten Abschnitt unsere M5Stacks mit diesem Broker verbinden.

3.2. Programmieren am M5Stack

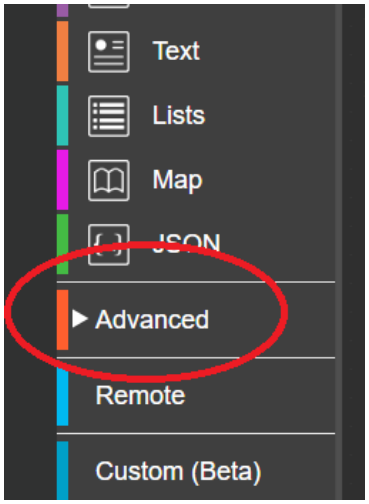
In diesem Kapitel werden die vier bereits vorgestellten „Kategorien“ des Smart-Home-Systems realisiert. Es ist zu erwähnen, dass, wenn man dies selbst machen will, nicht auf diese 4 Kategorien beschränkt ist und man alle Möglichkeiten und Kombinationen von Units und anderen Komponenten verwenden kann.

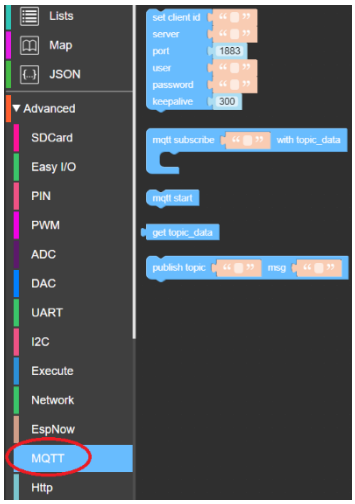
Als Überleitung vom letzten Kapitel wird das erste Ziel sein, sich mit einem M5Stack mit dem öffentlichen MQTT Broker von „HiveMQ“ zu verbinden.

3.2.1. Erstellung einer Verbindung zwischen M5Stack & MQTT Broker

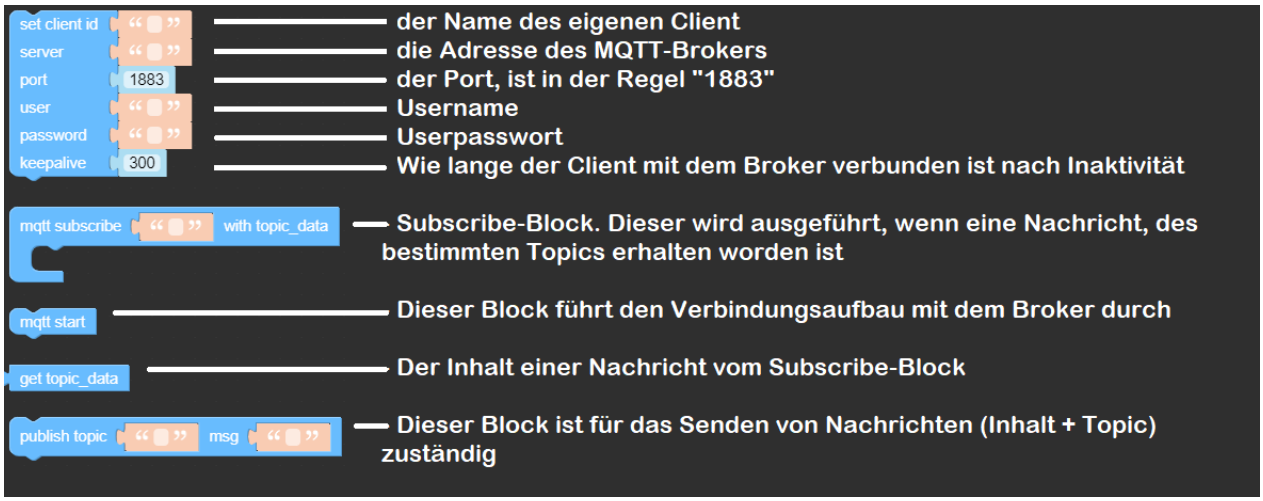
Wie im Kapitel 2.2 bereits vorgestellt, verwenden wir für die Implementierung am M5Stack die Programmierumgebung UIFlow.

Unter den Block „Advanced“ findet man die Blöcke für die MQTT Schnittstelle

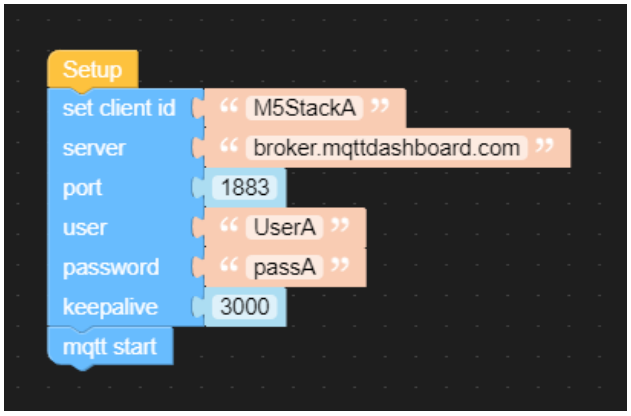




Unter der folgenden Abbildung sind die Funktionen der einzelnen MQTT-Blöcke beschrieben, mit denen wir uns gleich daraufhin beschäftigen werden.

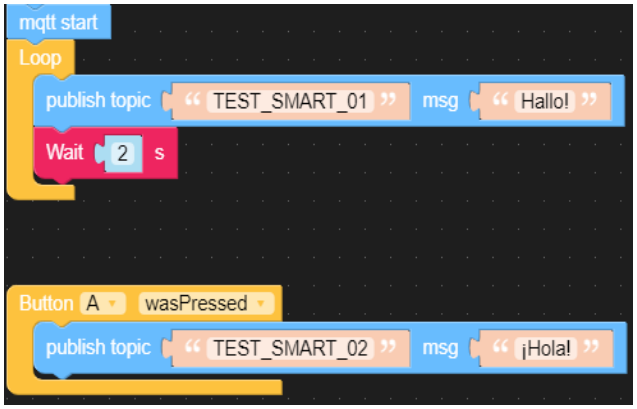


Zum Testen der MQTT Protokolls am M5Stack verwenden wir zuerst, wie für alle Programme, den Start-Block. Dieser wird am Start vom M5Stack ausgeführt und zusammen mit den Initialisierungsblock von MQTT soll die Verbindung mit dem Broker hergestellt werden und sieht wie folgt aus:



Nach dem Setup werden die Verbindungsinformationen für den Broker angegeben und anschließend soll die Verbindung mit dem „mqtt start“ Block hergestellt werden.

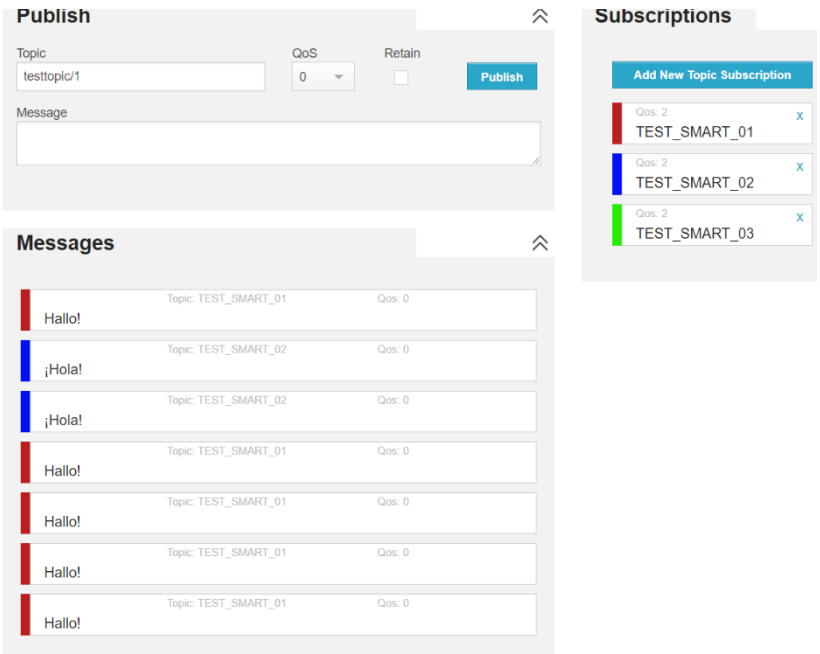
Hinweis: Da der öffentliche Broker keine User speichert und keine Passwörter abfragt, ist die Wahl des Namens für die ID usw. nicht von Bedeutung



Erweitert man nun den Setup-Block um diesen loop, so wird jede zweite Sekunde, eine Nachricht mit dem Inhalt „Hallo!“ an topic „TEST_SMART_01“ an den Broker gesendet

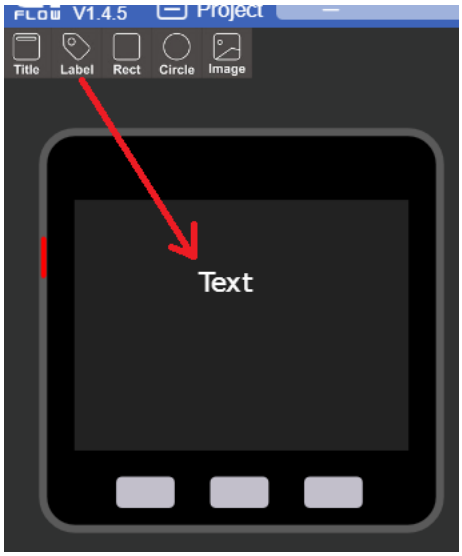
Bei diesem Block wird eine ähnliche Nachricht nur dann gesendet, wenn am M5Stack die Taste A gedrückt wird.

Dies kann man über den bereits genutzten WebClient testen und sollte nach erfolgreicher Konfiguration des M5Stacks ungefähr so aussehen:

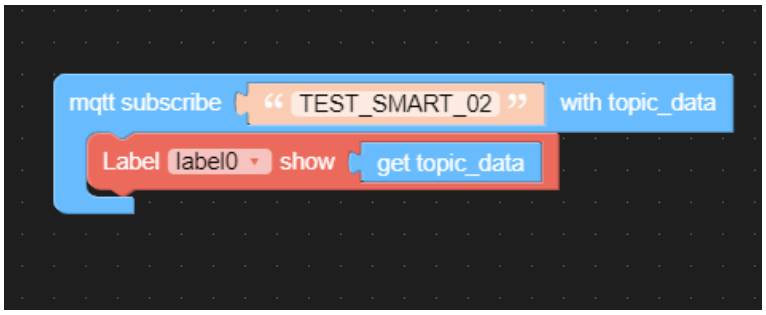


Nun hat man erfolgreich Nachrichten an den Broker gesendet. Der nächste Schritt ist nun das Empfangen von Nachrichten. Für das folgende Beispiel soll eine MQTT Nachricht, den Text am Bildschirm des M5Stacks verändern.

Zuerst muss ein „Label“ erstellt werden. Der Default Name des ersten erstellten Labels ist in der Regel „label0“.



Nun soll der M5Stack einen Topic subscriben, um über diesen dann Nachrichten zu empfangen.



Den Subscribe-Block kennen wir bereits aus der Abbildung weiter oben. Im oberen leeren Feld gibt man den Topic an, welchen man abonnieren möchte. Jeder Block, der an diesen angehängt wird, wird immer dann ausgeführt, sobald der MQTT-Broker eine Nachricht mit dem entsprechenden Topic weiterleitet und diese vom M5Stack empfangen wird.

Der „topic-data“-Block ist der Inhalt dieser Nachricht. Bei diesem Beispiel wird ein Textfeld, welches am M5Stack angezeigt wird, immer dann verändert, wenn man eine Nachricht bekommt mit dem entsprechenden Topic. Dies kann man wiederum mit dem WebClient testen, indem man dort eine Nachricht mit dem korrektem Topic published bzw. sendet. (Schritt 5 bei Kapitel 3.1.5 MQTT WebClient)

Sobald das Senden und Empfangen von MQTT Nachrichten funktioniert, ist man nun bereit mit der Implementierung der Smart-Home Kategorien anzufangen.

3.2.2. Smart-Home Implementierung am M5Stack

3.2.2.1. Home

Die erste Kategorie, die wir nun implementieren wollen, ist das Home-System, welche über einen Sensor die Temperatur im Zimmer abfragt und auch für die Beleuchtung sorgt.

Dafür fügen wir nun die entsprechenden Units in UIFlow ein. Dies wären die **ENV** und die **RGB**-**Unit**

Nun gilt es zu überlegen, was dieses System alles können und machen soll

- Es soll die Temperaturwerte vom Sensor an den Broker senden. Was genau wir damit machen, bzw. welches Gerät dies dann benötigt schauen wir uns zu einem späteren Zeitpunkt an.
- Es soll „Wunschtemperatur-Werte“ empfangen, und so feststellen können, ob es momentan im Zimmer zu heiß oder zu kalt ist.
- Es soll Nachrichten empfangen, mit denen die Beleuchtung reguliert wird.

Somit sind 3 Variablen erforderlich, bzw. 3 Topics mit denen unser M5Stack kommuniziert. Bennen wir die nach einem gewissen Schema, so dass sie klar erkennbar sind wofür sie stehen, aber trotzdem einzigartig, sodass keine andere Person diese momentan benutzt, da wir mit einem öffentlichen Broker kommunizieren. Bei topics wie nur „temperature“ würden wir mit hoher Wahrscheinlichkeiten Nachricht von Drittpersonen empfangen. Als Beispiel könnten wir „M5/jku“ als Präfix verwenden, damit wir unsere Variablen somit zu unserem Projekt zuordnen.

- **M5/jku/temp_actual** … Temperaturwerte des Sensors
- **M5/jku/temp_set** … Wunschtemperatur
- **M5/jku/brightness** … Beleuchtung

Hinweis: Topics sind mit „/“ ebenfalls wie Ordner verschachtelt. Würde man den Topic „M5/jku/#“ subscriben, würde man ALLE Nachrichten empfangen, die mit „M5/jku/“ beginnen.

Beginnen wir mit dem Grundaufbau im UIFlow.

- Allgemeiner Startblock
- MQTT Initialisierung (Informationen zu dem Broker)
- MQTT-Start

Für das Senden der Temperaturwerte benötigen wir nun den „publish topic“-Block. Darin sollen die Werte an den Topic geschickt werden. Aber wann? Wie oft? Das bleibt einem selbst überlassen, aber ich möchte durchgehend diese Information haben, deswegen baue ich diese Funktionalität in eine Schleife, die alle 2 Sekunden ausgeführt wird, ein, damit ich immer die aktuellen Werte habe.

Mit dem „Get env0 Temperatue“-Block erhält man die Temperaturwerte. Der „convert to int“ Block konvertiert mir die erhaltenen Werte in schöne Ganzzahlen. Mit dieser Schleife ist das Senden der Sensordaten auch schon bereits getan.

Würde man mehrere solche M5Stacks in verschiedenen Zimmer aufstellen, so müsse man noch unterschiedliche Topics für die Temperaturwerte konfigurieren, so dass man für jedes Zimmer die korrekten Werte bekommt, ansonsten würden mehrere ENV-Units eventuell unterschiedliche Temperaturwerte messen

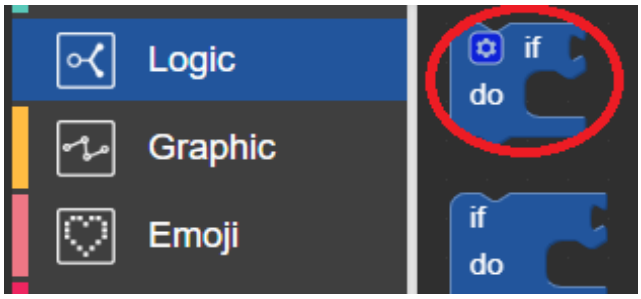
Als nächstes kümmern wir uns um die Beleuchtung. Das gute an dem RGB-Unit am M5Stack ist, dass es nicht nur an- und ausgeschaltet werden kann, sondern man auch die Lichtstärke regulieren kann. Deswegen muss man überlegen, wie soll diese Lichtstärke gesteuert werden? Der Drehschalter-Unit ANGLE würde sich da optimal dafür anbieten. Wenn man mit diesem etwas Vertraut ist, erkannt man, dass die Werte, die diese Unit ausgibt, von ungefähr 1-1024 gehen. Die Lichtstärke hingegen am RGB Unit geht von ungefähr 1-300 (Das bedeutet, dass sie bei ca. 300 ihr Maximum erreicht hat). Man könnte alle ANGLE Werte einen Lichtstärkewert zuweisen, aber das wäre vom Aufwand viel zu hoch und nicht zielführend für eine Einführung. Beschränken wir uns auf gewisse Beleuchtungsstärkezustände und kategorisieren wir diese

M5/jku/brightness	= 0	…	Kein Licht
	= 1	…	Schwach
	= 2	…	Mittelstark
	= 3	…	Starke Beleuchtung

Das Setzen dieser Werte über den MQTT-Broker muss sich unser jetziges System nicht kümmern, dafür gibt es die „Control-Kategorie“ die später implementiert wird. Jedoch können wir jetzt schon die Logik für die Beleuchtung programmieren, sollte eine solche Nachricht empfangen werden.

Der erste und wichtigste Block dafür ist der subscribe-Block

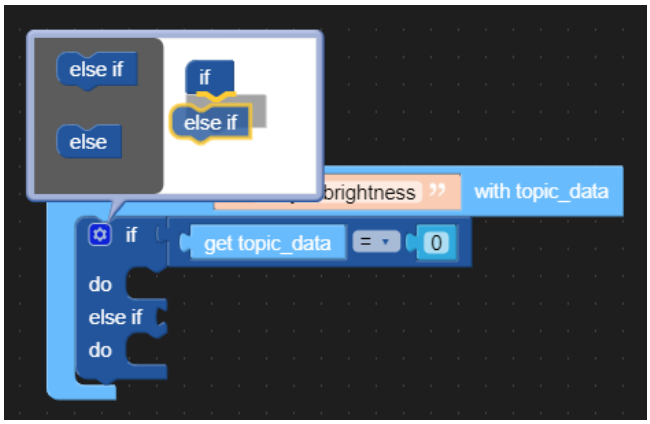
Da unsere Werte im Bereich 0-3 sind, soll zuerst beim Empfangen der Nachricht abgeprüft werden, um welchen von diesen es sich handelt und anhand von dieser reagiert werden und entsprechend die Lichtstärke gesetzt werden.



Für eine solche Überprüfung eignet sich der **„if-do“-Block** ideal, da sich dieser beliebig erweitern lässt.

Nun geht es darum, die Bedingung abzufragen, wann die Lichtstärke verändert werden soll, und dies funktioniert mit einem simplen **Vergleichsoperator**.

Wann soll das Licht ausgeschalten werden? Wenn die Nachricht mit dem Inhalt „0“ empfangen wird



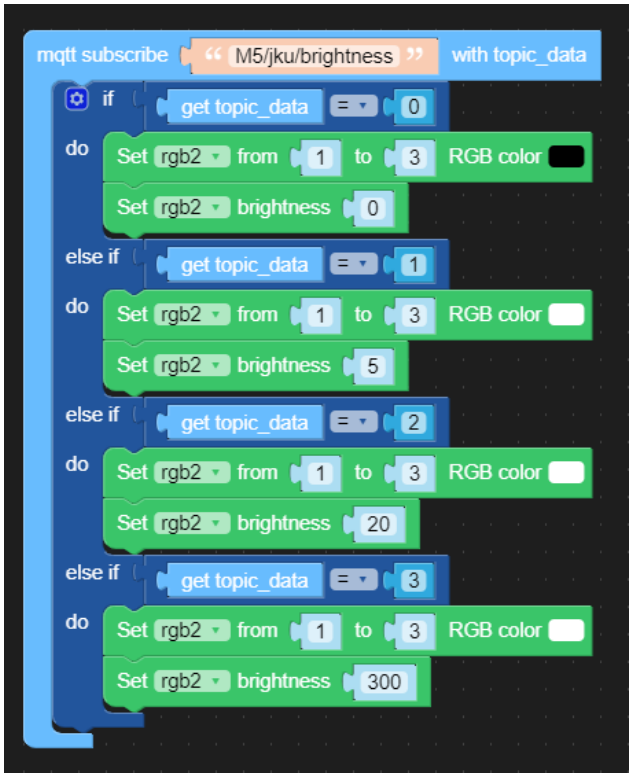
Die anderen Fälle (wann der Inhalt 1,2 oder 3 ist) können wir abbilden, indem wir beim „if“ das blaue Zahnrad auswählen, und weitere „else-if“ Bedingungen hinzufügen

Am Ende sollen somit 4 Überprüfungen stattfinden.

Hinweis: *Sollte man noch eine „else“ Überprüfung am Ende machen? Hängt davon ab, was man mit Nachrichten machen will, die z.B. keine Zahlen sind. Für unseren Anwendungsfall werde ich diese Nachrichten ignorieren und somit kein „else“ hinzufügen.*

Jetzt, da wir die korrekte Überprüfung erstellt haben, soll nun die Lichtstärke angepasst werden.

Mit diesen 2 Blöcken können wir die Farbe und Intensität des RGB-Units steuern.



So sieht am Ende die Steuerung der Beleuchtung aus. Die Werte für Intensität (brightness) können natürlich beliebig selbst gewählt werden sowohl auch eine Wunschfarbe.

Bei der ersten Überprüfung wird das Licht auf die Farbe „Schwarz“ gestellt, weil das gleich zu setzen ist mit keiner Beleuchtung.

Für die darauffolgenden Überprüfungen wird in meinem Fall immer die Farbe weiß mit unterschiedlicher Lichtstärke genommen.

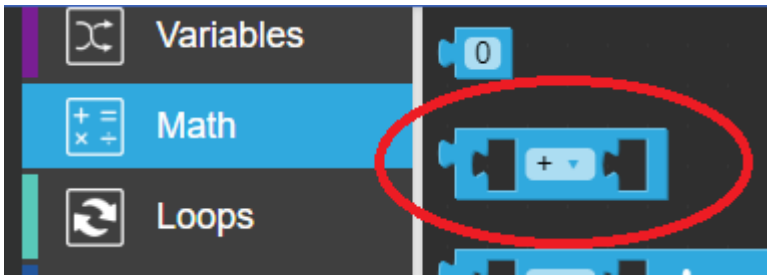
Die wichtigsten Funktionen besitzt unser erstes Smart-Home System schon. Es sendet Nachrichten mit den Temperaturwerten vom ENV-Unit und es liest und verarbeitet Nachrichten für die Beleuchtung. Was wir noch für diese Smart-Home-Kategorie hinzufügen können, ist die Überprüfung, ob die momentane Temperatur im Zimmer passt oder zu kalt, bzw. zu heiß ist. Die Wunschtemperatur ist, ebenfalls wie die Beleuchtung, eine Variable, die von einem anderem System erst gesendet wird.

Somit müssen wir ähnlich wie für die Beleuchtung einen Subscribe-Block hinzufügen, der anhand von der Wunschtemperatur etwas macht.

Für unseren Anwendungsfall soll der M5Stack selbst rot leuchten, wenn es zu heiß ist, und blau leuchten, wenn es zu kalt ist.

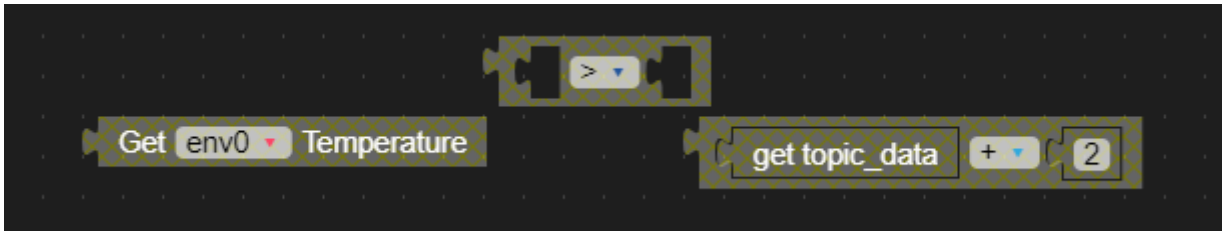
Dafür benötigen wir fürs Erste eine Abfrage auf den Wert selbst. Da eignet sich, wie im vorherigen Beispiel, der erweiterbare **„if-do“-Block**.

Bei welchem Wert soll der M5Stack aber nun farbig leuchten? Wenn die Temperatur nur 1 Grad über oder unter der Wunschtemperatur ist für mich etwas zu strikt. Ich bevorzuge es, wenn es zumindest 2 oder 3 Grad über oder unter dem Wunschwert ist.



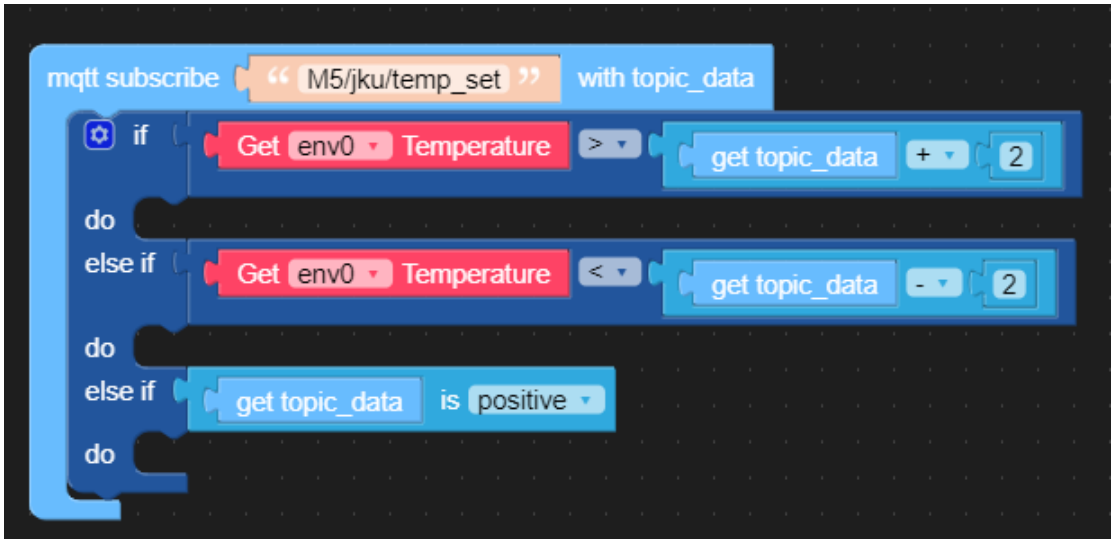
Um der Wunschtemperatur einen Spielraum zu geben, verwenden wir den **Additions-Block** von „Math“ und fügen zu der die Wunschtemperatur und eine Zahl hinzu.

Mit dem Vergleichsoperator kann man somit die Ist-Temperatur mit der Soll-Temperatur vergleichen.



(Hinweis: Diese Blöcke sind ausgegraut, weil sie momentan im Code nicht zusammengebaut sind, sie aber leichter in ihren Einzelteilen zu zeigen sind)

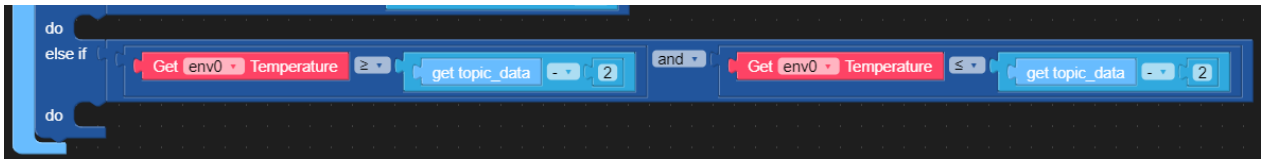
Ein fertiger Block zur Abfrage bzgl. Wunschtemperatur könnte so aussehen:



Das erste „if“ ist analog zur Abbildung darüber, sollte die tatsächliche Temperatur mehr als 2 Grad über der Wunschtemperatur sein. (zu heiß)

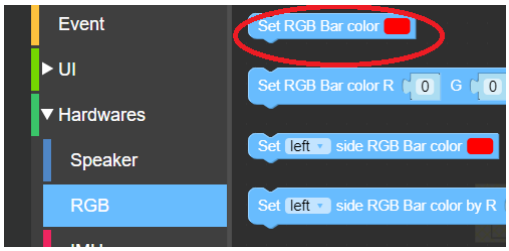
Das zweite „if“ ist für die andere Richtung, sollte die tatsächliche Temperatur weniger als 2 Grad unter der Wunschtemperatur sein. (zu kalt)

In den oberen Zwei Bedingungen soll der M5Stack entweder rot oder blau leuchten. Der optimale Zustand ist, wenn die Temperatur mit der Wunschtemperatur übereinstimmt, welches nun gilt zu überprüfen. Man könnte feststellen, ob die Temperatur in einem Wertebereich ist. Dies würde so aussehen.



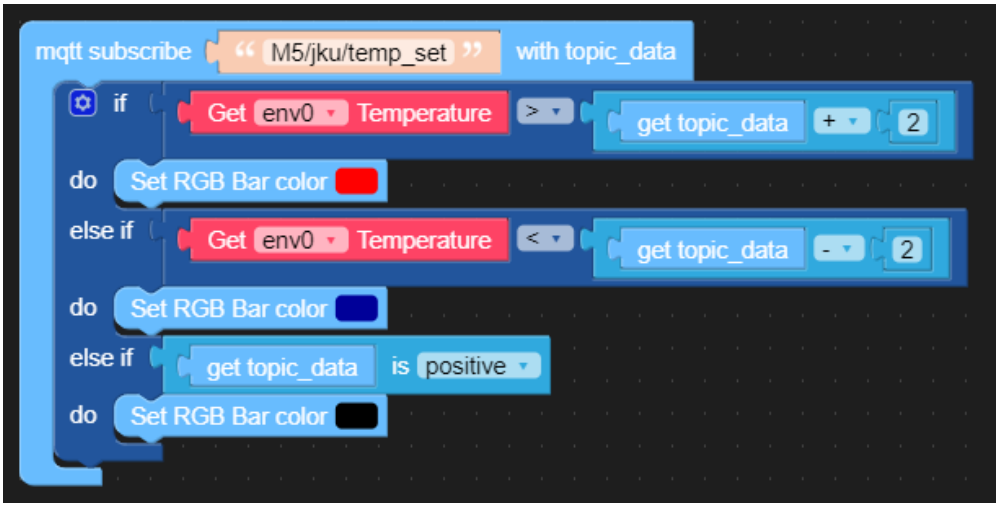
Auch wenn diese Bedingung funktioniert, wirkt sie leider etwas chaotisch. Nehmen wir an, die Wunschtemperatur ist 25 Grad. Dann wird bei dieser Bedingung im Grunde nur überprüft, ob die Ist-Temperatur über 23 und unter 27 Grad ist, oder einfacher ausgedrückt, ob sie zwischen 23 und 27 liegt.

Bei der Abbildung weiter oben wird diese Funktion verwendet. Sie prüft, ob die Wunschtemperatur eine positive Zahl ist. Dies würde für unseren Fall auch das gleiche Ergebnis liefern, da die anderen Zahlenbereiche (also z.B. unter 24 und über 26) bereits abgedeckt sind.



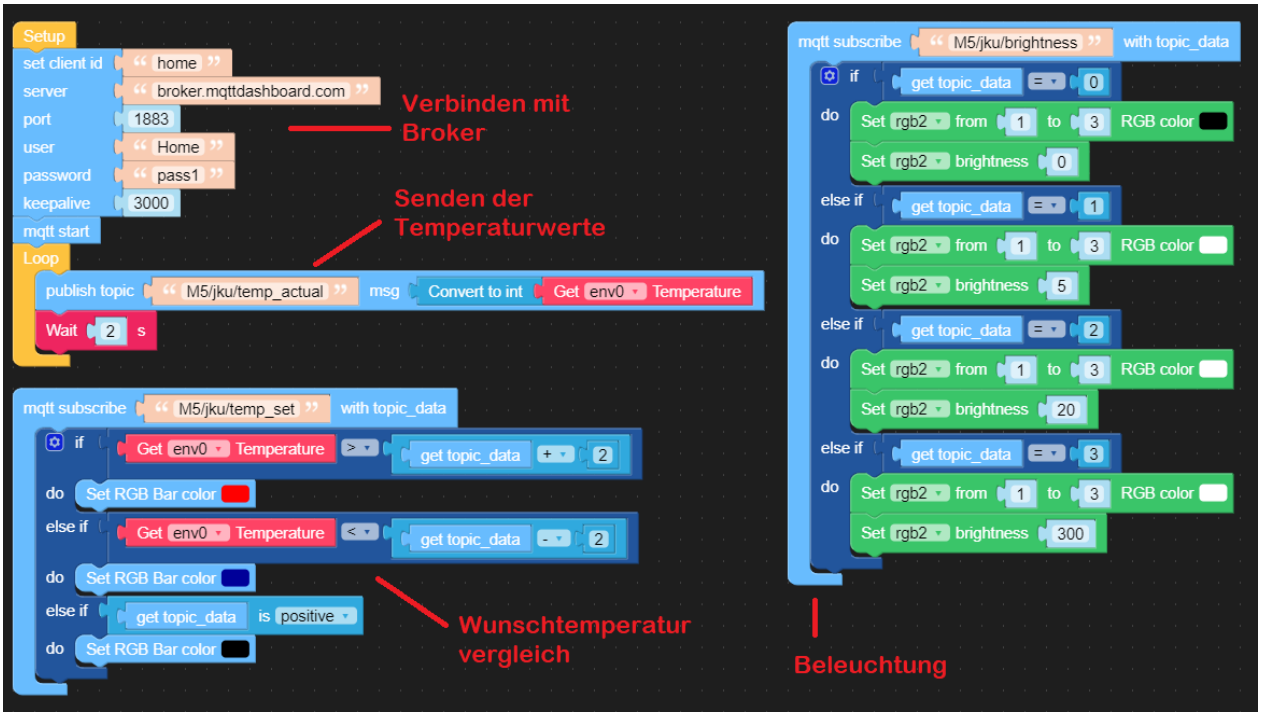
Zu guter Letzt gilt es jetzt das Leuchten des M5Stacks zu realisieren. Die Funktion dafür ist unter Hardware > RGB > „Set RGB bar color“ zu finden

Ein fertiger Subscribe-Block zum Erkennen, ob es im Zimmer zu heiß oder zu kalt ist, könnte somit so aussehen. Wenn man Geräte hätte, die die Raumtemperatur beeinflussen können (z.B. Ventilator, Heizstrahler usw.) und auch über eine MQTT Schnittstelle verfügen, könnte man genau in diesem Code eine Nachricht pushen (senden), damit diese aktiviert werden.



(Hinweis: die Farbe schwarz ist wieder gleich zu setzen wie mit keiner Beleuchtung)

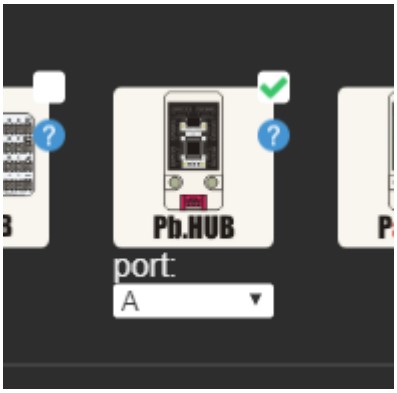
Die fertige „Smart-Home-Kategorie: Home“ sieht am Ende so aus



3.2.2.2. Plant

Wie bei der Home-Kategorie, überlegen wir uns zuerst, welche Units benötigt werden, und welche Variablen diese Kategorie sendet und empfangen soll.

Grundsätzlich geht es darum, die Pflanze zu beobachten, bzw. zu erkennen, ob die Erde der Pflanze zu trocken ist, oder sie zu wenig Licht bekommt. Dafür benötigen wir die **EARTH-Unit** und die **LIGHT-Unit**.



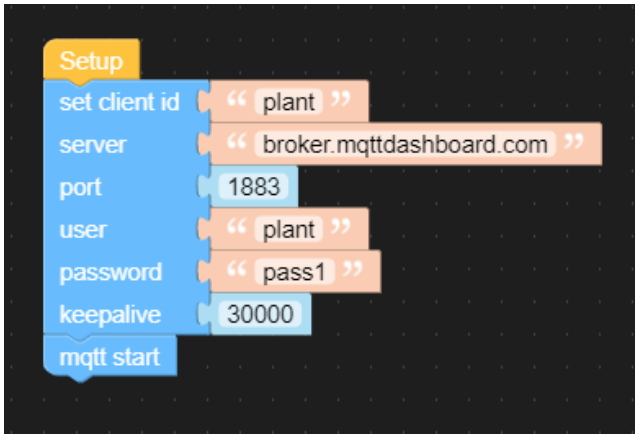
Das Problem dabei**:** Beide Units benötigen den Port-B (schwarz). Zum Glück gibt es dafür die **Pb.HUB-Unit**, die ermöglich, mehrere Units mit dem Port-B Anschluss an einem M5Stack anzuschließen.

Das hat zur Folge, dass man die anderen 2 Units nicht in UIFlow hinzufügen muss und man die Werte dieser Units über die Pb.HUB Schnittstelle erhält.

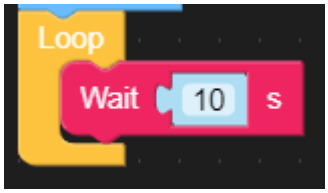
Wie sieht es nun mit den Topics aus. Welche Nachrichten benötigt man, welche soll man weiterschicken? Für unseren Anwendungsfall benötigen wir keine Nachrichten von anderen Geräten, jedoch wollen wir den Zustand der Pflanze weitergeben können. Deswegen werden keine Subscribe-Blöcke benötigt, sondern nur Publish-Blöcke.

Die zu verwendeten Variablen sind somit:

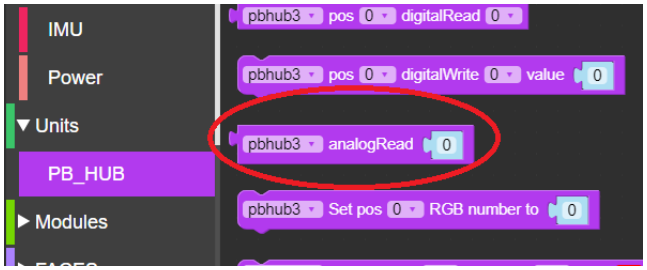
- **M5/jku/humidity** … Ob die Erde ausreichend bewässert ist
- **M5/jku/light** … Ob die Pflanze ausreichend Licht bekommt



Der Anfang ist so ziemlich gleich für alle M5Stack mit MQTT Implementierungen in UIFlow, es wird der gleiche MQTT Verbindungsblock benötigt

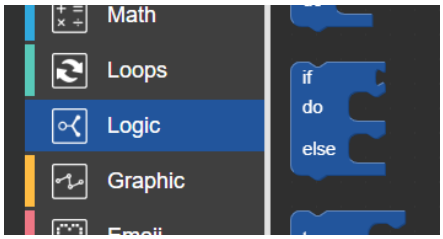


Bevor wir nun mit der Logik der Units anfangen, muss man im Vorhinein überlegen, wann soll eine Nachricht gesendet werden? Ich möchte alle paar Sekunden den Status der Pflanze senden. Deswegen benötigen wir eine Schleife

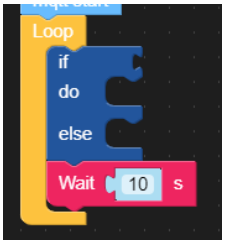


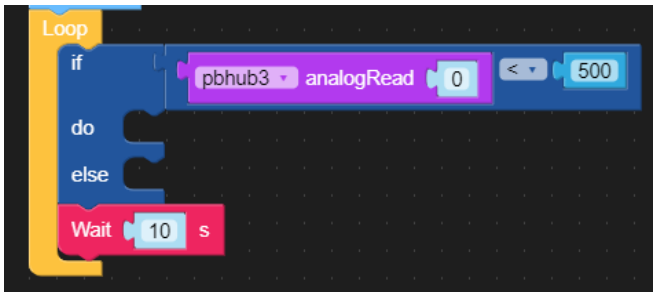
Den Status der Pflanze ermitteln wir über die Units, besser gesagt über den Pb.HUB, mittels dem analogRead-Block. Hier ist zu beachten, an welchen der 6 Ports man die Unit ansteckt. (Am Pb.HUB selbst, ist die Zahl gekennzeichnet)

Wenn man die LIGHT-Unit an den, Pb.HUB ansteckt, und dessen Werte sich über dieses „AnalogRead“ anschaut (z.B. über eine Label Ausgabe am M5Stack Bildschirm), erkennt man, dass starkes Licht zu einem niedrigen Wert führen, und hohe Werte bedeuten somit Dunkelheit. Für meine Pflanze zuhause möchte ich, dass die „Lichtwerte“ unter 500 sind, alles darüber wäre zu dunkel.

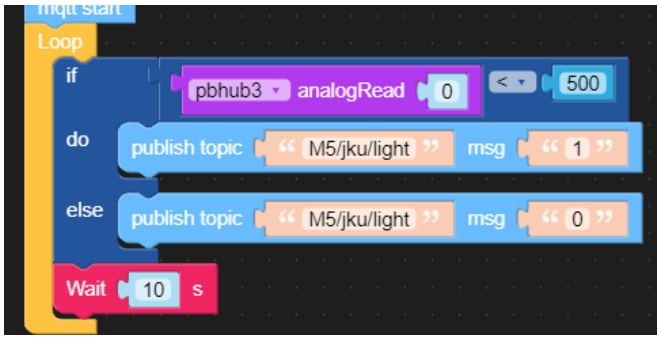


Deswegen beginnen wir wieder mit einer Bedingung, die man abfragen muss. Da es nur ein richtig oder falsch gibt, genügt hier eine einfache „if-do-else“-Schleife

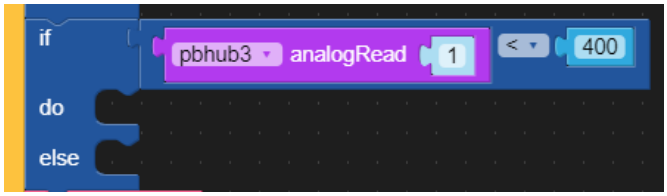




Die Abfrage selbst beim „if“ erfolgt über den Vergleichsoperator, den wir schon vorm vorherigen Kapitel kennen. Hier den AnalogRead abprüfen, ob der Wert unter 500 ist.



Im „do“-Teil soll die Nachricht gesendet werden, dass die Lichtverhältnisse für die Pflanze passen. Dies kann beliebig gewählt werden, für mich ist es einfacher zu halten mit „1“ für ok und „0“ für dunkel.

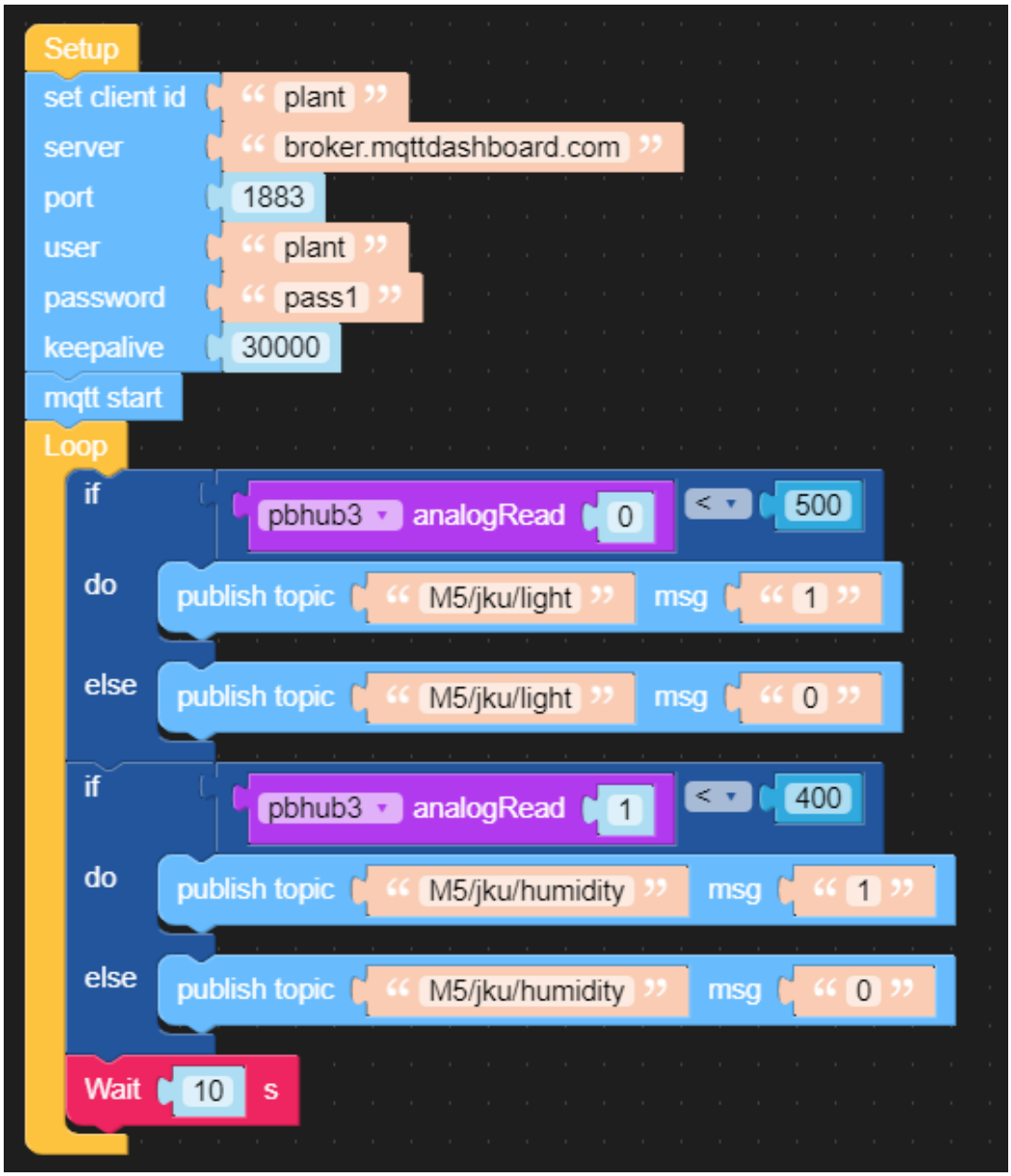


Für die Feuchtigkeit der Erde gehen wir analog vor und erstellen einen „if-do-else“ Block mit einem Vergleichsoperator mit dem AnalogRead. Beim Herumtesten fiel mir auf: Sehr trockene Erde hat Werte über 400.

(Frage: Warum zwei „if-do-else“-Blöcke und nicht gleich einen erweiterbaren „if-do“-Block wie bei der Home-Kategorie?

Antwort: Bei einem erweiterbaren „if-do“-Block kann maximal nur ein „do“-Bereich ausgeführt werden. In unserem Beispiel müssen wir jeweils für das Licht und für die Feuchtigkeit den Status abgeben, deswegen sind 2 gesonderte „if“ notwendig)

Hiermit sind wir auch schon fertig mit der M5Stack Implementierung für die Pflanze. Der momentane Zustand wird somit alle 10 Sekunden an den Broker gesendet. Was jedoch nicht beachtet wird, sind Tag- und Nachtzeiten. Das auf Nacht die Lichtwerte nie passen, ist selbstverständlich. Man könnte eventuell eine Überprüfung machen, dass diese Werte nur zu gewissen Uhrzeiten (z.B. Tagsüber) geschickt werden oder man summiert pro Tag alle diese „1“ und „0“ und findet heraus, ob die Pflanze über den ganzen Tag hinweg genug Licht bekommen hat. Aber für eine Demonstration genügt dies und der fertige Code sieht so aus:



3.2.2.3. Security

Da wir schon bereits von den vorherigen Kategorien genug Erfahrung gesammelt haben, können wir uns auf die wichtigen Schritte nun fokussieren für dieses System.

Welche Units?

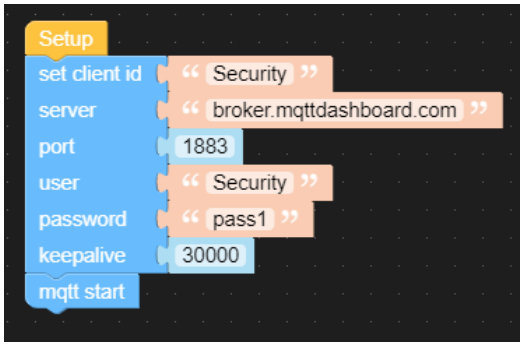
Für unser Sicherheitssystem wollen wir sicherstellen, dass niemand das Haus unbefugt betritt. Dafür benötigen wir den Bewegungsmelder-Unit **„PIR“** und auch eine Laserschranke mit dem **Laser TX**- und **Laser RX-Unit.** Ähnlich wie beim vorherigen Beispiel, sind das alle Units mit einem Port-B Anschluss, und somit ist es nicht möglich, alle an einem M5Stack anzuschließen, außer man benutzt einen Pb.HUB. Deswegen sind die Units, die man in UIFlow die **PIR-Unit** und der **Pb.HUB.**

Welche Variablen / Topics?

Mit dem Bewegungsmelder wollen wir eine Nachricht an den Broker senden, wann dieser eine Bewegung im Haus registriert. Außerdem soll auch die Laserschranke erkennen, wann diese unterbrochen worden ist. Vom Broker selbst, bzw. von anderen Geräten werden keine Informationen bzw. Nachrichten benötigt.

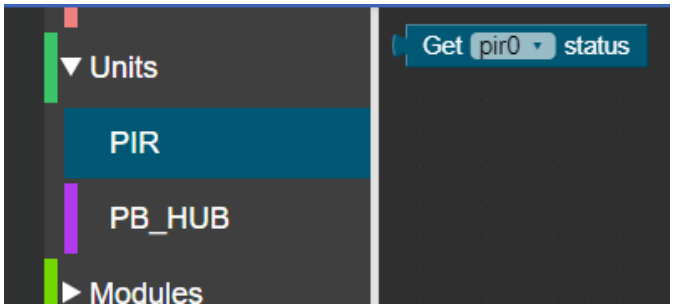
Die zu verwendeten Variablen sind somit:

- **M5/jku/laser** … Wann die Laserschranke unterbrochen wird
- **M5/jku/motion** … Wann eine Bewegung im Haus erkannt wird



Den Anfang macht wieder der MQTT-Initialisierungsblock für die Verbindung.

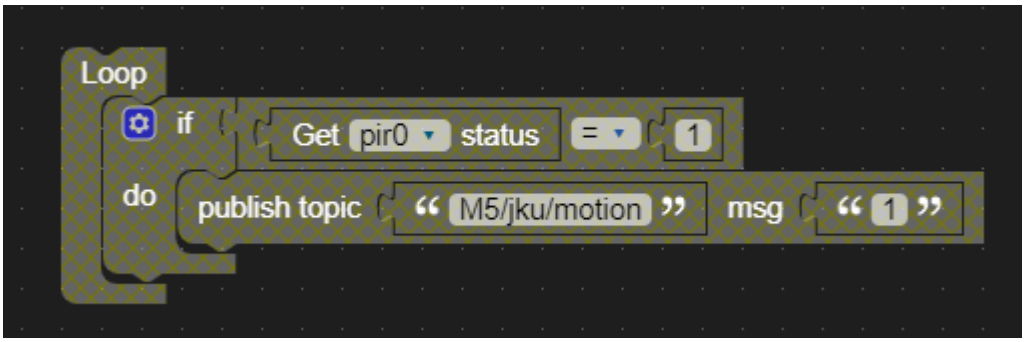
Nun stellt sich die Frage, wann soll immer eine Nachricht gesendet werden? Im vorherigen Beispiel haben wir eine Schleife erstellt, die alle paar Sekunden den Status der Units sendet. Für den Bewegungsmelder und für die Laserschranke ist es grundsätzlich nicht besonders interessant, wenn man alle paar Sekunden den Status erfährt, da sich im Normalfall kaum etwas tut. Daher würde es mehr Sinn machen, wenn man nur dann eine Nachricht sendet, wenn eine Änderung stattfindet.



Fangen wir mit dem Bewegungsmelder PIR an. Wird eine Bewegung erkannt, so ist der Wert „Get pir status“ dann „1“, andernfalls „0“.

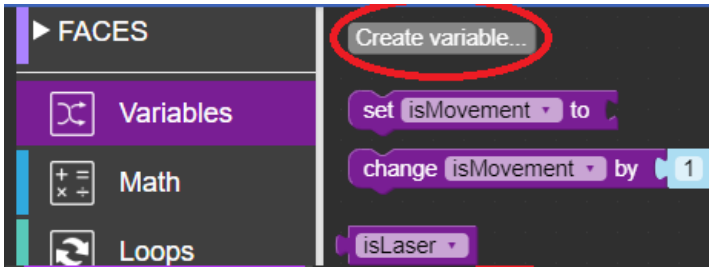
Da wir ständig überprüfen müssen, ob sich der Wert ändert, wird eine Schleife notwendig sein, diesmal ohne zu warten.

So könnte die Schleife aussehen, die immer beim Erkennen einer Bewegung die Nachricht an den Broker sendet.

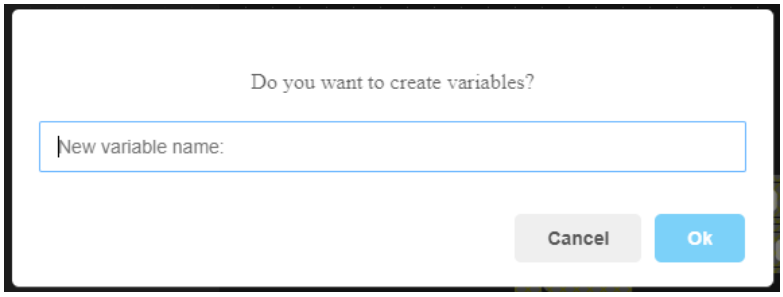


Jedoch, wenn wir dies beim WebClient testen, werden wir erkennen, dass dies ungünstige Resultate liefert. Da die Schleife **sehr oft** pro Sekunde durchlaufen wird und bei einer Bewegungserkennung der Status vom PIR für paar Sekunden auf „1“ bleibt, senden wir hunderte, wenn nicht tausende Nachrichten an den Broker. Dies gilt es zu vermeiden, da wir so unser Kommunikationssystem mit dem Broker stören und überlasten können.

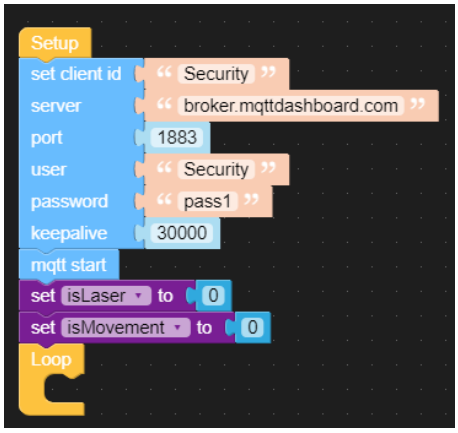
Das Ziel ist es somit, wenn sich der Status ändert, nur eine einzige Nachricht zu senden. Die können wir mit eigen erstellten Variablen erreichen die den Zustand des Units speichern.



In UIFlow können wir ganz einfach solche Variablen erstellen, indem wir eine unter „Variables“ > „Create variable…“ erstellen,

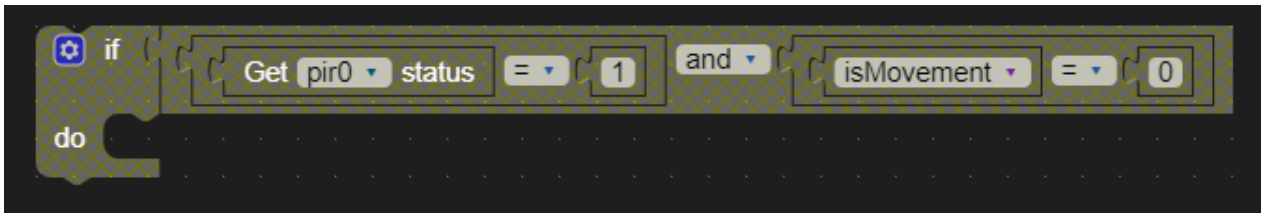


und ihr einen passenden Namen geben. Ich hab für die PIR Variable den Namen „isMovement“ und für die Laser Variable „isLaser“ gegeben und sollen beide mit dem Wert „0“ starten

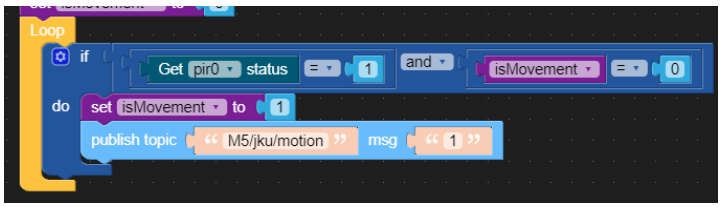


Da der optimale Zustand dieser Variable bei „0“ liegt, wird dies am beim „Setup“-Bereich festgelegt, vor der Schleife.

Der nächste Schritt, ist die Abfrage (also das „if“) so umzuändern, so dass es die Variable nun berücksichtigt. Es soll also nur die Nachricht senden, wenn der neue Status „1“ ist, und der alte „0“ ist.



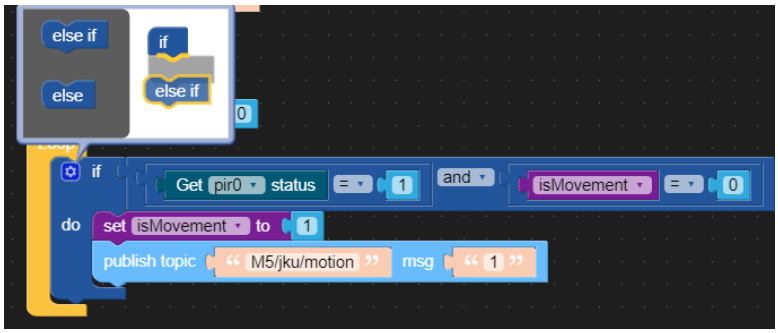
Wenn man die „if“-Abfrage von der Abbildung weiter oben mit einem „and“-Operator erweitert und prüft, ob die Variable auch „0“ ist, sollte dies für diesen Fall passen. Da momentan die Variable nicht verändert wird, und somit ständig auf „0“ bleibt, werden wir das selbe Resultat haben, und es werden wieder hunderte Nachrichten geschickt. Deswegen muss die Variable, im Falle, dass eine Bewegung erkannt wird, auf „1“ gestellt werden, damit dieser „do“-Block nur einmal ausgeführt wird.



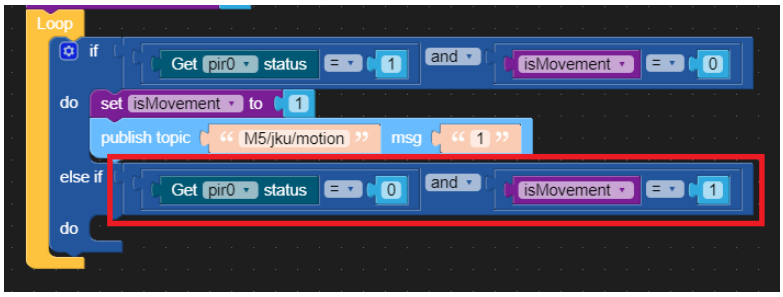
Mit dem Setzen der Variable auf 1 wird dies nun sichergestellt. Dazu soll auch gleich die Nachricht gesendet werden.

Das Problem jetzt: Beim Ausführen wird die Nachricht korrekterweise nur einmal gesendet, jedoch bleibt die Variable auf 1, sodass es nicht mehr dazu kommen kann, erneut eine zu senden. Dazu muss der Fall noch realisiert werden, wenn keine Bewegung erkannt war. Dies erfolgt sehr ähnlich nur mit umgekehrten Ziffern.

Hier gilt es somit



1. Den „if“-Block um einen „else-if“ Bereich zu erweitern für die Abfrage.



2. Die Bedingung so zu formulieren, dass wenn keine Bewegung erkannt wird aber vorher schon eine erkannt worden ist (isMovement = 1), dieser Block ausgeführt wird.

3. Nachdem die Bedingung korrekt ist, soll die Ausführung die Variable auf „0“ setzen (weil keine Bewegung mehr erkannt wird) und diese Nachricht auch an den Broker senden.

Beim fertigen Code erkennt man, dass das zweite „if“ gleich ist wie die erste, nur die „0“ und „1“ Zahlen vertauscht sind. Somit wird immer bei einer Änderung des Zustands nur eine Nachricht gesendet. Eine für das Erkennen der Bewegung und eine, dass keine Bewegung mehr stattfindet.

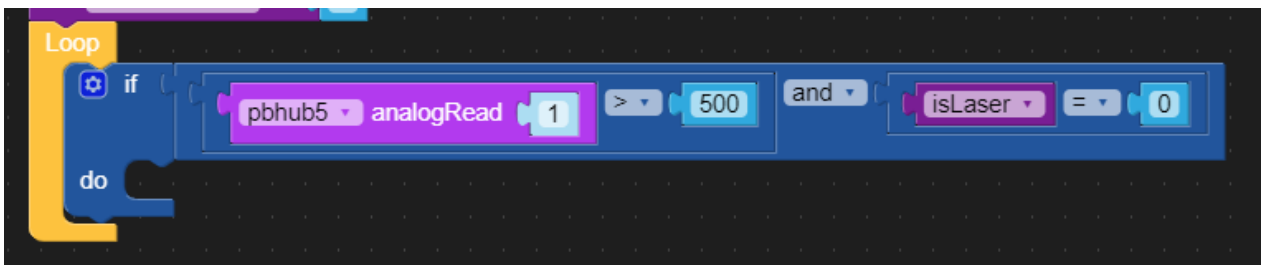
Laser RX / TX

Nun zu der Implementierung der Laserschranke. Eines vorweg: Sie funktioniert ziemlich gleich wie die für den Bewegungsmelder. Sobald die Laserschranke unterbrochen worden ist, soll eine Nachricht gesendet werden und der Status davon gespeichert werden. Wenn die Verbindung zwischen den Laserstrahl Sender und Empfänger wieder intakt ist, soll wieder eine Änderungsnachricht gesendet werden, sowie der Status zurückgesetzt werden.

Da wir diesmal kein PIR-Unit spezifischen Block zur Überprüfung des Status haben, wie erkennen wir nun dies bei der Laserschranke? Wir haben in unserem Fall die zwei Laser Units an dem Pb.HUB angeschlossen. Wie in der vorherigen Kategorie können wir Werte über den **„analogRead“ Block** lesen. Dabei ist nur der Wert des Empfängers (LaserRX) wichtig, da der Sender (LaserTX) durchgehend einen Laserstrahl von sich gibt.

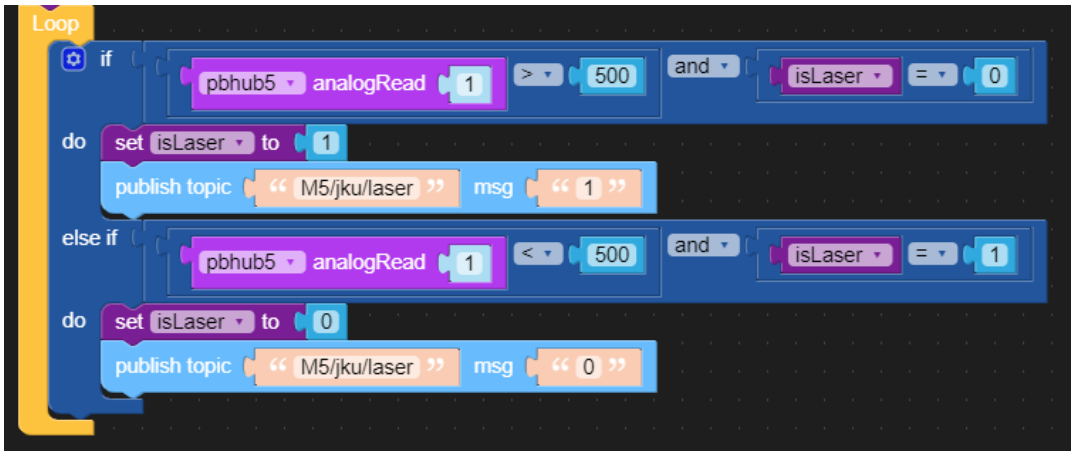
Wenn man mit dem Laser-Units herumtestet, (z.B. mit einer Labelausgabe von dem „analogRead“) erkennt man, dass der analogRead Wert durchgehend ein hoher ist. Zielt der Laser genau auf den Empfänger, so wird der Wert sehr gering. Der „Normalzustand“ ist, wenn der Laser durchgehend genau auf den Empfänger zielt. (*wichtig: die Laser-Units gut befestigen!*)

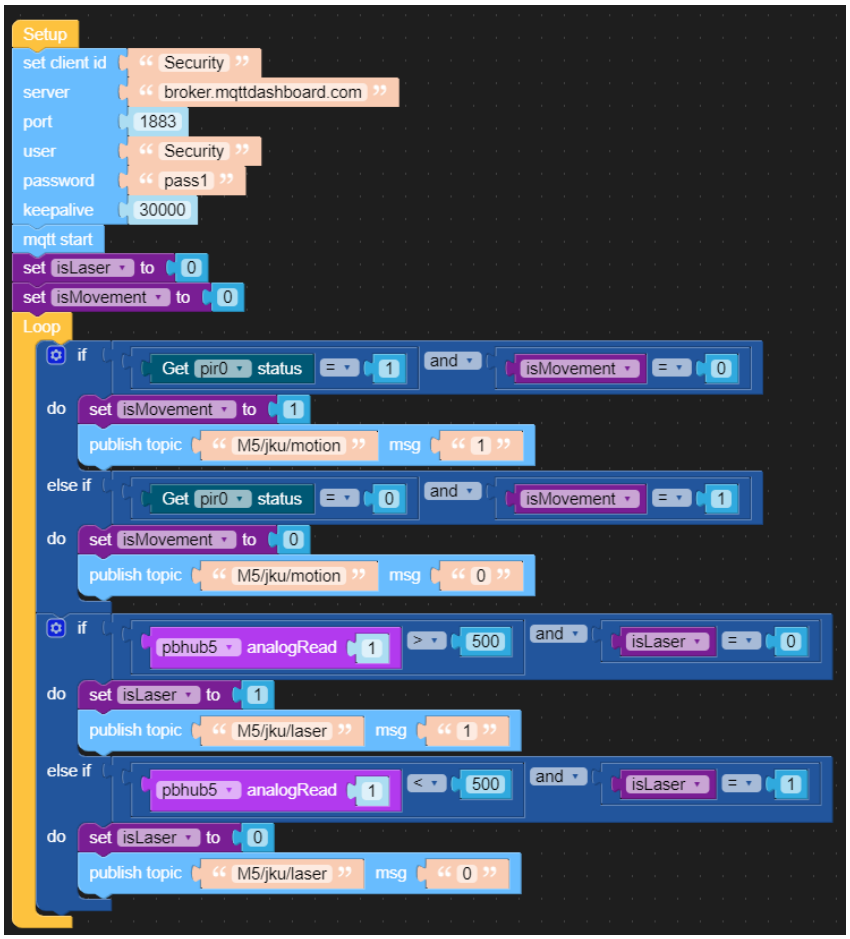
Für meinen Fall habe ich den Wert „500“ als Vergleichswert genommen, da „analogRead“ entweder weit unter 500 oder weit über 500 ist.



Analog wie beim Bewegungsmelder benötigen wir somit wieder eine „if“-Abfrage wo einerseits der Wert von analogRead abgefragt werden muss, andernfalls auch Variable des Lasers.

Wird die Schranke unterbrochen, so soll die Variable geändert und eine Nachricht an den Broker gesendet werden. Wie im vorherigen Beispiel müssen wir für den anderen Fall ein „else-if“ hinzufügen und nur die Ziffern ändern.





Die fertige Kategorie sieht somit wie folgt aus

Nochmals möchte ich erwähnen, dass es für das Thema Sicherheit sehr viele Units für den M5Stack gibt, und man diese Kategorie somit beliebig erweitern könnte

3.2.2.4. Control

Diese Kategorie ist für das Beobachten und Steuern der anderen Geräte zuständig. Alternativ könnte man statt dieser auch eine eigene Handy Applikation erstellen mit derselben Funktionalität (dazu im nächsten Kapitel mehr), aber da der M5Stack ein Bildschirm mit mehreren Tasten und auch über Units mit Tasten verfügt, können wir dies am M5Stack selbst realisieren.

Zuerst gilt es einzuteilen, welche Bereiche man beobachten will und welche man steuern will. Außerdem sieht man hier eine gute Zusammenfassung aller Topics in unserem System

Steuern:

Wir können einerseits die Wunschtemperatur steuern, anderseits die Beleuchtung im Haus. Die Topics dafür wären

- **M5/jku/temp_set** … Wunschtemperatur
- **M5/jku/brightness** … Beleuchtung

Beobachten:

Alle anderen Topics bzw. Nachrichten, die von den M5Stacks gesendet werden

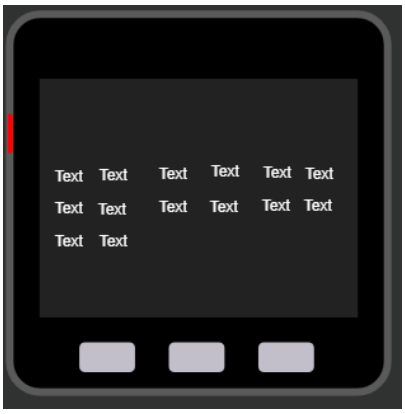
- **M5/jku/temp_actual** … Temperaturwerte des Sensors
- **M5/jku/humidity** … Ob die Erde ausreichend bewässert ist
- **M5/jku/light** … Ob die Pflanze ausreichend Licht bekommt
- **M5/jku/laser** … Wann die Laserschranke unterbrochen wird
- **M5/jku/motion** … Wann eine Bewegung im Haus erkannt wird

Somit benötigen wir 5 Subscribe-Blöcke und 2 publish-Blöcke.

Anzeige:

Fangen wir mit der Bildschirmanzeige an. Auch wenn wir, neben den 5 zu beobachtenden Topics, nur zwei verändern wollen, ist es natürlich von Vorteil zu wissen, wie der Wert dieser 2 momentan ist. Deswegen sollten wir am Bildschirm alle 7 Werte darstellen können.

Dafür benötigen wir einige Text-Labels. Das Design ist einem selbst überlassen. Ich habe für meine Anzeige 14 Labels erstellt.



Warum 14? Weil 7 Lables Variablen der Topics sind und sich ändern. Die anderen 7 geben als Text die Bezeichnung dieser 7 an.

Nun mit Klick auf diese kann ich deren ID (Idententifikations-Namen) verändern, sowie auch den Text den sie anzeigen

Mit ein paar Text Änderungen in UIFlow kann man für eine gute Übersicht der Werte sorgen. Das „#“ der einzelnen Werte sind die Variablen, die durch die Geräte über den MQTT-Broker erst entstehen werden bzw. verändert werden.

Hinweis: Man kann den Namen der Lables belassen (z.B. lable0, lable1 usw.), jedoch ist zu empfehlen, diese in diesem Fall umzubenennen, da man so diese viel leichter findet, wenn man Werte ihnen im Code zuweisen möchte.

Subscribe-Variablen

Die Logik für das Beobachten der Werte ist im Grunde sehr simpel. Ein Subscribe-Block auf das entsprechende Topic und beim Empfangen von Nachricht dies über das Label am Bildschirm anzeigen lassen.

Sofern der MQTT-Startblock nicht vergessen worden ist, genügen solche Subscribe-Blöcke vollkommen aus, um die Werte anzeigen zu lassen

Wenn man die Labels korrekt benennt, führt das gleich zu einer besseren Übersicht.

Hinweis: Bei den Variablen der Pflanze und der Security werden Werte wie „0“ und „1“ angezeigt. Man könnte, wenn man will, dies schöner anzeigen lassen, indem man die Label nicht die Topic-Werte direkt anzeigen lässt, sondern über eine „if“ Abfrage das Label einen bestimmten Text anzeigen lässt.

Beispiel:

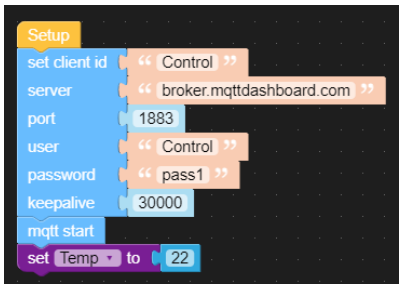
So wird die Frage, ob die Pflanze ausreichend bewässert ist, mit „ja“ oder „nein“ beantwortet, anstatt „0“ oder „1“.

Publish-Variablen

Bisher brauchte man für die Anzeige keine Units für den M5Stack, da dieser nur Werte anzeigen musste von anderen Geräten. Nun geht es darum, selbst Werte an den Broker zu senden. Dabei geht es um die **Lichtstärke** und um die **Wunschtemperatur**.

Wunschtemperatur

Für diese Variable eignet sich die „Dual-BUTTON-Unit“ sehr gut, da sie zwei Tasten hat, eine rote und eine blaue. Mit einem Tastendruck auf den roten Knopf soll die Wunschtemperatur um eins erhöht werden, und bei dem blauen Knopf um eins verringert werden. Was ist jedoch die „Wunschtemperatur“? Mit welchem Wert beginnt sie?

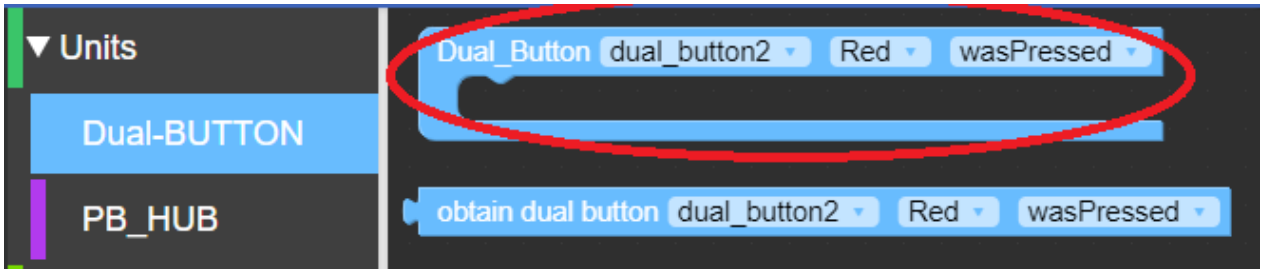


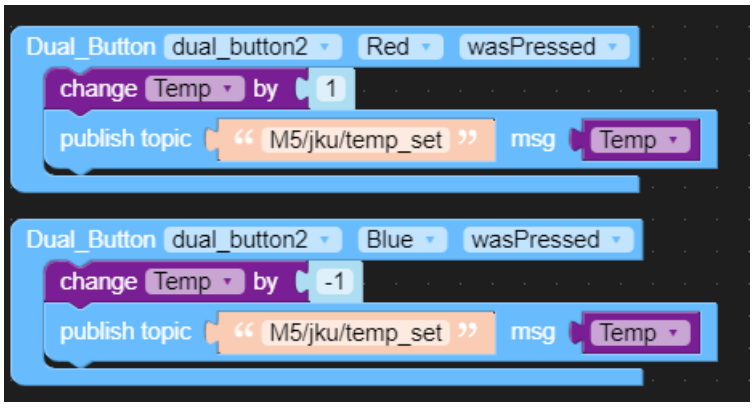
Dafür muss zuerst eine UIFlow Variable erstellt werden und beim Start-Block vor einer Schleife auf einen Wert gesetzt werden. Ich habe sie „Temp“ benannt und den Wert „22“ gewählt, weil dies für mich eine angenehme Temperatur ist.

Außerdem soll auch gleich beim Start des M5Stacks eine MQTT-Nachricht mit dem Startwert dieser Variable gesendet werden

Hinweis: Dieses Senden passiert nur einmal genau beim Einschalten den M5Stacks

Nun zur Steuerung dieser Variable. Nachdem die richtige Unit in UIFlow hinzugefügt worden ist, muss man den Block in der Abbildung darunter hinzugefügt werden. Einmal für den roten Knopf, einmal für den blauen.





Da beide Werte Zahlen sind, können wir den Variablen-Block „change X by“ gut verwenden.

Diese zwei Blöcke sorgen dafür, dass beim Tastendruck, die Wunschtemperatur verändert wird, und als Nachricht an den MQTT-Broker gesendet wird

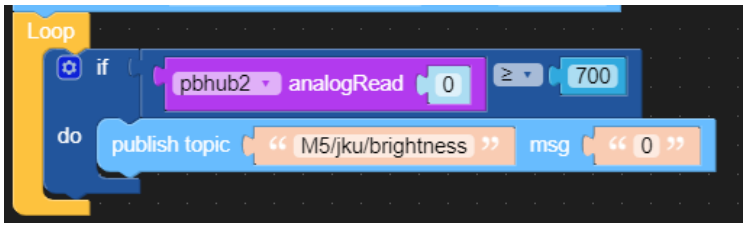
Lichtstärke

Für die Lichtstärke nutzen wir die Unit „**ANGLE**“. Ein wiederkehrendes Problem ist, dass die beide Units den Port-B Anschluss benötigen, und wir somit wieder auf den Pb.HUB angewiesen sind. Da wir damit erfahren genug sind, sollte dies keine Herausforderung mehr darstellen. Einfach die ANGLE-Unit an den Pb.HUB anstecken und über eine Label-Ausgabe testen, welche Werte beim Drehen des Drehschalters angezeigt werden.

In meinem Fall sind es Werte von 0 bis ca. 1000. Ist der Drehschalter bis zum Anschlag im Uhrzeigersinn, so ist der Wert sehr klein. Für gewöhnlich dreht man Drehschalter im Uhrzeigersinn, um die Lichtstärke zu erhöhen. Deswegen soll, je kleiner der ANGLE-Wert (bzw. der „analogRead-Wert“ vom Pb.HUB), die Lichtstärke erhöht werden.

Da ständig die Werte von der ANGLE-Unit gelesen werden muss, wird hier eine Schleife benötigt. Um die Werte zu überprüfen, brauchen wir einen „if“-Block, am besten einen erweiterbaren, da wir in der Home-Kategorie **vier** Werte für die Lichtstärke definiert haben (0,1,2,3)

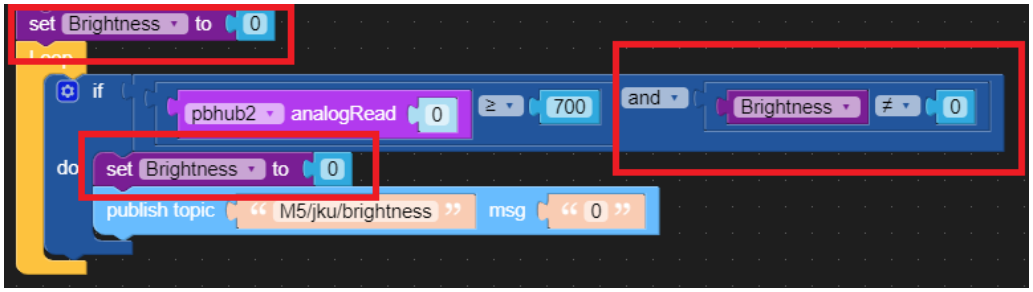
Fangen wir mit der Situation an, dass das Licht ausgeschaltet werden soll. Der dafür definierte Wert war „0“ für das Topic „M5/jku/brightness“. Der ANGLE-Wert dafür ist wie gesagt ein sehr hoher. Da die Spannweite von 0 bis ca- 1000 geht, könnten wir diese auf 4 gleich große Wertebereiche aufteilen und für alle 4 eine Abfrage machen, ob sich der ANGLE-Wert da befindet. Ich habe beim Testen herausgefunden, dass Grenzen bei **700, 500,200** recht sinnvoll sind.



Die erste Abfrage könnte so in etwas aussehen

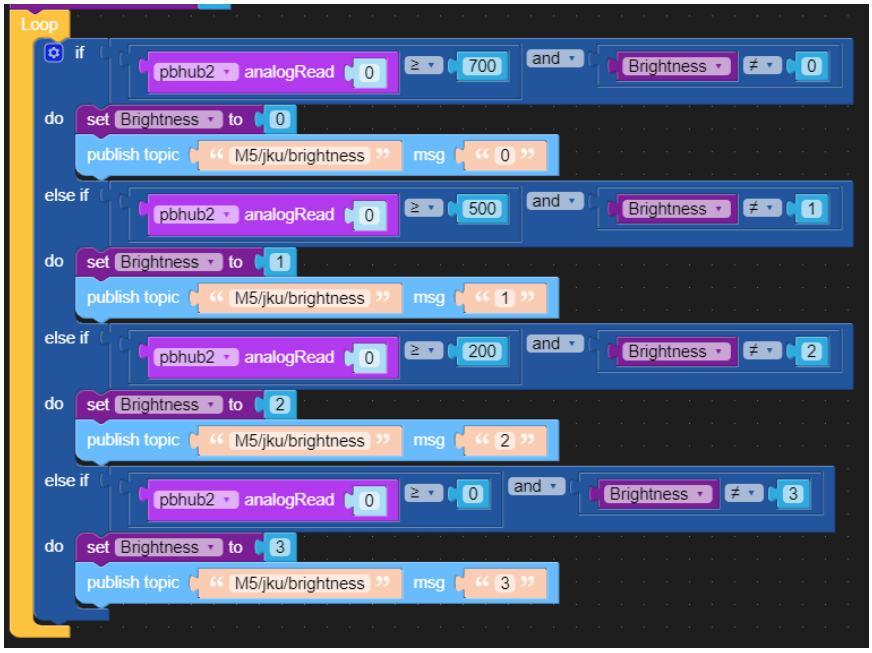
Beim Testen werden wir erkennen, dass ein bekanntes Problem auftauchen wird. Sobald der ANGLE-Drehschalter so gedreht ist, dass die Werte über 700 sind (gegen den Uhrzeiger), dann werden wieder hunderte von Nachrichten an den Broker gesendet. Dies gilt es wieder zu vermeiden. Deswegen brauchen wir wieder eine Variable für die Lichtintensität und wollen nur dann eine Nachricht senden, wenn sich die Stärke **ändert**. Beginnen soll sie mit dem Wert „0“.

Der „do“-Bereich von diesem „if“ soll nur dann ausgeführt werden, wenn die ANGLE-Werte über 700 sind **UND** die Lichtstärke davor **nicht** „0“ war. Wenn dieser Block ausgeführt wird, dann soll auch diese Variable auf „0“ gesetzt werden, damit nur eine Nachricht gesendet wird.



Diese Abfrage müssen wir nun für die anderen Wertebereiche der ANGLE-Unit wiederholen und entsprechend die Variable setzen. Dabei erweitern wir den „if“-Block um 3 weitere „else-if“-Bereiche.

Erweitert man den „if“-Block, so würde man zu so einem ähnlichen Ergebnis kommen.



Jedoch werden wir ein Problem feststellen, wenn man dies dann testen würde.

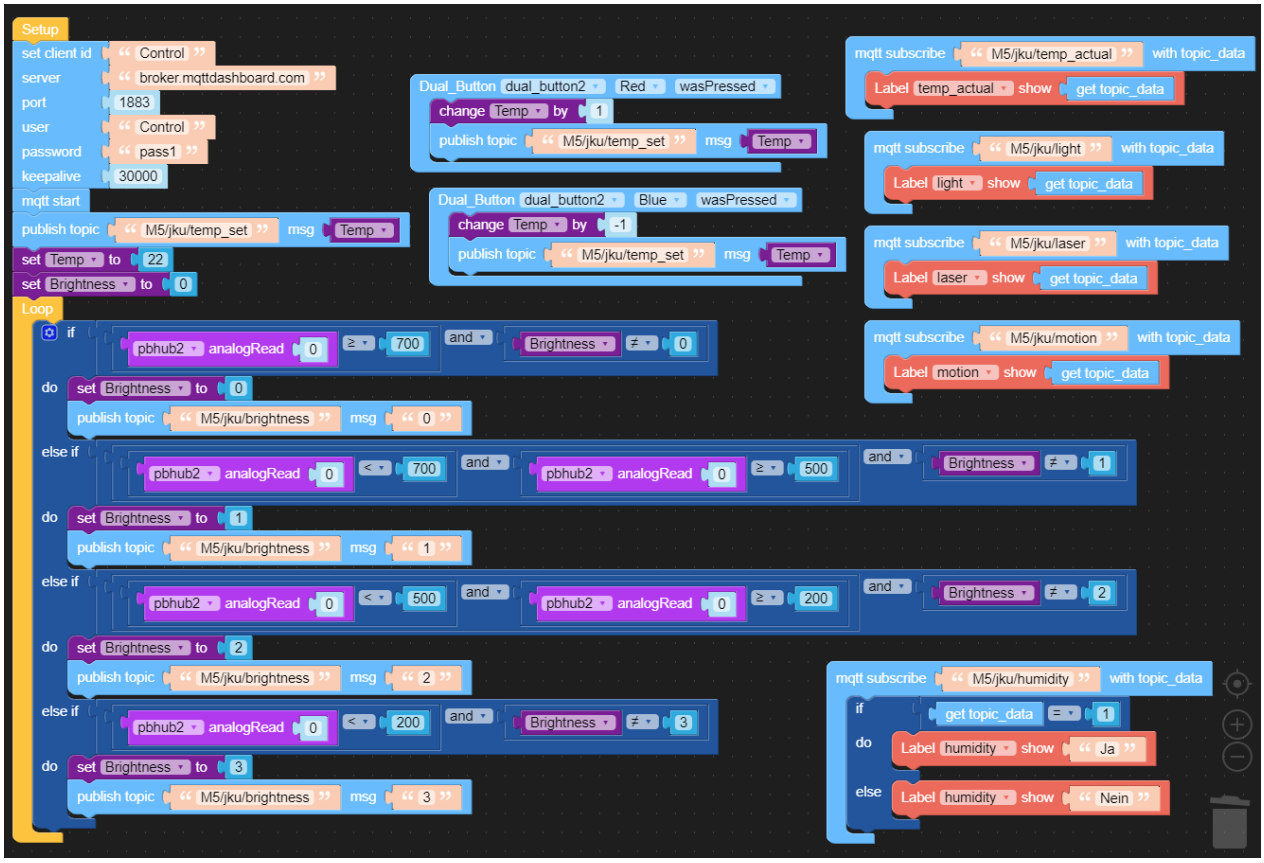
Der Vergleich, ob der „Pb.HUB“-Wert über einen Wert ist, zum Beispiel bei 1000, ist bei allen 4 Abfragen „richtig“.

Somit würde das Licht in diesem Fall durchgehend „flackern“, da alle 4 „if“ durchgehend ausgeführt werden.

Aus diesem Grund müssen wir die Werteabfrage noch optimieren, sodass Werte zwischen zwei Grenzen geprüft werden. Zum Beispiel zwischen 700 und 500.

Hier ist es wichtig mit den Vergleichsoperatoren („kleiner“ bzw. „kleiner oder gleich“) aufzupassen, sodass kein Wert in „if“-Abfragen öfters vorkommen kann.

Es schien fast unmöglich, aber der gesamte Code lässt sich gerade noch so auf einem Bildschirm anzeigen lassen.

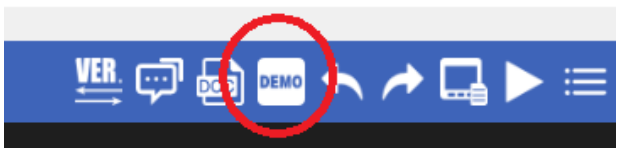


Zusammenfassung

Damit wären nun alle 4 Smart-Home-Kategorien fertig realisiert. Wenn alles korrekt angesteckt worden ist, der Code richtig implementiert ist und die M5Stacks über eine Internet Verbindung verfügen und mit dem gleichen MQTT-Broker kommunizieren, sollte das ganze Zusammenspiel dieser Geräte mit deren Units funktionieren.

Sollte etwas nicht genau funktionieren wie in dieser Dokumentation beschrieben, so macht es immer Sinn zu prüfen warum. „Debugging“ ist der englische Begriff in der Softwareentwicklung, um etwaige Fehler zu identifizieren. Für UIFlow empfiehlt es sich oft, ein Text Feld am M5Stack anzeigen zu lassen und Werte von Units bzw. von MQTT-Nachrichten anzeigen zu lassen.

Der WebClient dient auch als gute Ressource um MQTT-spezifische Funktionen der M5Stacks zu testen. Die Möglichkeit, von dort aus, Nachrichten auch zu senden hilft auch sehr viel.



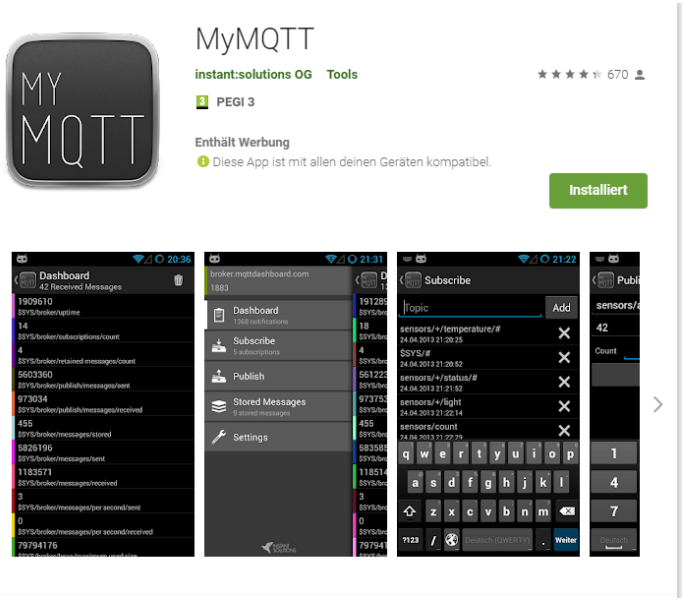
Ansonsten findet man in UIFlow einige Beispielcodes. Diese sind in der oberen rechten Ecke zu finden.

3.3. Steuerung über Mobilgerät

Was gibt es Schöneres als die eigenen Geräte zuhause über sein Mobiltelefon zu steuern? Mit MQTT ist das zum Glück ein Kinderspiel. Den MQHive WebClient kennen wir schon bereits. Dieser funktioniert am Smartphone genauso. Natürlich ist die Bedienung auf einem Mobilgerät nicht optimal. Außerdem wenn man dies öfters nutzt, merkt man, dass man die Topics, beim Neustart des Browser, immer wieder neu subscriben muss, was mit der Zeit recht anstrengend wird und nicht besonders „user-friendly“.

Zum Glück gibt es auch eine fertige App die da ein bisschen Abhilfe schafft.

MyMQTT



Der große Vorteil ist, dass die Adresse des MQTT-Brokers, sowie die abonnierten Kanäle gespeichert werden. Mit dieser kann man recht simpel somit Nachrichten erhalten vom Broker und auch Nachrichten senden (z.B. die Lichtstärke einstellen).

Für iOS gibt es auch einige Apps, wo das Mobiltelefon als MQTT-Client fungieren kann. Unter folgendem Link kann man sich im App Store einige davon ansehen.

3.3.1. Eigene Handy-App entwickeln

Man kann natürlich auch eine eigene Applikation für das Smartphone entwickeln. Der große Vorteil ist dabei, dass die Verbindungsinformationen und das Subscriben von Topics einem erspart bleibt, da dies im Hintergrund bereits passiert. Die Bedienung ist genau dem Anwendungsfall angepasst (Anzeige für Temperatur, Pflanzenstatus, Sicherheitsaspekte) und die Steuerung erfolgt viel angenehmer und übersichtlicher.

Das große Problem dabei: Eine eigene App zu entwickeln ist kein leichtes Unterfangen! Für Softwareentwickler ist dies sicher eine interessante Herausforderung, aber für Laien möglicherweise eine Qual.

Aus diesem Grund werden zumindest Ressourcen und Tipps für die Interessierten bereitgestellt.

Wo fange ich überhaupt mit so einer App-Entwicklung an? Wie gehe ich da vor?

Die Antwort auf diese Frage hängt davon ab, für welches Betriebssystem die Applikation ausgelegt werden soll. Will man die Applikation für Android Smartphones entwickeln, so bietet sich Android Studio als IDE (Entwicklungsumgebung) gut an. Für diese existiert auch eine MQTT-API (Schnittstelle für Funktionen spezifisch für MQTT). Hierbei wird die Programmiersprache **Java** verwendet. Aber auch in **Python** gibt es eine Möglichkeit eine Applikation mit MQTT Funktionalitäten zu programmieren. Möchte man eher die Applikation für iOS entwickeln, so empfiehlt es sich, die Applikation in Swift zu programmieren.

Paar nützliche Links

Android

- Link für den Download der Entwicklungsumgebung **Android Studio**
  https://developer.android.com/studio#downloads
- Link für eine **allgemeine Einführung** in die App-Entwicklung für Android Studio
  https://developer.android.com/training/basics/firstapp
- „**Paho Android Service**“ ist die bereits erwähnte API, um MQTT-spezifische Funktionen ausführen zu können. Dieser Link dient als Startpunkt für die Installation und Implementierung
  https://www.eclipse.org/paho/clients/android/
- Ebenfalls ein Link für Paho Android Service von HiveMQ. Eine etwas detailliertere Einführung mit guten Erklärungen. Sehr zu empfehlen!
  https://www.hivemq.com/blog/mqtt-client-library-enyclopedia-paho-android-service/
- Links für die Paho-API für **Python**
  https://pypi.org/project/paho-mqtt/
  https://www.eclipse.org/paho/clients/python/

iOS

- IDEs für iOS
  https://developer.apple.com/xcode/ide/
  https://atom.io/
- Einführung App-Entwicklung iOS
  https://developer.apple.com/library/archive/referencelibrary/GettingStarted/DevelopiOSAppsSwift/
- MQTT Client für iOS
  https://github.com/flightonary/Moscapsule
  https://github.com/emqx/CocoaMQTT
- Beispiel für eine iOS Implementierung von MQTT
  https://www.cloudamqp.com/docs/swift_ios_mqtt.html

Neben Textuellen Ressourcen empfehlt es sich auch Videos von solchen Einführungen anzusehen. Auf „Youtube“ und ähnlichen Plattformen gibt es einige gute Anleitungen, wie man eine simple Handyapplikation mit MQTT entwickeln kann. Einige meiner Favoriten sind

- https://www.youtube.com/watch?v=BAkGm02WBc0
- https://www.youtube.com/watch?v=I-XyqsxaGQc

Hoffentlich reichen diese Informationen aus, um mit der App-Entwicklung starten zu können, sollte man interessiert sein, eine solche für sein eigenes Smart-Home-System zu entwickeln.

4. Abbildungsverzeichnis

TODO