Projektarbeit Cocktailautomat

Cocktailautomat mit Steuerung über ESP32 – S3 und Raspberry Pi

Erstellt von Niklas Granel

Inhalt

1. Allgemeine Informationen

- **Projektname:** Cocktailautomat
- **Datum:** 18.03.2025 bis 06.05.2025

2. Projektbeschreibung

Kurzbeschreibung:
Es soll ein Automat gebaut werden, der selbstständig Getränke mischt. An einen Touchdisplay kann man auswählen, was man haben möchte. Es soll auch ein Reinigungsprogramm geben, um die Schläuche zu reinigen.

3. Anforderungen und Funktionalitäten

✅ Sensorik

Der Füllstand der Flaschen soll mit einem VL53L0X Sensor überwacht werden. Außerdem soll überwacht werden, ob ein Glas unter dem Auslass steht. Dies geschieht mit einer Reflexionslichtschranke.

✅ Aktoren-Steuerung

Die Peristaltik Pumpen zum Pumpen der Getränke werden über Relais Module mit Optokopplern angesteuert. Ein LED-Ring zeigt an auf welcher Position das Glas stehen soll.

✅ Webinterface & Benutzerinteraktion

Die Bedienung soll über ein Touchdisplay direkt am Automaten laufen.

4. Benötigte Komponenten

4.1 Hardware

Komponente	Modell/Typ	Funktion
Mikrocontroller	ESP 32- S3	Steuerung
Sensor 1	VL53L0X	Füllstands Überwachung Flaschen
Sensor 2	Datalogic 950811290	Überwachung Glas vorhanden
Aktor 1	Peristaltik Pumpen	Dosierung der Flüssigkeiten
Aktor 2	LED-Ring	Beleuchtung vom GLS
Stromversorgung	MW HDR-100-12N MW HDR-60-5 RPI PS 27W BK EU	12V Netzteil 5V Netzteil Raspberry Pi Netzteil 5V
Weitere Bauteile	Raspberry Pi 5, Touch Display für Raspberry Pi	Für Node-RED und MQTT-Server

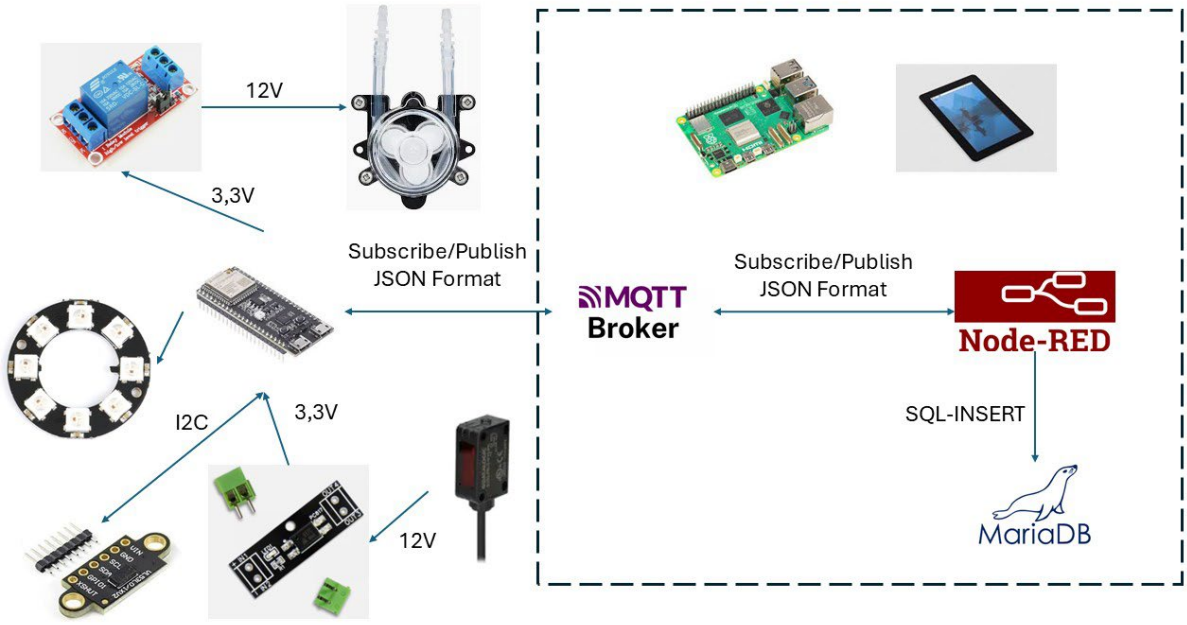
Komponente	Modell/Typ	Funktion
	PC817 Optokoppler Relais Modul 3,3V	12V Schaltsignal auf 3,3V Spannung bringen Ansteuerung der Pumpen

4.2 Software & Datenbank

Komponente	Technologie	Funktion
Microcontroller-Code	Micro Python	Steuerung der Dosierung und des LED-Rings sowie auslesen der Sensoren
Webinterface	Node-RED	Bedienung des Automaten, Auswahl der Dosiermengen
Datenbank	Maria DB	

5. Systemarchitektur

Hier eine Übersicht



6. Zeitplanung (Meilensteine)

Datum	Aufgabe
KW 12 - 13	Materialbeschaffung, ggf. Hardwareaufbau
KW 14	Hardwareaufbau, ggf. ESP 32 und Raspberry Pi einrichten

Datum	Aufgabe
KW 15 - 16	ESP 32 und Raspberry Pi einrichten Programmierung
KW 17	Restarbeiten
KW 18	Finale Testphase
KW 19	Abgabe

7. Offene Fragen & Herausforderungen

Die Herausforderungen sind, dass einige Oberflächen leicht zu reinigen sein müssen. Außerdem wird mir Flüssigkeiten gearbeitet. Davor muss die Elektronik geschützt werden. Die Flüssigkeiten müssen genau dosiert werden. Außerdem muss Node-RED auf dem Raspberry Pi eingerichtet werden und mit dem Touchdisplay bedienbar sein.

8. Fazit & Zielsetzung

Das Projekt war zeitlich knapp bemessen. Dadurch, dass ich 2 Wochen krank war und eine Woche voll arbeiten musste, konnten einige Zusatz Features nicht umgesetzt werden. Diese können als mögliche Erweiterung umgesetzt werden. Es wird nur der Füllstand von zwei Flaschen gemessen. Dieses kann man auf alle acht Flaschen erweitern. Leider habe ich es zeitlich nicht geschafft den LED-Ring mit einzuprogrammieren. Aktuell kann man in der Touch Eingabe nur die Zutatenmengen auswählen. Auch dort kann man vorgefertigte Rezepte verwenden. Außerdem konnte ich den Microphyten Code nicht mehr optimieren.

Eine größere Herausforderung war die Installation vom Raspberry PI. Auf diesem läuft Node-RED, der MQTT-Broker, sowie Maria DB. Dies hat nach der Einarbeitung erfolgreich funktioniert.

Allgemein war die Integration der MQTT-Kommunikation am herausforderndsten.

9. Installationsanleitung

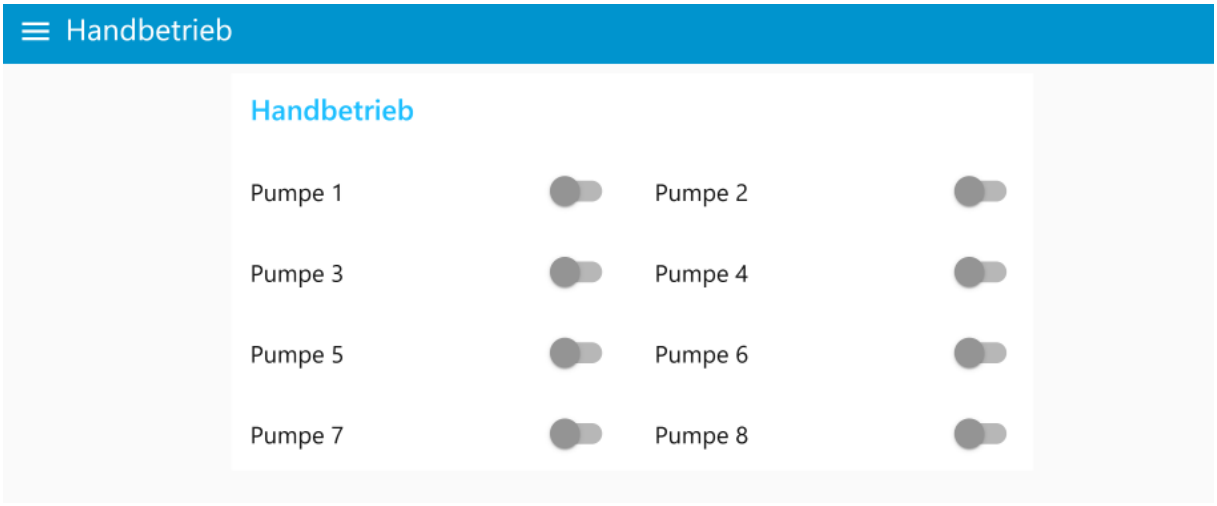
Der Cocktailautomat muss an einem Ort mit WLAN aufgestellt werden. Danach müssen der Raspberry Pi und der ESP 32 mit dem WLAN verbunden werden. Der Raspberry Pi muss ein fest IP-Adresse vom Router erhalten. Diese feste IP-Adresse muss im ESP 32 eingetragen werden, um die Kommunikation mit dem MQTT-Broker sicherzustellen.

10. Bedienungsanleitung

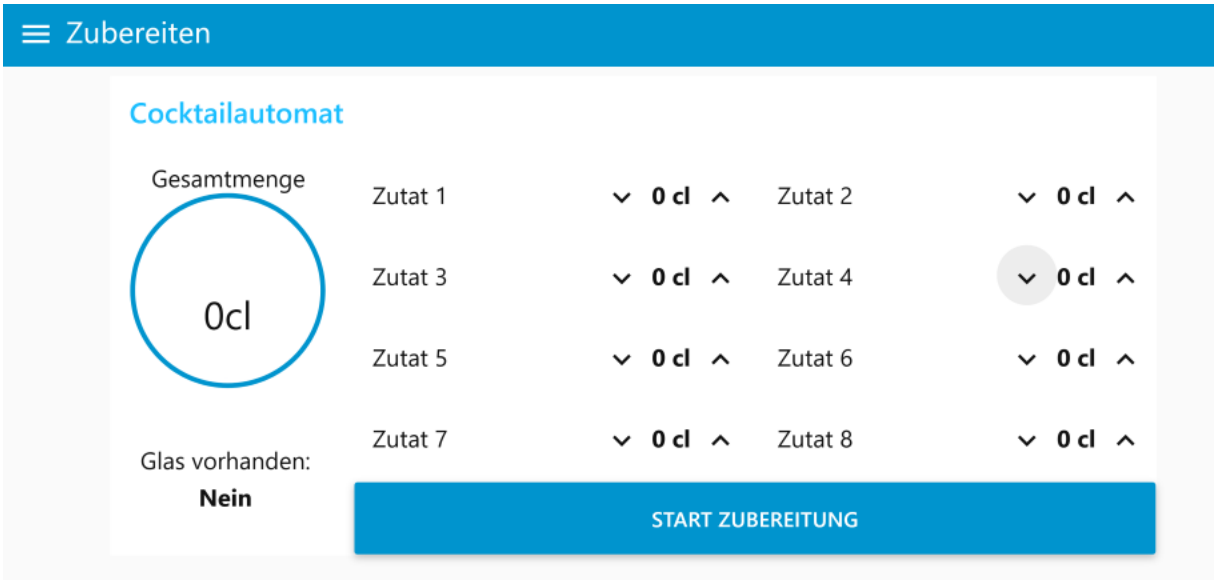
Der Cocktailautomat muss mit dem Strom verbunden werden. Danach muss der FI/LS Schalter auf der Rückseite eingeschaltet werden. Nun fährt der Cocktailautomat hoch. Nachdem alles gestartet ist, muss auf dem Raspberry Pi das Node RED Dashboard gestartet werden.

Nun werden die Flaschen mit den Zutaten in den Automaten gestellt und die Schläuche kommen in die Flaschen und der Füllstands Sensor kommt auf den Flaschenkopf.

Als nächstes müssen die Schläuche des Cocktailautomaten befüllt werden. Dafür wir ein Glas unter den Auslass des Cocktailautomaten gestellt. Danach werden im Handbetrieb alle Zutaten nacheinander angesteuert, bis die Zutat aus dem Auslass kommt.



Ist dieser Schritt abgeschlossen kann auf die Zubereiten Ansicht gewechselt werden. Hier können die Mengen der Zutaten ausgewählt werden. Spätestens jetzt sollte man das Glas unter den Auslass stellen. Mit Betätigung des Startbutton beginnt der Automat nacheinander die Zutaten in das Glas zu füllen. Nach diesem Vorgang kann das Glas entnommen werden. Wenn kein Glas unter dem Auslass steht, lässt sich der Vorgang nicht starten.



Dies kann nun so lange wiederholt werden, bis die Zutaten leer sind. Diese können dann gewechselt werden und dann geht es weiter.

Soll der Automat werden außer Betrieb genommen werden, müssen die Schläuche gereinigt und entleert werden. Dafür werden die Schläuche aus den Flaschen mit den Zutaten

genommen. Diese kommen jetzt in Behälter mit lauwarmem Wasser. Nun werden alle Schläuche, die benutzt wurden, durchgespült. Dafür muss wieder auf den Handbetrieb gewechselt werden. Pro Zutat soll ein halbes Glas Wasser ca. 15cl durchlaufen. Dafür ein Glas unter den Auslass stellen und die Zutaten nacheinander auf der Bedienüberfläche anwählen. Ist das Wasser einer Zutat durchgelaufen, wird der Zutatenschlauch aus dem Wasser gezogen und die Pumpe saugt den Schlauch leer. Kommt keine Flüssigkeit mehr aus dem Auslass vom Gerät kann die Zutat abgeschaltet werden und die Reinigung vom nächsten kann beginnen. Sind alle Schläuche gereinigt, können die Oberflächen unter dem Auslass sowie die Stellplätze der Flaschen mit einem feuchten Lappen gereinigt werden.

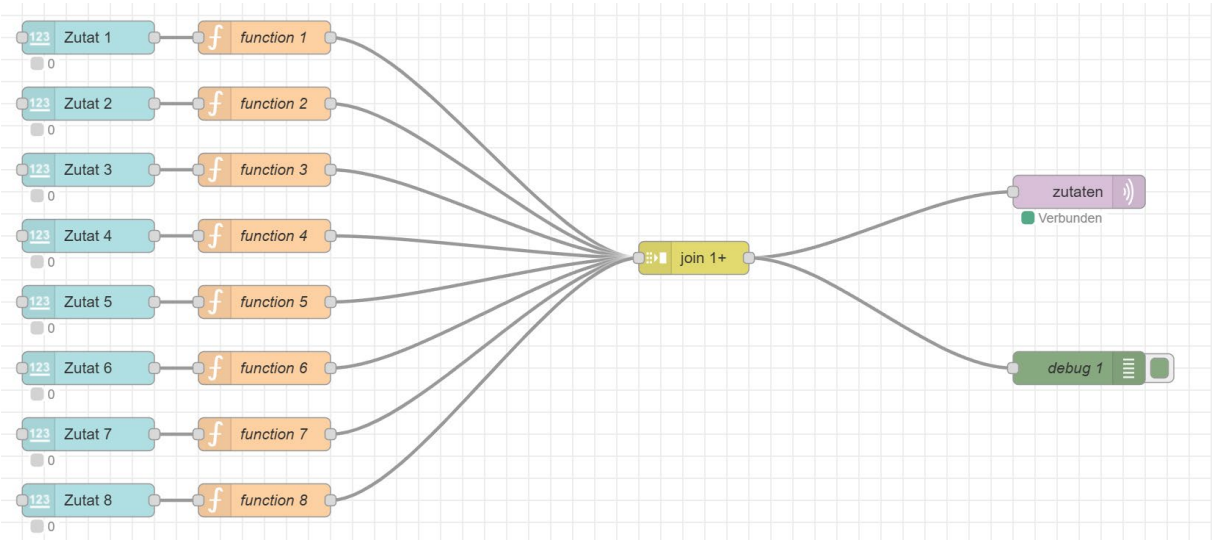
11. Technische Besonderheiten

Als Technische Besonderheit ist bei mir die Steuerung über ein Raspberry PI mit Touchdisplay zu nennen. Node RED, Maria DB und der MQTT-Broker laufen auf ihm. Außerdem benötigt mein Cocktailautomat 3 verschiedene Spannungsebenen 3,3V, 5V und 12 V. Auch die eingesetzt Reflexionslichtschranke kommt aus einen Industriellen Anwendung und der Signalpegel musste angepasst werden.

12. Persönliches Fazit

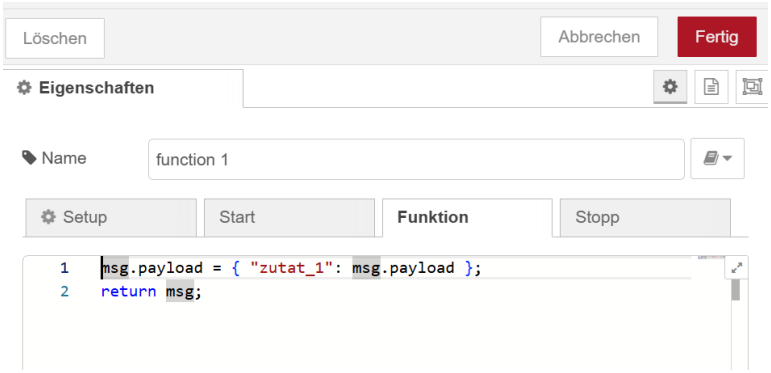
Die Grundfunktion ist gegeben. Ich kann einstellen, von welcher Zutat wie viel ins Glas gefüllt werden soll. Einige Sensor Daten konnte ich leider nicht so einbinden, wie ich wollte. Die Latenzzeit ging hoch und damit war keine vernünftige Bedienung mehr möglich. Um den Code ressourcenschonender zu gestaltet, fehlt mir am Ende die Zeit. Auch den LED-Ring konnte ich aus Zeitgründen bis zur Abgabe nicht mehr einbinden. Ich habe gemerkt, dass das Projekt deutlich komplexer war als ich es mir zu Anfang vorgestellt habe und bei der Datenübertragung zwischen ESP und Raspberry Pi einiges schief gehen kann. Die zwei Krankheitswochen haben mir gefehlt, Ich konnte die verlorene Zeit leider nicht wieder aufholen.

13. Node-RED

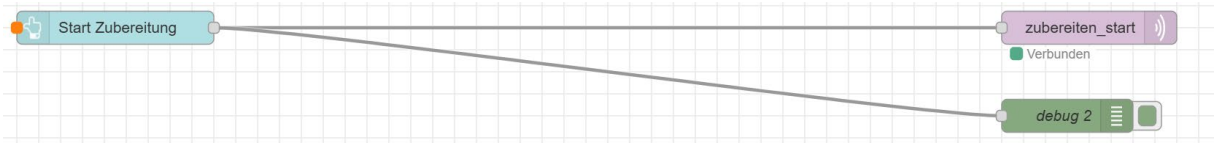


Dieser Teil dient zur Eingabe der Zutatenmengen und der Versendung per MQTT.

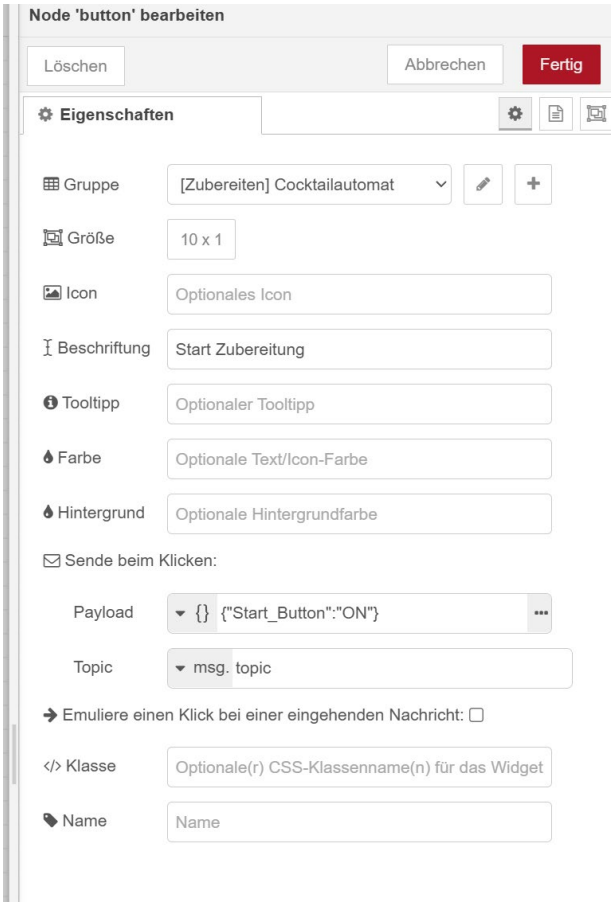
Der Inhalt einer Funktion Node:

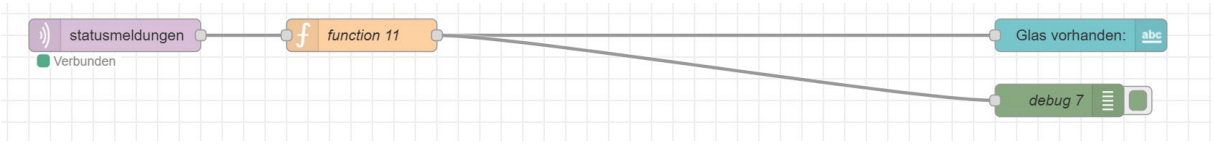


Dies ist als Beispiel function 1. Alle anderen Funktionen sind identisch. Natürlich ändert jede ich auf die entsprechende Zutat also zutat_1, zutat_2, usw.

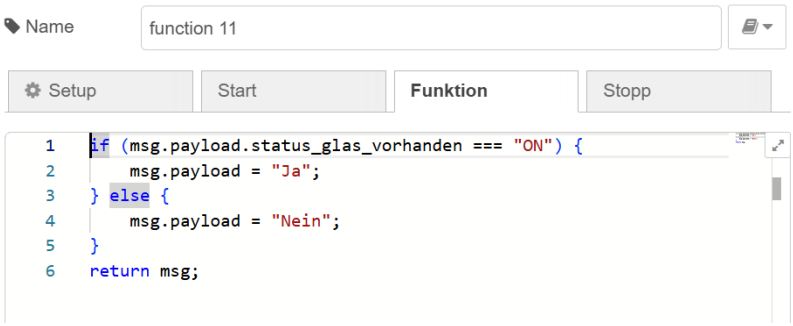


In diesem Teil wir der Startbefehl verarbeitet

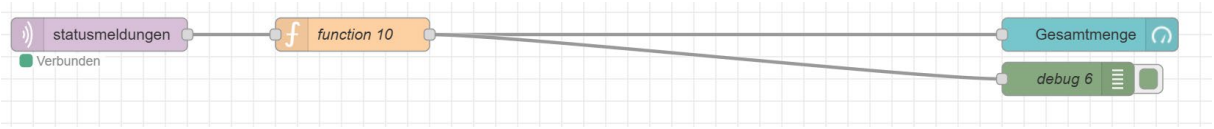


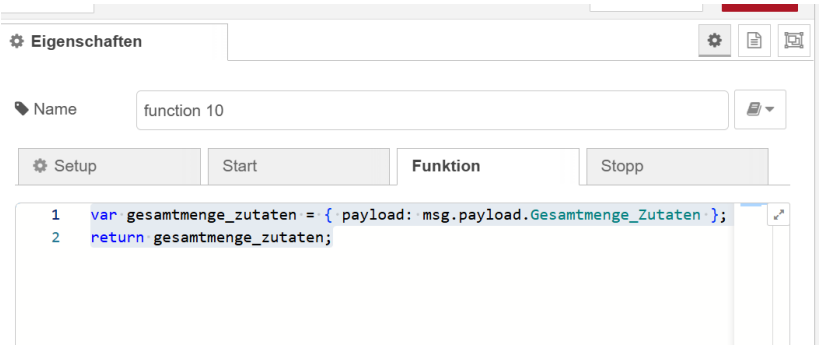


In diesem Teil wird die Nachricht verarbeitet, dass ein Glas vorhanden ist

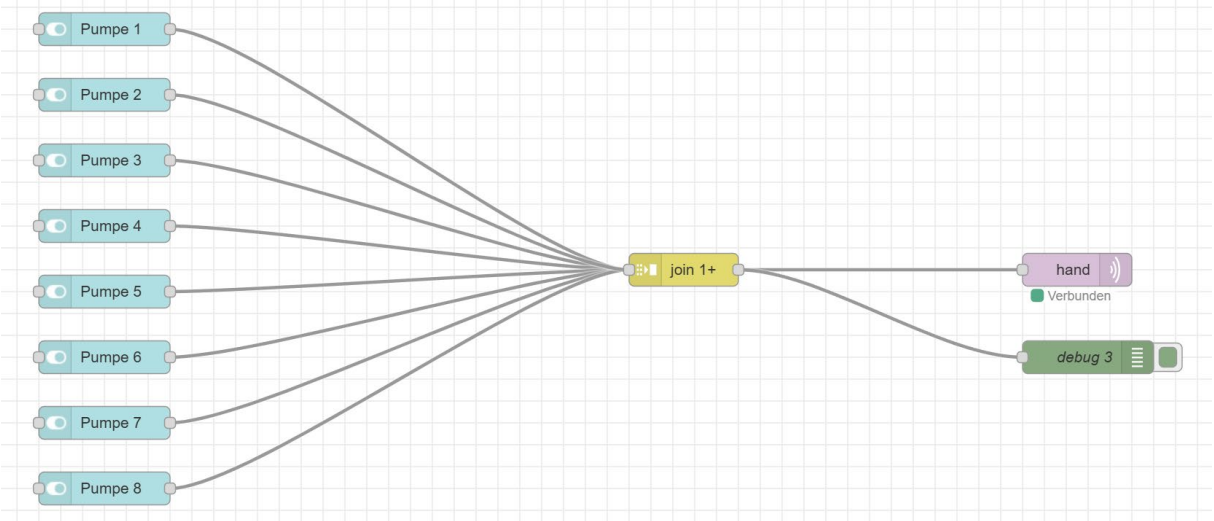


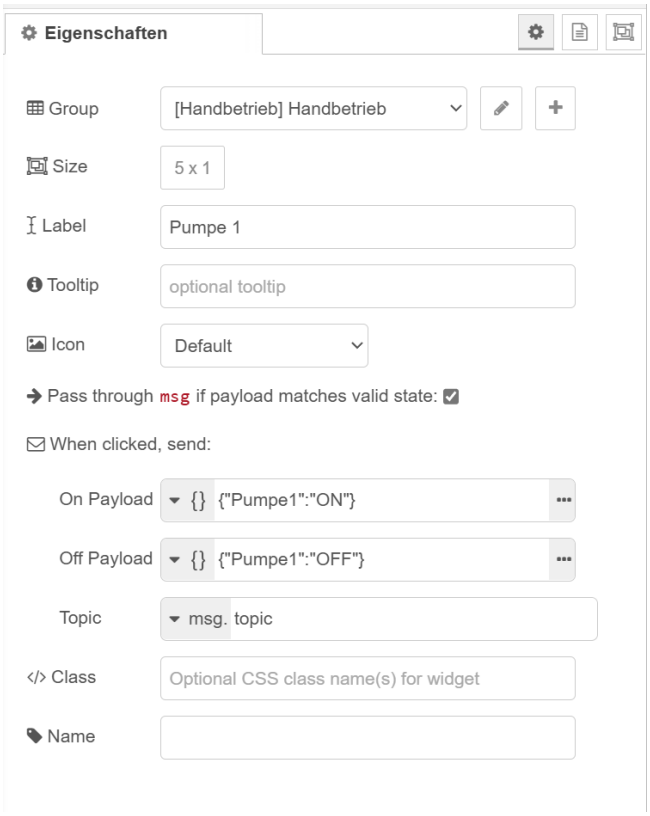
Im nächsten Teil wir die errechneten Gesamtmenge zurückgegeben und auf der Eingabe ausgeben



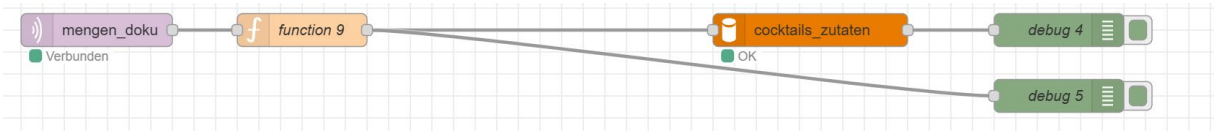


Danach folget die Bedienung im Handbetrieb

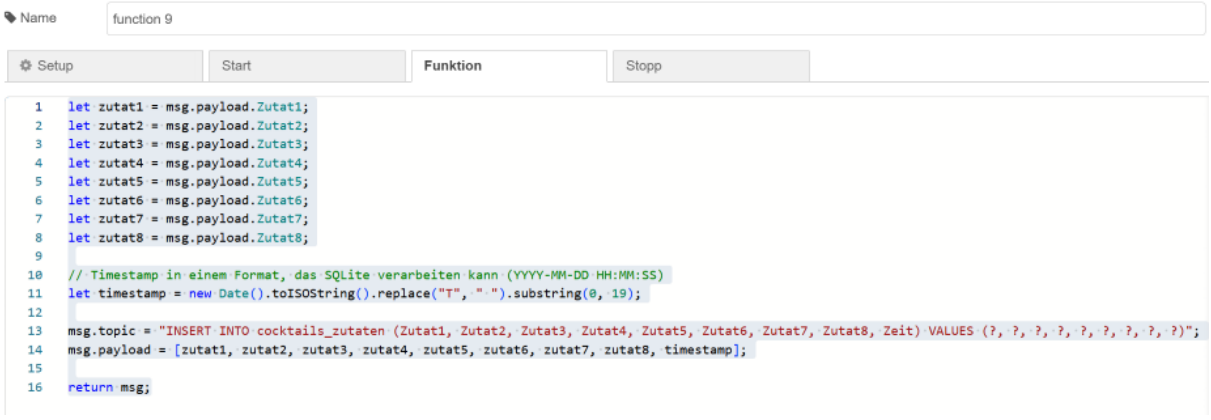


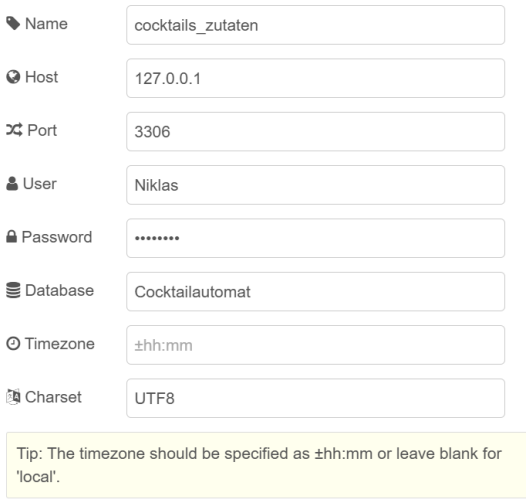


Alle Touch Schalter sind gleich konfiguriert. Nur eben für die entsprechende Pumpe.

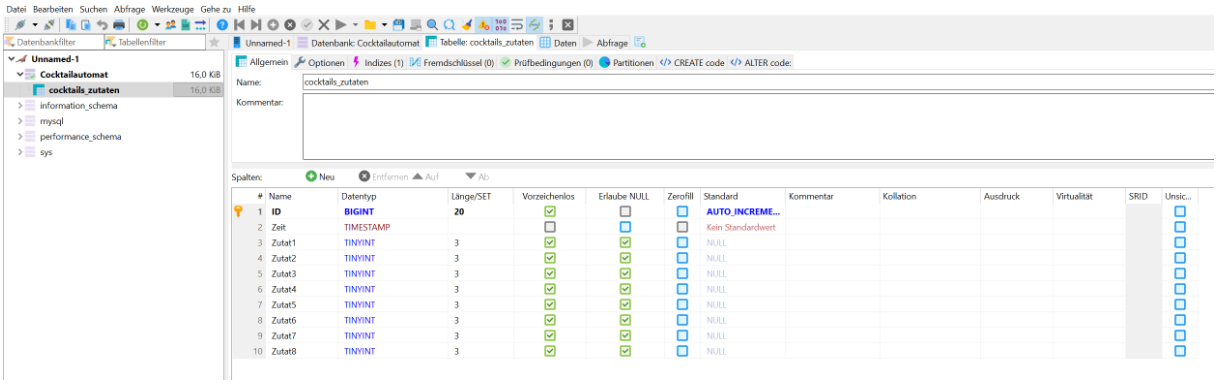


Hier werden die Flüssigkeitsmengen empfangen in die Datenbank geschrieben.





14. Datenbank



Cocktailautomat.cocktails_zutaten: 4 Zeilen gesamt (exakt)

#	ID	Zeit	Zutat1	Zutat2	Zutat3	Zutat4	Zutat5	Zutat6	Zutat7	Zutat8
1	1	2025-05-06 19:03:37	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
2	2	2025-05-06 19:05:02	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
3	3	2025-05-06 19:08:23	1	1	1	1	1	1	1	1
4	4	2025-05-06 19:09:17	0	1	1	3	3	0	0	0

15. Microphyten Code

```
#-------------------------------------------------------------
--------------

# Programm füt den Betrieb eines Cocktailautomaten mit Node
Red bedienung

# Ersteller : Niklas Granel

# Datum: 06 .05.2025

# Version 1.5
```

```
#-------------------- Bibliotheken -------------------

import network

import ujson

import time

from umqtt.simple import MQTTClient

from machine import Pin, I2C

from VL53L0X import VL53L0X


#-------------------- Pins -------------------

# Pumpen sind mit digitalen Ausgängen verbunden (GPIO 35 bis
40, 2, 1)

# HIGH = Pumpe EIN, LOW = Pumpe AUS


pumpe_1 = Pin(35, Pin.OUT)

pumpe_2 = Pin(36, Pin.OUT)

pumpe_3 = Pin(37, Pin.OUT)

pumpe_4 = Pin(38, Pin.OUT)

pumpe_5 = Pin(39, Pin.OUT)

pumpe_6 = Pin(40, Pin.OUT)

pumpe_7 = Pin(2, Pin.OUT)

pumpe_8 = Pin(1, Pin.OUT)


# Eingangspin zum Erkennen, ob ein Glas vorhanden ist (LOW =
kein Glas, HIGH = Glas erkannt)
```

```
glas_vorhanden = Pin(4, Pin.IN) # Pulldownwiederstand extern


# I2S Pins

scl_1 = Pin(6)

sda_1 = Pin(7)

scl_2 = Pin(15)

sda_2 = Pin(16)


#-------------------- Zugänge -------------------

# Zugangsdaten für WLAN und MQTT-Broker IP-Adresse

# Diese werden für die Verbindung zum Netzwerk und Node-RED
benötigt


WIFI_ssid = "FRITZ!Box 7530 RR"

WIFI_password = "67307380203238062131"

ip_adresse_mqtt_server = "192.168.178.36"

mqtt_client_id = "esp32_cocktailautomat"

mqtt_topic_zubereiten_start = "zubereiten_start"

mqtt_topic_zutaten = "zutaten"

mqtt_topic_hand = "hand"

mqtt_topic_statusmeldungen = "statusmeldungen"

mqtt_topic_mengen_doku = "mengen_doku"


#-------------------- Pumpenleitung in cl/min ----------------
---

# wird zur Berechnung der Einschaltzeit verwendet
```

```
pumpenleistung = 15


#-------------------- Globale Variablen -------------------


zutat_1 = 0

zutat_2 = 0

zutat_3 = 0

zutat_4 = 0

zutat_5 = 0

zutat_6 = 0

zutat_7 = 0

zutat_8 = 0

start_button = "OFF"

fuellstand_zutat_1 = 0

fuellstand_zutat_2 = 0

fuellstand_zutat_3 = 0

fuellstand_zutat_4 = 0

fuellstand_zutat_5 = 0

fuellstand_zutat_6 = 0

fuellstand_zutat_7 = 0

fuellstand_zutat_8 = 0

status_glas_vorhanden = "OFF"

letzter_glas_status = None

pumpe_1_hand = "OFF"

pumpe_2_hand = "OFF"

pumpe_3_hand = "OFF"
```

```
pumpe_4_hand = "OFF"

pumpe_5_hand = "OFF"

pumpe_6_hand = "OFF"

pumpe_7_hand = "OFF"

pumpe_8_hand = "OFF"

dosierung_zutat_1 = 0

dosierung_zutat_2 = 0

dosierung_zutat_3 = 0

dosierung_zutat_4 = 0

dosierung_zutat_5 = 0

dosierung_zutat_6 = 0

dosierung_zutat_7 = 0

dosierung_zutat_8 = 0

schwellwert_zutat_1 = 250

schwellwert_zutat_2 = 250

fuellstand_warnung_1_gesendet = False

fuellstand_warnung_2_gesendet = False

letzte_messung = time.ticks_ms()


#---------- Schwellwerte Füllstand in mm ----------

# Schwellwerte in Millimetern: Ab wann soll eine Warnung
gesendet werden, wenn eine FLasche fast leer ist?


schwellwert_zutat_1 = 250

schwellwert_zutat_2 = 250
```

```
#---------- I2C erzeugen ----------

# Zwei getrennte I2C-Busse für je einen VL53L0X
Füllstandssensor

# Diese Sensoren messen den Abstand zum Flüssigkeitsspiegel


i2c_1 = I2C(0, sda=sda_1, scl=scl_1)

i2c_2 = I2C(1, sda=sda_2, scl=scl_2)


#---------- Sensorobjekt erzeugen ----------


fuellstand_sensor_1 = VL53L0X(i2c_1)

fuellstand_sensor_2 = VL53L0X(i2c_2)


#---------- WLAN-Verbindung für Hauptprogramm ----------


def connect_to_wifi(WIFI_ssid, WIFI_password):

    wlan = network.WLAN(network.STA_IF)

    wlan.active(True)

    wlan.connect(WIFI_ssid, WIFI_password)

    while not wlan.isconnected():

        print("Verbinde mit WLAN...")

        time.sleep(1)

    print("Verbunden! IP:", wlan.ifconfig())


#---------- Subprogramm Zutaten aufschlüsseln ----------

# Diese Callback-Funktion wird aufgerufen, wenn MQTT-
Nachrichten zum Thema "zutaten" empfangen werden.
```

```
# Sie extrahiert die jeweilige Dosiermenge für jede Zutat aus
der empfangenen JSON-Nachricht

# und speichert sie in den globalen Variablen zutat_1 bis
zutat_8.



def sub_zutaten(topic, msg):

    global zutat_1, zutat_2, zutat_3, zutat_4, zutat_5,
zutat_6, zutat_7, zutat_8

    try:

        daten = ujson.loads(msg) # Umwandlung der empfangenen
Nachricht von JSON in ein Python-Dictionary

        zutat_1 = daten.get("zutat_1", 0)

        zutat_2 = daten.get("zutat_2", 0)

        zutat_3 = daten.get("zutat_3", 0)

        zutat_4 = daten.get("zutat_4", 0)

        zutat_5 = daten.get("zutat_5", 0)

        zutat_6 = daten.get("zutat_6", 0)

        zutat_7 = daten.get("zutat_7", 0)

        zutat_8 = daten.get("zutat_8", 0)

        print("Zutat 1:", zutat_1)

        print("Zutat 2:", zutat_2)

        print("Zutat 3:", zutat_3)

        print("Zutat 4:", zutat_4)

        print("Zutat 5:", zutat_5)

        print("Zutat 6:", zutat_6)

        print("Zutat 7:", zutat_7)

        print("Zutat 8:", zutat_8)
```

```
        gesamtmenge_zutaten = zutat_1 + zutat_2 + zutat_3 +
zutat_4 + zutat_5 + zutat_6 + zutat_7 + zutat_8



        #Gesamtmenge über MQTT versenden für Visualisierung



        nachricht_gesamtmenge_zutaten =
ujson.dumps({"Gesamtmenge_Zutaten": gesamtmenge_zutaten})

        mqtt_client.publish(mqtt_topic_statusmeldungen,
nachricht_gesamtmenge_zutaten)



    except Exception as e:

        print("Fehler beim Parsen:", e)



#---------- Subprogramm Zubereitung starten ----------

# Diese Funktion wird aufgerufen, wenn über MQTT das Signal
zum Start der Zubereitung kommt.

# Diese Narchicht soll nur empfangen werden, wenn auch ein
Glas vorhanden ist.



def sub_zubereitung_start(topic, msg):

    if glas_vorhanden.value() == 1:

        global start_button

        try:

            daten = ujson.loads(msg)

            start_button = daten.get("Start_Button", "OFF")

            print("Startbutton:", start_button)

        except Exception as e:

            print("Fehler beim Parsen:", e)
```

```
#---------- Subprogramm Handbetrieb ----------

# Diese Funktion verarbeitet MQTT-Nachrichten zum Thema "hand"

# und schaltet die Pumpen einzeln je nach empfangenem Befehl
ein oder aus.


def sub_handbetrieb(topic, msg):

    global pumpe_1_hand, pumpe_2_hand, pumpe_3_hand,
pumpe_4_hand

    global pumpe_5_hand, pumpe_6_hand, pumpe_7_hand,
pumpe_8_hand


    try:

        daten = ujson.loads(msg)


        pumpe_1_hand = daten.get("Pumpe1", "OFF")

        pumpe_2_hand = daten.get("Pumpe2", "OFF")

        pumpe_3_hand = daten.get("Pumpe3", "OFF")

        pumpe_4_hand = daten.get("Pumpe4", "OFF")

        pumpe_5_hand = daten.get("Pumpe5", "OFF")

        pumpe_6_hand = daten.get("Pumpe6", "OFF")

        pumpe_7_hand = daten.get("Pumpe7", "OFF")

        pumpe_8_hand = daten.get("Pumpe8", "OFF")


        pumpe_1.value(1 if pumpe_1_hand == "ON" else 0)

        pumpe_2.value(1 if pumpe_2_hand == "ON" else 0)

        pumpe_3.value(1 if pumpe_3_hand == "ON" else 0)
```

```
        pumpe_4.value(1 if pumpe_4_hand == "ON" else 0)
        pumpe_5.value(1 if pumpe_5_hand == "ON" else 0)
        pumpe_6.value(1 if pumpe_6_hand == "ON" else 0)
        pumpe_7.value(1 if pumpe_7_hand == "ON" else 0)
        pumpe_8.value(1 if pumpe_8_hand == "ON" else 0)

    except Exception as e:
        print("Fehler beim Parsen oder Setzen:", e)

#---------- Subprogramm Mischen ----------
# Unterprogramm zum Automatischen zusammenmischen der
Flüssigkeiten

def automatische_zubereitung():
    zutaten = [
        (zutat_1, pumpe_1),
        (zutat_2, pumpe_2),
        (zutat_3, pumpe_3),
        (zutat_4, pumpe_4),
        (zutat_5, pumpe_5),
        (zutat_6, pumpe_6),
        (zutat_7, pumpe_7),
        (zutat_8, pumpe_8),
    ]

    for index, (menge, pumpe) in enumerate(zutaten, start=1):
```

```
        if menge > 0:

            einschaltzeit = menge / pumpenleistung * 60  # 
Umrechnung cl → Sekunden

            print("Starte Pumpe", index, "für", einschaltzeit, 
"Sekunden")

            pumpe.value(1)

            time.sleep(einschaltzeit)

            pumpe.value(0)

        else:

            print("Zutat", index, "hat 0 cl – Pumpe wird nicht 
aktiviert")



    # Hier werden die Zutatenmengen über Node-RED an die 
DAtenbank gesendet



    nachricht_zutaten_datenbank = ujson.dumps({"Zutat1": 
zutat_1, "Zutat2": zutat_2, "Zutat3": zutat_3, "Zutat4": 
zutat_4, "Zutat5": zutat_5, "Zutat6": zutat_6, "Zutat7": 
zutat_7, "Zutat8": zutat_8 })

    mqtt_client.publish(mqtt_topic_mengen_doku, 
nachricht_zutaten_datenbank)



#---------- Subprogramm Callback ----------

# Dieses Unterprogramm richtet die Verschiedenen Callbacks ein



def globale_topicabfrage(topic, msg):

    topic = topic.decode()

    if topic == mqtt_topic_zubereiten_start:

        sub_zubereitung_start(topic, msg)

    elif topic == mqtt_topic_zutaten:
```

```
        sub_zutaten(topic, msg)
    elif topic == mqtt_topic_hand:
        sub_handbetrieb(topic, msg)


#---------- Subprogramm reconnect MQTT ----------
# Diese Funktion richtet die MQTT-Verbindung ein, wenn sie
verloren wurde


def reconnect():
    global mqtt_client
    try:
        mqtt_client.connect()
        mqtt_client.subscribe(mqtt_topic_zubereiten_start)
        mqtt_client.subscribe(mqtt_topic_zutaten)
        mqtt_client.subscribe(mqtt_topic_hand)
    except OSError as e:
        print("Verbindung fehlgeschlagen:", e)
        time.sleep(5)


#--------------- Aktivierung WLAN verbindung ---------------


connect_to_wifi(WIFI_ssid, WIFI_password)


#--------------- MQTT client erzeugen -------------


mqtt_client = MQTTClient(mqtt_client_id,
ip_adresse_mqtt_server, keepalive=30)
```

```
mqtt_client.set_callback(globale_topicabfrage)

mqtt_client.connect()

mqtt_client.subscribe(mqtt_topic_zubereiten_start)

mqtt_client.subscribe(mqtt_topic_zutaten)

mqtt_client.subscribe(mqtt_topic_hand)




#--------------- Hauptschleife -------------

# Diese Schleife wird dauerhaft ausgeführt und reagiert auf
das Startsignal vom MQTT-Server.

# Sobald das Signal "ON" empfangen wird, wird der Cocktail
gemäß den gespeicherten Zutatenmengen gemischt.




while True:

    try:

        mqtt_client.check_msg()




    except OSError as e:

        print("MQTT Fehler:", e)

        reconnect()


    # Hier wird der Status ob ein Glas vorhande ist per MQTT
versendet


    aktueller_status = glas_vorhanden.value()
```

```
    if aktueller_status != letzter_glas_status:
        if aktueller_status == 1:
            mqtt_client.publish(mqtt_topic_statusmeldungen,
ujson.dumps({"status_glas_vorhanden": "ON"}))

        else:
            mqtt_client.publish(mqtt_topic_statusmeldungen,
ujson.dumps({"status_glas_vorhanden": "OFF"}))

        letzter_glas_status = aktueller_status  #
Aktualisieren für den nächsten Vergleich

    if start_button == "ON" and glas_vorhanden.value() == 1:
        print("Starte automatische Zubereitung...")
        automatische_zubereitung()
        start_button = "OFF"  # Zurücksetzen, um mehrfachen
Start zu verhindern

    jetzt = time.ticks_ms()

    # Hier wird der Füllstand gemessen und verarbeitet

     # Nur alle 5 Sekunden messen
    if time.ticks_diff(jetzt, letzte_messung) >= 5000:
        fuellstand_zutat_1 = fuellstand_sensor_1.read() - 50
        fuellstand_zutat_2 = fuellstand_sensor_2.read() - 40
```

```
        # Prüfen, ob Schwellwert unterschritten wurde
        if fuellstand_zutat_1 < schwellwert_zutat_1:
            if not fuellstand_warnung_1_gesendet:

mqtt_client.publish(mqtt_topic_statusmeldungen,
                    ujson.dumps({"warnung_zutat_1":
f"Füllstand zu niedrig: {fuellstand_zutat_1} mm"}))
                fuellstand_warnung_1_gesendet = True
        else:
            fuellstand_warnung_1_gesendet = False

        if fuellstand_zutat_2 < schwellwert_zutat_2:
            if not fuellstand_warnung_2_gesendet:

mqtt_client.publish(mqtt_topic_statusmeldungen,
                    ujson.dumps({"warnung_zutat_2":
f"Füllstand zu niedrig: {fuellstand_zutat_2} mm"}))
                fuellstand_warnung_2_gesendet = True
        else:
            fuellstand_warnung_2_gesendet = False

        # Debug-Ausgabe
        print("Füllstand 1:", fuellstand_zutat_1)
        print("Füllstand 2:", fuellstand_zutat_2)

        letzte_messung = jetzt
```