**Projektarbeit Cocktailautomat**

**Cocktailautomat mit Steuerung über ESP32 – S3 und Raspberry Pi**

**Erstellt von Niklas Granel**

Inhalt

[**1. Allgemeine Informationen** 3](#_Toc197465703)

[**2. Projektbeschreibung** 3](#_Toc197465704)

[**3. Anforderungen und Funktionalitäten** 3](#_Toc197465705)

[**4. Benötigte Komponenten** 3](#_Toc197465706)

[**4.1 Hardware** 3](#_Toc197465707)

[**4.2 Software & Datenbank** 4](#_Toc197465708)

[**5. Systemarchitektur** 4](#_Toc197465709)

[**6. Zeitplanung (Meilensteine)** 4](#_Toc197465710)

[**7. Offene Fragen & Herausforderungen** 5](#_Toc197465711)

[**8. Fazit & Zielsetzung** 5](#_Toc197465712)

[**9. Installationsanleitung** 5](#_Toc197465713)

[**10. Bedienungsanleitung** 5](#_Toc197465714)

[**11. Technische Besonderheiten** 7](#_Toc197465715)

[**12. Persönliches Fazit** 7](#_Toc197465716)

[**13. Node-RED** 7](#_Toc197465717)

[**14. Datenbank** 11](#_Toc197465718)

[**15. Microphyten Code** 11](#_Toc197465719)

**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** | Datalogic950811290 | 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  PC817 Optokoppler  Relais Modul 3,3V | Für Node-RED und MQTT-Server  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

**Ein Bild, das Text, Elektronik, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.**

**6. Zeitplanung (Meilensteine)**

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

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

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.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

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**

Ein Bild, das Text, Reihe, Diagramm, Screenshot enthält.

KI-generierte Inhalte können fehlerhaft sein.

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

Der Inhalt einer Funktion Node:

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

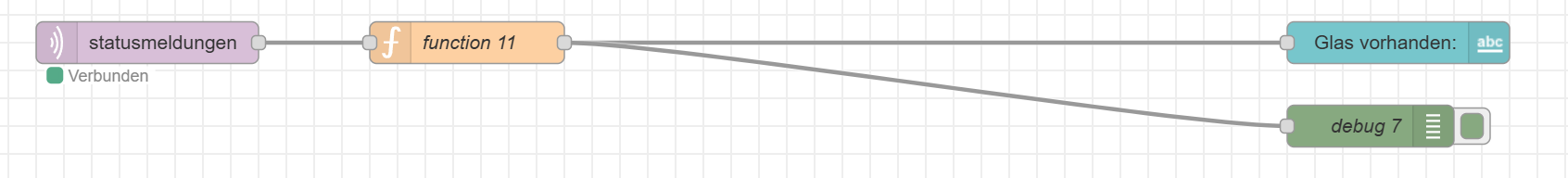
Dies ist als Beispiel function 1. Alle anderen Funktionen sind identisch. Natürlich ändert jede ich auf die entsprechende Zutat also zutat\_1, zutat\_2, usw.



In diesem Teil wir der Startbefehl verarbeitet

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

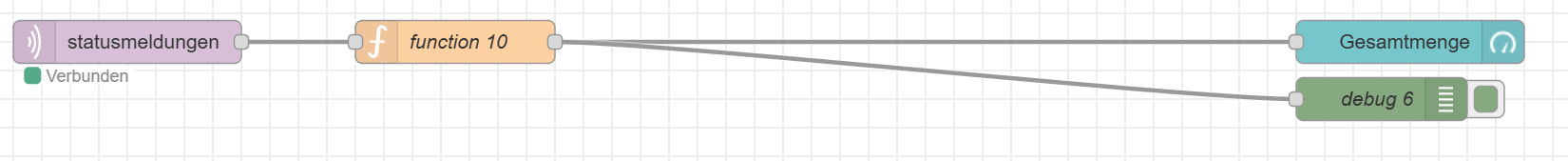


In diesem Teil wird die Nachricht verarbeitet, dass ein Glas vorhanden ist

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

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



Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Danach folget die Bedienung im Handbetrieb

Ein Bild, das Reihe, Text, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Screenshot, Zahl, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

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



Hier werden die Flüssigkeitsmengen empfangen in die Datenbank geschrieben.

Ein Bild, das Text, Software, Schrift, Webseite enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

**14. Datenbank**

Ein Bild, das Text, Reihe, Zahl, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Schrift, Reihe, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

**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