Fachhochschule
Dortmund

Bachelorarbeit

Oleksii Baida
Matrikelnummer 7210384

Sicherheits- & Steuerungssytem für das Haus

Bericht

20. Januar 2025

Inhaltsverzeichnis

1 Einleitung **3**

2 Grundlagen & Theorie **4**
2.1 Hardware . . . 4
2.1.1 Arduino . . . 4
2.1.2 ESP8266 . . . 4
2.1.3 Raspberry Pi . . . 4
2.1.4 ESP32 . . . 4
2.1.5 M5Stick . . . 4
2.1.6 BME680 . . . 4
2.1.7 VCNL . . . 4
2.2 Kommunikationsprotokolle . . . 4
2.2.1 HTTP . . . 4
2.2.2 MQTT . . . 4
2.3 Software . . . 4
2.3.1 PlatformIO . . . 4
2.3.2 Asyncio . . . 4
2.3.3 FastAPI . . . 4
2.3.4 SQLAlchemy . . . 4
2.3.5 Uvicorn . . . 4
2.3.6 HTML & TailwindCSS . . . 4
2.3.7 Javascript . . . 4
2.3.8 WebSocket . . . 4
2.3.9 Linux-Pakete für Raspberry Pi . . . 4

3 Konzeption das Systems **5**
3.1 Komponenten des Systems . . . 5
3.2 Architektur und Datenfluss . . . 5

4 Implementierung und praktische Umsetzung **7**
4.1 Einrichtung der Hardwarekomponenten . . . 7
4.1.1 Arduino . . . 7
4.1.2 Raspberry Pi . . . 7
4.1.3 Setup der ESP-Module . . . 7
4.1.4 M5Stick . . . 11
4.1.5 ESP32 . . . 13
4.2 Softwareentwicklung . . . 13
4.2.1 Webserver . . . 13
4.3 Datenbank . . . 13
4.4 Frontend . . . 13
4.5 Integration der Komponenten . . . 13

5 Ergebnisse und Diskussion **14**

6 Quellen **15**

Abbildungsverzeichnis **16**

Programmcode **16**

1 Einleitung

In einer Welt, die zunehmend von vernetzten Geräten und dem Internet der Dinge geprägt ist, wird die Entwicklung effizienter und benutzerfreundlicher Systeme zur Steuerung und Überwachung von Gebäuden immer relevanter. Im Jahr 2024 wurde in Deutschland die Anzahl von über 19 Millionen Haushalten, die ein oder mehrere smarte Geräte besaßen, verzeichnet. Es wird prognostiziert, dass sich diese Zahl innerhalb der nächsten drei Jahre verdoppeln wird [4].

Moderne Steuerungs- und Sicherheitssysteme tragen zur Effizienzsteigerung und Ressourcenschonung bei. Laut Günther Ohland, Vorstandsmitglied des Branchenverbands SSmarthome Initiative Deutschland", ermöglichen diese Systeme eine Reduktion des Heizenergieverbrauchs um 20 bis 30 Prozent [5]. Die Kosten für die smarte Technik rechnen sich in der Regel nach zwei Jahren. Die Systeme übernehmen ein Teil der täglichen Aufgaben, wie das Ein- und Ausschalten des Lichts, die Reglung der Raumtemperatur oder das Aufräumen des Hauses etc. Der Aufgabenbereich der Systeme ist dabei nur nach den Bedürfnissen der Benutzerinnen und Benutzer abgegrenzt.

Im Rahmen meiner Bachelorarbeit wird ein System zur Steuerung und Überwachung des Hauses entwickelt. Das Ziel dieser Arbeit ist die Erstellung einer Schnittstelle, die die Interaktion des Benutzers mit den Geräten in seinem Haushalt ermöglicht und den Benutzer über gefährliche Vorgänge in seinem Haus informiert.

Im Rahmen der Entwicklung dieses Systems wurden die aktuellen Technologien zur Erstellung eines Webinterfaces und zur Kommunikation zwischen den Geräten eingesetzt. **TODO Kurz erklären was in Kapitel 2,3,4,5 ... erklärt wird**

2 Grundlagen & Theorie

In diesem Abschnitt erfolgt die detaillierte Darstellung der technischen Informationen zu den verwendeten Komponenten und Technologien.

2.1 Hardware

2.1.1 Arduino

2.1.2 ESP8266

2.1.3 Raspberry Pi

2.1.4 ESP32

2.1.5 M5Stick

2.1.6 BME680

2.1.7 VCNL

2.2 Kommunikationsprotokolle

2.2.1 HTTP

2.2.2 MQTT

2.3 Software

2.3.1 PlatformIO

2.3.2 Asyncio

2.3.3 FastAPI

2.3.4 SQLAlchemy

2.3.5 Uvicorn

2.3.6 HTML & TailwindCSS

2.3.7 Javascript

2.3.8 WebSocket

2.3.9 Linux-Pakete für Raspberry Pi

3 Konzeption das Systems

Im Rahmen dieses Projektes wurde ein System entwickelt, welches die Funktionalitäten eines Kontroll- und Verwaltungssystems mit denen eines IoT-Systems vereint. Die Integration von Sensordaten und Benutzerinteraktionen stellt einen wesentlichen Aspekt des Systems dar. Die Realisierung erfolgt durch die Kombination verschiedener Technologien und Teilsysteme, darunter eine Webanwendung auf FastAPI, eine MQTT-Kommunikationsschicht und verschiedene Aktoren und Sensoren, die auf Arduino- oder ESP-Module basieren.

3.1 Komponenten des Systems

Das entwickelte System basiert auf einer modularen Architektur, die mehrere Komponenten integriert. Jede dieser Komponenten erfüllt eine spezifische Rolle im Gesamtsystem:

- **Raspberry Pi:** Der zentrale Server, der das lokale WLAN-Netzwerk bereitstellt, den MQTT-Broker hostet und die Webanwendung ausführt.
- **Arduino:** Ausgestattet mit mehreren Sensoren, die Gefahren wie Feuer und Gas erkennen und den Zugang zum Haus sichern. Entsprechende Meldungen werden an den Server gesendet.
- **M5Stick:** Angeschlossen an die Temperatur- und Lichtsensoren und sendet die aufgezeichneten Daten auf den Server.
- **ESP32:** TTTOOOOODDDDOOOO
- **Webserver:** Eine auf FastAPI basierende RESTful API, die Benutzern den Zugriff auf das System und die Steuerung von Geräten ermöglicht.
- **Datenbank:** Eine lokale Datenbank zur Speicherung von Benutzerinformationen und Gerätekonfigurationen.
- **Web-Anwendung:** Eine browserbasierte Benutzeroberfläche, die den Benutzern eine intuitive Steuerung und Visualisierung der Daten ermöglicht.

3.2 Architektur und Datenfluss

Der Raspberry Pi dient als zentraler Server des Systems. Der Minicomputer stellt ein WLAN-Netzwerk zur Verfügung und hostet den Webserver mit der Datenbank sowie den MQTT-Broker. Alle Benutzerinteraktionen, Sensordaten, Datenflüsse und Datenverarbeitungen finden auf dem Server statt. Somit ist das System stark zentralisiert. Das System ist somit stark zentralisiert und arbeitet nur lokal. Das bedeutet, dass sich alle Benutzer in einem lokalen Netzwerk mit dem Raspberry Pi befinden müssen.

Die Geräte verbinden sich mit dem WLAN, das vom Raspberry Pi zur Verfügung gestellt wird. Die Sensoren senden ihre Daten an den MQTT-Broker, der auf dem Raspberry Pi läuft. Die Aktoren abonnieren die Command-Topics mit der entsprechenden "Geräte-ID".

Im Kern des Systems befindet sich ein Webserver, der auf dem Raspberry Pi ausgeführt wird. Dieser Webserver stellt eine RESTful-API zur Verfügung. Für den Zugriff zu der API wurde eine Webseite aufgebaut. Die API bietet folgende Funktionen an:

- Authentifizierung und Autorisierung der Benutzer
- Verwaltung der Gerätekonfigurationen und Benutzerprofile

- Bereitstellung von Endpunkten zur Abfrage und Steuerung von IoT-Geräten
- Bidirektionale Kommunikation mit den IoT-Geräten

Die Benutzerdaten und Konfigurationen werden in der lokalen Datenbank auf dem Server gespeichert.

4 Implementierung und praktische Umsetzung

4.1 Einrichtung der Hardwarekomponenten

4.1.1 Arduino

Der Arduino mit den angeschlossenen Sensoren stellt das Sichterheitskomponente des Systems dar. Dieses Teilsystem erkennt die Gefahren von Gas und Feuer und alarmiert den Benutzer. Außerdem stellt der Arduino der gesicherte Zugang zu dem Haus durch die Eingabe einer PIN. Die Anschließung der Sensoren und Aktoren an Arduino ist in der PA1 [1] beschrieben.

Der Arduino kann nicht direkt mit einem WLAN verbunden werden, da er kein WLAN-Modul besitzt. Für diesen Zweck habe ich ein ESP8266-Modul verwendet. Der wird mit dem Arduino durch eine UART-Schnittstelle angeschlossenen und mit dem WLAN verbunden. Die Kommunikation zwischen Arduino und ESP8266 erfolgt über die serielle Schnittstelle. Der Arduino gibt die entsprechenden Kommanden durch Serial an ESP8266 weiter und der ESP8266 führt die Befehle aus. Die detaillierte Beschreibung zur Verbindung von Arduino und ESP8266 ist in dem Bericht zu meiner Projektarbeit 2 [2] Kapitel 4.1 zu finden.

Der Arduino sendet und empfängt die MQTT-Nachrichten via ESP8266. Ein Mal pro Sekunde sendet der Arduino eine MQTT-Nachricht in das Topic ßtatus/¡ID¿/ïn dem die Bereitschaft der angeschlossenen Sensoren geteilt wird. Bei der Erkennung einer Gefahr wird sofort der Alarm ausgelöst und eine MQTT-Nachricht in das Topic älarm/¡ID¿"mit dem entsprechenden Text gesendet.

4.1.2 Raspberry Pi

4.1.3 Setup der ESP-Module

In diesem Projekt werden drei verschiedene ESP-Module verwendet: ESP8266, ESP32 und der auf dem ESP32 basierende M5Stick. Jedes Modul erfüllt spezifische Aufgaben, aber alle Module müssen sowohl mit dem WLAN als auch mit dem MQTT-Broker verbunden sein. Daher ist die Implementierung der Funktion `setup()` für alle drei Module ähnlich aufgebaut.

Beim Start des ESP-Moduls wird die Funktion `setup()` (Listing 1) aufgerufen. Zunächst wird es versucht, die WLAN-Zugangsdaten aus dem EEPROM auszulesen (Zeile 5-10). Das EEPROM wird mit einer Größe von 128 Byte initialisiert. Dies ist notwendig, bevor Daten aus dem Speicher gelesen werden können. Es werden zwei Felder vom Typ `char` wurden mit den in den eckigen Klammern angegebenen Längen erstellt (Zeile 6-7), um die aus dem EEPROM gelesenen Daten zu speichern. Beide Felder sind zunächst mit Nullen gefüllt. Der Name des WLAN-Netzes (SSID) wird ab Adresse 0 des EEPROM ausgelesen und mit der Funktion `EEPROM.get(0, eeprom_ssid)` im Char-Feld gespeichert. Das Passwort wird ab Adresse 32 aus dem EEPROM ausgelesen.

Anschließend prft die Boolean-Funktion `is_valid_string()` (Listing 2), ob der übergebene String-Parameter eine Länge größer als 0 und kleiner als `max_length`, sowie eine korrekte Terminierung hat. Zusätzlich gibt die Funktion den Wert `False` zurück, wenn der String ein Standard-EEPROM-Zeichen mit dem Wert `0xFF` enthält.

Sind die gültigen Werte im EEPROM gespeichert, versucht das Modul, sich mit diesen Zugangsdaten mit dem WLAN zu verbinden (Zeile 14). Die Funktion `connect_wifi` (Listing 3) schaltet das WLAN-Modul in den Modus SStationüm, was die Verbindung mit anderen WLAN-Netzwerken erlaubt. Danach wird es versucht mit dem WLAN mit übergebenen Zugangsdaten zu verbinden. In der Variable `wifi_repeat` ist die Anzahl der Versuche zur Erstellung der Verbindung gespeichert. Jede Sekunde wird den Status der Verbindung überprüft. Wenn die Verbindung erfolgreich erstellt wurde, gibt die Funktion `True` aus.

Nach erfolgreicher Verbindung mit dem WLAN, wird die Funktion `connect_mqtt` (Listing 4) aufgerufen. Zunächst wird die Funktion `set_topics` aufgerufen, die die Topics zum Senden und Empfangen der Nachrichten erstellt. Dabei werden die Topics mit `DEVICE_ID` für jedes Gerät individuell formuliert. Der `mqttClient` ist ein Objekt der Klasse `PubSubClient`. Die Funktion `setServer()` erhält als Parameter die IP-Adresse des MQTT-Brokers sowie den Port, auf dem der Broker lauscht. Diese Parameter sind als Konstanten definiert. Die Callback-Funktion wird beim Empfang einer MQTT-Nachricht ausgeführt und ist für jedes Modul unterschiedlich aufgebaut. Anschließend verbindet sich das Modul mit dem MQTT-Broker und abonniert das `SUBSCRIBE_TOPIC`.

Falls die aus dem EEPROM ausgelesenen Daten ungültig sind oder keine erfolgreiche WLAN-Verbindung aufgebaut werden kann, wird die Funktion setup_ap() (Listing 7) aufgerufen. Diese Funktion versetzt das WLAN-Modul in den Modus „Access Point“ und stellt einen Webserver bereit. Die Konfiguration des Access Points erfolgt über die Befehle WiFi.softAP() und WiFi.softAPConfig(), wobei die Parameter des Access Points in Listing 6 definiert sind.

Der Webserver wird durch das Objekt `server` der Klasse `AsyncWebServer` zur Verfügung gestellt. Der Webserver wird mit dem Befehl `server.begin()` gestartet. Die HTTP-Route wird innerhalb der Funktion server.on() festgelegt. Beim Aufruf der Root-Route (/") wird eine HTTP-GET-Anfrage bearbeitet, die mit einer HTML-Seite beantwortet wird. Diese Seite, deren Inhalt in der Variablen `html_page` (Listing 6) gespeichert ist, enthält ein Formular zur Eingabe der WLAN-Zugangsdaten.

Wenn der Benutzer auf die Taste „Submit“ drückt, wird eine HTTP-POST-Anfrage an den Server gesendet. Die Zugangsdaten werden aus dem request-Objekt ausgelesen und auf ihre maximale Länge überprüft. Wenn die Daten gültig sind, werden sie im EEPROM gespeichert. Der EEPROM wird initialisiert, die neuen Zugangsdaten werden gespeichert, und die Änderungen mit `EEPROM.commit()` übernommen. Anschließend wird das ESP-Modul mit `ESP.restart()` neu gestartet, um die neuen WLAN-Daten zu verwenden.

Diese Setup-Konfiguration wird bei allen ESP-Modulen verwendet. Je nach Modul kann diese Funktion erweitert sein. Beispielsweise wird bei Modulen mit Display angezeigt, ob die Verbindung aufgebaut wurde.

```
1  void setup()
2  {
3    Serial.begin(9600);
4    // GET WiFi Daten aus EEPROM
5    EEPROM.begin(128);
6    char eeprom_ssid[MAX_SSID_LENGTH] = {0};
7    char eeprom_password[MAX_PASSWORD_LENGTH] = {0};
8    EEPROM.get(0, eeprom_ssid);
9    EEPROM.get(32, eeprom_password);
10   EEPROM.end();
11   // Daten gefunden
12   if (is_valid_string(eeprom_ssid, MAX_SSID_LENGTH) &&
       is_valid_string(eeprom_password, MAX_PASSWORD_LENGTH))
13   {
14     if (connect_wifi(eeprom_ssid, eeprom_password))
15     {
16       connect_mqtt();
17       return;
18     }
19   }
20   // keine WLAN-DAten gefunden
21   setup_ap();
```

```
}
```

Listing 1: ESP: setup

```
bool is_valid_string(char *data, int max_length)
{
  if (strlen(data) == 0 or strlen(data) > max_length)
    return false;
  for (int i = 0; i < max_length; i++)
  {
    if (data[i] == '\0')
      return true; // End of valid String
    if (data[i] == 0xFF) // Default EEPROM
      return false;
  }
  return false;
}
```

Listing 2: ESP: is_valid_string

```
bool connect_wifi(char *ssid, char *password)
{
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  for (uint8_t i = 0; i < wifi_repeat; i++)
  {
    if (WiFi.status() == WL_CONNECTED)
    {
      Serial.println(WiFi.localIP());
      return true;
    }
    delay(1000);
  }
  return false;
}
```

Listing 3: ESP: connect_wifi

```
void mqtt_connect()
{
  set_topics();
  mqttClient.setServer(MQTT_BROKER_ADRRESS, MQTT_PORT);
  mqttClient.setCallback(callback);
  for (int i = 0; i < wifi_repeat; i++)
  {
    if (mqttClient.connect(DEVICE_ID))
      mqttClient.subscribe(SUBSCRIBE_TOPIC);
      return;
    else
      delay(1000);
  }
}
```

Listing 4: ESP: connect_mqtt

```
void set_topics()
{
  SUBSCRIBE_TOPIC = (char *)malloc(strlen(TOPIC_COMMAND) + strlen(
     DEVICE_ID) + 2);
  PUBLISH_TOPIC = (char *)malloc(strlen("data") + strlen(DEVICE_ID)
     + 2);

  strcpy(SUBSCRIBE_TOPIC, TOPIC_COMMAND);
  strcat(SUBSCRIBE_TOPIC, "/");
  strcat(SUBSCRIBE_TOPIC, DEVICE_ID);

  strcpy(PUBLISH_TOPIC, "data");
  strcat(PUBLISH_TOPIC, "/");
  strcat(PUBLISH_TOPIC, DEVICE_ID);
}
```

Listing 5: ESP: set_topics

```
const char AP_SSID[] = "ESP8266";
const char AP_PASSWORD[] = "setupesp";
IPAddress ap_ip(10, 0, 0, 1);
IPAddress ap_gateway(10, 0, 0, 1);
IPAddress ap_subnet(255, 255, 255, 0);
AsyncWebServer server(80);
const String html_page = R"rawliteral(
  <html>
  <body>
    <h2>Wi-Fi Configuration</h2>
    <form action="/save" method="POST">
      SSID:<br>
      <input type="text" name="ssid" required><br>
      Password:<br>
      <input type="password" name="password" required><br><br>
      <input type="submit" value="Submit">
    </form>
  </body>
  </html>
)rawliteral";
```

Listing 6: ESP8266: ap_konfiguration

```
void setup_ap()
{
  WiFi.mode(WIFI_AP);
  WiFi.softAP(AP_SSID, AP_PASSWORD);
  WiFi.softAPConfig(ap_ip, ap_gateway, ap_subnet);

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
            { request->send(200, "text/html", html_page); });

  server.on("/save", HTTP_POST, [](AsyncWebServerRequest *request)
            {
              // get input data
```

```
13                  String ssid = request->getParam("ssid", true)->value()
                        ;
14                  String password = request->getParam("password", true)
                        ->value();
15                  if (ssid.length() > MAX_SSID_LENGTH - 1 || password.
                        length() > MAX_PASSWORD_LENGTH - 1)
16                  {
17                    return;
18                  }
19                  // String in char[]
20                  char new_ssid[MAX_SSID_LENGTH] = {0};
21                  strncpy(new_ssid, ssid.c_str(), MAX_SSID_LENGTH - 1);
22                  char new_password[MAX_PASSWORD_LENGTH] = {0};
23                  strncpy(new_password, password.c_str(),
                        MAX_PASSWORD_LENGTH - 1);
24
25                  // in EEPROM speichern
26                  EEPROM.begin(128);
27                  EEPROM.put(0, new_ssid);
28                  EEPROM.put(32, new_password);
29                  EEPROM.commit();
30                  EEPROM.end();
31
32                  request->send(200, "text/html", "WiFi saved. Rebooting
                        ...");
33                  ESP.restart(); });
34
35    server.begin();
36  }
```

Listing 7: ESP: setup_up

ESP8266 Der ESP8266 wird für die Verbindung des Arduinos mit dem WLAN verwendet. Zu seinem Aufgaben gehört das Senden und Empfangen der MQTT-Nachrichten in entsprechenden Topics. Wie in meinem Bericht zur Projektarbeit 2 [2] Kapitel 4.1 beschrieben, ist der ESP8266 über eine UART-Schnittstelle mit dem Arduino verbunden.

Nach der Einrichtung ist das Modul betriebsbereit. Bei jedem Pogrammdurchlauf wird die Funktion `readSerialData` aufgerufen. Diese liest die vom Arduino gesendeten Daten ab und formuliert die MQTT-Nachricht sowie das Topic. Die detaillierte Beschreibung ist in Projektarbeit 2 [2] Kapitel 4.1 zu finden.

4.1.4 M5Stick

Der M5Stick wird zur Überwachung der Licht- und Luftbedingungen eingesetzt. Die Idee ist, dass der Sensor die aktuelle Wetterbedingungen überwacht und diese an dem Server sendet. Für diesen Zweck soll der Sensor an dem Fenster draußen positioniert werden.

Am Modul sind die Sensoren BME680 und VCNL4040 angebunden. Der BME680 erfasst Umweltdaten wie Temperatur, Luftfeuchtigkeit und Gaswiderstand. Der VCNL4040 dient als Lichtsensor und misst Lichtbedingungen wie Umgebungshelligkeit, weißes Licht sowie Annäherung. Diese Daten werden an dem Server via MQTT-Nachrichten gesendet.

Für die Verwendung der beiden Sensoren im Programmcode müssen die entsprechenden Bibliotheken importiert werden und die Klassenobjekten für die Sensoren erstellt werden (Listing

8).

```
#include <Adafruit_BME680.h>
#include <Adafruit_VCNL4040.h>
Adafruit_BME680 bme_sensor;
Adafruit_VCNL4040 vcnl4040 = Adafruit_VCNL4040();
```

Listing 8: M5Stick: Sensoren

Setup Beim Starten des Moduls wird die Funktion `setup()` (Listing 9) aufgerufen. Nach der Initialisierung der seriellen und I2C-Verbindung wird es versucht, die WLAN-Zugangsdaten aus dem EEPROM abzulesen.

```
void setup()
{
  Serial.begin(115200);
  Wire.begin();
  m5_setup();
  EEPROM.begin(128);

  char eeprom_ssid[MAX_SSID_LENGTH] = {0};
  char eeprom_password[MAX_PASSWORD_LENGTH] = {0};
  EEPROM.get(0, eeprom_ssid);
  EEPROM.get(32, eeprom_password);
  EEPROM.end();
  // Daten gefunden
  if (is_valid_string(eeprom_ssid, MAX_SSID_LENGTH) &&
     is_valid_string(eeprom_password, MAX_PASSWORD_LENGTH))
  {
    Serial.println("STRING VALID");
    if (wifi_connect(eeprom_ssid, eeprom_password))
    {
      i2cScan.begin(SDA2, SCL2, 400000);
      mqtt_connect();
    }
    else
    {
      setup_ap();
    }
    vcnl_setup();
    bme_setup();
  }
  else
  { // keine WLAN-DAten gefunden
    setup_ap();
  }
}
```

Listing 9: M5Stick: setup

4.1.5 ESP32

4.2 Softwareentwicklung

4.2.1 Webserver

4.3 Datenbank

4.4 Frontend

4.5 Integration der Komponenten

MQtt client, erhalten Daten von Broker etc

5 Ergebnisse und Diskussion

6 Quellen

Literatur

[1] O. Baida, *Anbindung der Sensoren und Aktoren an den Arduino zur Realisierung eines Sicherheitssystems*, Projektarbeit 1, 2024.

[2] O. Baida, Projektarbeit 2 *Sicherheitssystem fur das Haus basierend auf Arduino, ESP8266 & Raspberry Pi* https://github.com/oleksiibaida/PA2.git

[3]

[4] Statista, *„Smart Home - Anzahl der Haushalte in Deutschland 2028“*, Zugegriffen: 13. Januar 2025. [Online]. Verfügbar unter https://de.statista.com/prognosen/885611/anzahl-der-smart-home-haushalte-in-deutschland

[5] J. Breithut, *„Strom und Heizung: Wann ein Smart Home wirklich beim Energiesparen hilft“*, Der Spiegel, 17. Juli 2022. Zugegriffen: 13. Januar 2025. [Online]. Verfügbar unter: https://www.spiegel.de/netzwelt/gadgets/strom-und-heizung-wann-ein-smart-home-wirklich-beim-energiesparen-hilft-a-ffb4b710-

Links zur verwendeten Hardware:

[6] Arduino.cc, *Arduino UNO*, https://docs.arduino.cc/hardware/uno-rev3/

[7] Raspberry Pi Foundation, *Raspberry Pi 1 B+*, https://www.raspberrypi.com/products/raspberry-pi-1-model-b-plus/

[8] Espressif, *ESP8266*, https://www.espressif.com/, https://www.electronicwings.com/sensors-modules/esp8266-wifi-module

Links zur verwendeten Software:

[9] Dr Andy Stanford-Clark, Arlen Nipper, *Message Queuing Telemetry Transport*, https://mqtt.org/

[10] Guido van Rossum, Python Software Foundation, *Python*, https://www.python.org/

[11] Telegram FZ-LLC, *Telegram Messenger*, https://github.com//telegramdesktop/tdesktop

Linux-Packete:

[12] Jouni Malinen, *hostapd*, https://w1.fi/hostapd/, Zugriff am: 19. September 2024.

[13] Simon Kelley, *dnsmasq*, https://dnsmasq.org/doc.html, Zugriff am: 20. September 2024.

[14] Eclipse Foundation, *Eclipse Mosquitto*, https://mosquitto.org/

ESP- und Arduino-Bibliotheken

[15] Knolleary, *PubSubClient*, https://pubsubclient.knolleary.net/, Zugriff am: 21. Oktober 2024.

[16] ESPWIFI.h, https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html

[17] EEPROM.h, https://docs.arduino.cc/learn/built-in-libraries/eeprom/

[18] Keypad.h https://docs.arduino.cc/libraries/keypad/

[19] R. Scholz, *Syncloop*, Persönliche Mitteilungen

Python-Bibliotheken

[20] Pierre Fersing, Roger Light *paho-mqtt*, https://pypi.org/project/paho-mqtt/, Zugriff am: 21. Oktober 2024.

[21] Open Source, *python-telegram-bot*, https://docs.python-telegram-bot.org/en/v21.6/

[22] Python Software Foundation, *json*, https://docs.python.org/3/library/json.html

[23] Python Software Foundation, *threading*, https://docs.python.org/3/library/threading.html

[24] Python Software Foundation, *queue*, https://docs.python.org/3/library/queue.html

[25] Gerhard Häring, *sqlite3*, https://docs.python.org/3/library/sqlite3.html

[26] Lawrence Hudson, *pyzbar*, https://github.com/NaturalHistoryMuseum/pyzbar/

[27] Intel, *OpenCV*, https://github.com/opencv/opencv-python

[28] Aio-Libs, *aiohttp*, https://github.com/aio-libs/aiohttp

Abbildungsverzeichnis

Tabellenverzeichnis

Programmcode

1 ESP: setup 8
2 ESP: is_valid_string 9
3 ESP: connect_wifi 9
4 ESP: connect_mqtt 9
5 ESP: set_topics 10
6 ESP8266: ap_konfiguration 10
7 ESP: setup_up 10
8 M5Stick: Sensoren 12
9 M5Stick: setup 12