HOCHSCHULE für Angewandte Wissenschaften
Fachhochschule LANDSHUT · UNIVERSITY of Applied Sciences
Hochschule für Technik, Sozialwesen, Wirtschaft

FAKULTÄT Informatik


# Bachelorarbeit

## Entwicklung eines kostengünstigen und universellen Prototypen für Proof-of-Concept im Bereich Smart City

Daniel Schmuckermeier

Betreuer: Prof. Dr. Abdelmajid Khelil

**ERKLÄRUNG ZUR BACHELORARBEIT**

Schmuckermeier, Daniel

# Hochschule Landshut
# Fakultät Informatik

Hiermit erkläre ich, dass ich die Arbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt, sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

26.2.21 ..... (Datum)

Schmuckermeier ..... (Unterschrift des Studierenden)

# *Abstract*

*Das Ziel dieser Arbeit ist es, einen universellen und kostengünstigen Prototypen zu entwickeln. Dessen Zweck darin liegt, als Proof-of-Concept verwendet zu werden. Das heißt es ist eine erste, praktische Hilfe in der Anforderungsanalyse, um die Umgebung und den Use-Case sehr gut abschätzen zu können. In Anbetracht der Klimaproblematik müssen alltägliche Prozesse, wie zum Beispiel die Müllabfuhr, noch effizienter arbeiten, um ihr entgegen zu wirken. Die Vorgehensweise war es, verschiedene Hardware-Plattformen miteinander zu vergleichen und nach zuvor definierten Anforderungen zu bewerten. Der Sinn dahinter war es, das beste Board für diesen Fall zu ermitteln, da dies die Grundlage für alle weiteren Entscheidungen bildet. Darauf aufbauend wurde die beste Architektur und der Zugang, für die Boards, zum Internet bestimmt. Nachdem alle Entscheidungen getroffen wurden, wurde der Prototyp in der Praxis umgesetzt. Als Use-Case wurde eine Home-Build Kameraanlage realisiert. Hierbei sollen Kameras einfach hinzugefügt oder entfernt werden können. Dazu wurde ein Backend und ein Frontend aufgebaut und großen Wert auf die Benutzerfreundlichkeit gelegt. Dieser Prototyp wird am Ende auf seine anfangs gestellten Anforderungen überprüft. Als Ergebnis lässt sich festhalten, dass es alle Anforderungen sehr gut umsetzt, bis auf eine. Diese ist jedoch dem gewählten Use-Case geschuldet. Somit lässt er sich ideal in der Anforderungsanalyse verwenden um die Anforderungen des, gerade zu entwickelnen Projekts, genauestens zu bestimmen. Das spart enorm Kosten und Zeit ein.*

# Inhaltsverzeichnis

# 1 Einleitung

Viele Prozesse im Alltag besitzen Potenzial - nachhaltig - optimiert zu werden. Aufgrund vieler Faktoren wird das Optimum nicht ausgereizt. Zu den Faktoren zählen: großer Aufwand, zu wenige Daten, Energieverbrauch oder hohe Kosten. Würde man beispielsweise die Mülltonnen einer Stadt mit entsprechender Technik, die den Füllstand anzeigt, ausstatten, könnte bei hohem Füllstand die Müllabfuhr die Mülltonne entleeren. Das Entleeren der Tonne bei hohem Füllstand führt zu optimierten Entsorgungswegen. So werden nur die Mülltonnen geleert die einen bestimmten Schwellwert überschreiten, unabhängig davon in welchen Zeitintervallen, dies erforderlich ist. Durch das gezielte Entleeren nach Bedarf sinkt der CO2-Ausstoß enorm. Momentan ist so ein Vorhaben wirtschaftlich nicht rentabel, weil keine Echtzeitdaten zum Füllstand vorliegen und die Technik hohe Kosten verursacht.

## 1.1 Problemstellung und Ziel der Arbeit

Auf den Markt fehlt ein kostengünstiges Produkt, welches es erlaubt auf einfache Art und Weise Sensoren und ihre Steuerung miteinander zu vernetzen. Damit auch Bastler, zum Beispiel ihren Haushalt optimieren können. Das soll dazu beitragen, dass noch mehr Abläufe erkannt werden, die optimiert werden können. Dazu soll das Produkt ein Interface anbieten, mit diesem die Prozesse überwacht, kontrolliert und gesteuert werden. Die neu gewonnenen Daten können für weitere Optimierungen im Ablauf verwendet werden. So wird nachhaltig für eine grünerere Erde gesorgt.

Das langfristige Ziel ist es, Smart Cities zu erschaffen. Diese zeichnen sich durch hohe Nachhaltigkeit und den Umgang mit modernen Technologien aus. Mit Einbezug der Verwaltung, Mobilität, Energie und Kommunikation wird auch die Lebensqualität steigen (vgl. N. L. Dipl.-Ing. (FH) Stefan Luber 2016).

Ziel dieser Arbeit ist es, einen Prototypen zu entwickeln, der für die erste Phase einer Projektentwicklung Einhalt findet. Das bedeutet, um die Anforderungen

eines Projekts möglichst präzise zu definieren, muss zuerst die Umgebung und der Use-Case genau abgebildet werden. Dazu wird ein Proof-of-Concept erstellt. Das geschieht mit Allround-Prototypen, die viele Fälle abbilden können. Sind die Anforderungen bekannt, so kann ein, speziell für den Use-Case entwickeltes Produkt, deutlich effizienter und zielorientierter die Anforderungen erfüllen. Dieser Allround-Prototyp soll mit dem Konzept der Smart Cities harmonieren. Das heißt: benutzerfreundlich und skalierbar, da Städte auch in Zukunft wachsen werden. Ebenso geringe Kosten, geringer Energieverbrauch, weitreichende Zugriffsmöglichkeiten und ein Interface, um die Prozesse zu beobachten, kontrollieren und steuern zu können, sind von Bedeutung.

## 1.2 Struktur der Arbeit

Im ersten Kapitel werden die Anforderungen des Prototypen für den bislang unbekannten Use-Case definiert. Dann folgt die Auswahl verschiedener Hardware-Boards, Programmiersprachen, Architekturen und Verbindungsmöglichkeiten zum Internet. Diese werden nach den zuvor definierten Anforderungen bewertet und ausgewählt. Nach der Auswahl, wird ein konkreter Anwendungsfall realisiert. Dies wird eine kamerabasierte Anlage sein, die jederzeit mit weiteren Kameras erweitert werden kann. Dabei wird auf das Backend, das Frontend und die Benutzerfreundlichkeit eingegangen. Am Ende der Umsetzung, wird der Prototyp anhand der anfangs definierten Anforderungen bewertet. Zu guter Letzt folgt ein Fazit über die Arbeit. Ebenso wird ein Ausblick für weitere Entwicklungschritte zum Prototypen aufgezeigt.

# 2 Anforderungen an den Prototypen

Zuerst muss definiert werden, welche Eigenschaften dieser Prototyp erfüllen muss. Zu den Merkmalen zählen: Skalierbarkeit und Modularität. Des Weiteren zählt zu den qualitativen Merkmalen eine einfache und benutzerfreundliche Verwendung. Dazu soll sie effizient und sicher im Umgang mit Daten agieren und zugleich einen geringen Energieverbrauch aufweisen. Zusätzlich soll ein weitreichender Zugriff möglich sein, um die Prozesse aus mindestens 100 Meter beobachten und beeinflussen zu können.

Darunter werden folgende Annahmen getroffen: Unter Skalierbarkeit wird verstanden, dass es problemlos in der Größe des Netzes variieren kann. Dies kann auf zwei verschiedene Arten geschehen. Einmal, dass der Prototyp jederzeit und unkompliziert zu einem bereits bestehenden Netz hinzugefügt werden kann. Ebenso sollen verschiedene Sensoren und anderweitige Module zu einem Prototypen ergänzt werden können. Dies wird auch unter Modularität aufgefasst. Dies ermöglicht es, das in einem Netz, Prototypen mit verschiedenen Sensoren - wie im Baukastenprinzip - zusammengestellt werden können. Das führt dazu, dass der Prototyp möglichst viele Anwendungsfälle abdecken kann. So kann der Prototyp universell eingesetzt werden und für jeden Anwendungsfall individuell angepasst werden. Ein Produkt welches in Smart Cities Platz findet, muss mindestens Größenordnungen von $10^n$ , mit $n >> 2$ standhalten. Genauere Abschätzungen hängen immer von der Größe der Stadt und dem Anwendungsfall selbst ab. Als Mindestwert für den Proof-of-Concept-Prototypen wird ein skalierbarer Wert von zehn Geräten gleichzeitig angesehen. Im Bestenfall sollten es mehr sein, um auch etwas größere Use-Cases abbilden zu können. Bei der Energiebetrachtung ist der wichtigste Aspekt, der umweltschonende. Für den Einsatz in Smart City sollte die Laufzeit vom batteriebetriebenen Produkten mindestens ein paar Monate, idealerweise ein paar Jahre sein. Dennoch hängt es auch dort stark vom Anwendungsfall ab. Ein Temperatursensor, der

alle paar Stunden ein paar Werte sendet und den Rest der Zeit schläft hat andere Anforderungen, als eine Kamera die permament läuft. Ebenso sollte man den Energiefaktor immer gesondert betrachten. Befindet sich das Produkt in einer sonnigen Umgebung kann man mittels Solarpanels den Verbrauch entgegenwirken. Genauso bei sicherheitskritschen Systemen, sollten die Energiebetrachtung eine untergeordnete Rolle spielen. Da wäre es vorteilhaft auf eine dauerhafte Stromversorgung umzusteigen. Den Prototypen soll man mit einer Batterie über ein paar Tage, idealerweise ein paar Wochen, versorgen können, je nach Anwendungsfall. Das sollte ausreichen, um genug Daten für eine gezielte Anforderungsanalyse zu geben. Als Referenzwert wird bei der Batterie von einer 9 Volt Blockbatterie ausgegangen, mit einer Batteriekapazität von 1200 mAh. Mit dieser Formel wird die Batterielebensdauer im folgenden berechnet. Jene Annahmen können in der Realität abweichen. Diese dienen lediglich zu Vergleichszwecken zwischen den Hardwareboards.

$$\text{Batterielebensdauer in h} = \frac{\text{Batteriekapazität in mAh}}{\text{Laststrom in mA}}$$

Anforderung an die Benutzerfreundlichkeit, ist eine intuitive Bedienung und ein selbsterklärendes Feedbacksystem. Diese soll es erlauben, Projekte einfach abbilden zu können und schnell erste Ergebnisse zu produzieren.

# 3 Auswahl der Hardware-Plattform, Architektur und Anbinden an das Internet

Als nächstes muss abgewogen werden, welche Boards, Programmiersprache, Architektur und Möglichkeit sich mit dem Internet zu verbinden, die Beste ist, um die Anforderungen erfolgreich abzubilden. Das wird nun sukzessiv und aufbauend auf vorherige Entscheidungen behandelt. Weshalb als Netz das Internet gewählt wurde, wird im Kapitel 3.4 erläutert.

## 3.1 Auswahl der Hardwareboards

Im Folgenden werden die Boards Arduino Uno, der Raspberry Pi 4 Model B und dem ESP32 miteinander verglichen. Vom Arduino Projekt und der Raspberry Pi Familie gibt es weitere Boards, die jedoch alle nur für spezielle Fälle geeignet oder bereits veraltet sind. Zum Beispiel der Arduino Due, der mit "einer wesentlich höheren Leistung" (Mazzari 2016) verfügt. Dieses Board ist für ein Projekt perfekt prädestiniert mit anspruchsvollen Algorithmen oder Anforderungen im Bereich der künstlichen Intelligenz.

Die Kriterien, nach denen bewertet wird, sind:

- die Leistung der Hardware
- Stromverbrauch und Energiesparoptionen
- Preis
- Größe der Boards
- Anzahl der Pins

Ebenso werden Möglichkeiten gesucht, die sich für die Steigerung der Benutzerfreundlichkeit eignen. Auch spielt der Aspekt der Speichergröße des Boards eine wesentliche Rolle, damit die gesamten Quellcode-Zeilen Platz finden.

## Arduino Uno

Seinen Ursprung hat das Arduino Projekt in Italien gefunden. Arduino ist ein quelloffenes Projekt, sowohl auf der Hardware als auch auf der Software-Seite. Der Arduino ist eine Mikrocontroller-Einheit (microcontroller unit - MCU). Das heißt er ist nur für simple Aufgaben konzipiert. Sie lesen Eingabesignale, verarbeiten diese und leiten daraus Ausgabesignale ab. Praktisch betrachtet heißt das, als Input wird eine Form von Sensordaten (Temperatur, Luftfeuchtigkeit, etc.) oder software-seitig, durch z.B. Twitter, entgegengenommen. Dieses Signal wird verarbeitet und kann ein Ausgangssignal, einen Motor, eine LED, etc., steuern. Das Ziel von Arduino-Projekten ist die Interaktion und Automatisierung der Umgebung (vgl. Lang 2020b, vgl. Carolo 2020).

Das beliebteste unter den Arduino Boards ist das Arduino Uno. Als Prozessor ist ein ATmega328P verbaut. Somit haben wir 32 KB als Flash Memory, 2 KB als SRAM und 1 KB als EEPROM. Ein Uno ist 68,6 mm lang und 53,4 mm breit. Mit einem Gewicht von 25 Gramm kommt es einer gewöhnlichen Zigarettenschachtel sehr nahe. Dazu hat es 14 digitale Pins, wovon 6 in der Lage sind Pulsweitenmodulation, d.h. analoge Signale zu erzeugen. Des Weiteren gibt es 6 analoge Pins. Der marktübliche Preis beträgt 20 Euro (alle Preise - Stand Januar 2021) (vgl. Arduino o.D.[a]). Im Normalbetrieb, ohne LED, verbraucht er ca. 45 mA (ca. 27h Batterielaufzeit) (mit LED - ca. 95 mA). Mit einer Sleep-Bibliothek kann der Verbrauch um weitere 25 Prozent gesenkt werden, auf ca. 34 mA (ca. 35h). Um den Jackpot zu knacken, muss man die interne Stromversorgung von 5 Volt auf 3,3 umstellen und die Default Clock von 16 MHz auf 8. Damit kann man über 90 Prozent an Energie einsparen. Das entspreche einer Batterielaufzeit von über 11 Tagen (vgl. Ramos 2013, vgl. Kobbe 2017, vgl. David 2019). Als Besonderheit hat er einen USB-B Buchse, mit dieser kann er am PC oder einer Stromquelle angeschlossen werden. Ebenso lassen sich Erweiterungen sogenannte Shields auf ein Board ergänzen. Diese Hardware-seitigen Erweiterungskarten runden das System mit zusätzlicher Funktionalität unter anderem WiFi, Bluetooth, 4G, Bildschirm und weiteren Optionen ab. Zusätzlich kann eine integrierte LED für ein Feedbacksystem verwendet werden.

## Raspberry Pi 4 Model B

Das Board britischer Herkunft ist ein Einplatinencomputer (single-board computer - SBC) von der Raspberry Pi Foundation. Ein Einplatinencomputer wird da-

durch charakterisiert, dass sämtliche zum Betrieb benötigten elektrischen Komponenten auf einer einzigen Leiterplatte zusammengefasst sind. Die Raspberry Pi Produkte sind vollwertige Mini-Computer, ausgestattet mit einem Prozessor, RAM, Video- und Tonausgabe und sämtliche Netzwerk und USB-Anschlüsse. Durch die große Vielfalt wird er gerne als Medienserver, Retrokonsole oder Netzwerkspeicher verwendet. Durch seine GPIO Schnittstellen kann es mit Sensoren und anderen Modulen aller Art kommunizieren und damit z.B. eine Robotersteuerung umsetzen (vgl. heise-online 2021, vgl. Carolo 2020). Herr Carolo beschreibt den Unterschied zwischen Einplatinencomputern und Mikrocontroller-Einheiten, wie den Arduino, sehr treffend: "While a microprocessor is like a brain, a microcontroller is an entire body, being a whole computer in a chip". Daraus folgert er auch: "In practice, this means that SBCs (such as Raspberry Pis) are capable of performing some tasks that MCUs (like Arduinos) can't handle. But on the other hand, an SBC might offer too much for a simpler application"(Carolo 2020).

Der Raspberry Pi 4 Modell B hat einen Quad core Cortex-A72 verbaut. Als Speicher kann man zwischen 1/2/4 oder 8 GB RAM wählen. Dazu wird eine microSD für das Betriebsystem benötigt - in beliebiger Größe. Seine Abmessungen lauten in der Länge 93,0 mm und in der Breite 63,5 mm. Dazu wiegt er 46 Gramm. Er besitzt 40 GPIO Pins, d.h. "dass diese Schnittstelle für Ein- und Ausgaben genutzt werden kann, jedoch nicht näher spezifiziert ist. Somit kann man diese Schnittstelle selber nach eigenem Ermessen verwenden" (Hannes Heusel 2020). Der Preis für das billigste Model liegt bei 40 Euro, das teuerste Model bei 77 Euro (vgl. Foundation o.D.). Im Leerlauf-Modus verbraucht er ca. 560 mA. Die Batterie wäre nach etwa 2h entladen (vgl. Eames 2019, vgl. Geerling 2019). Um weitere Energieeinsparungen vorzunehmen muss man Dienste abstellen. So kann man den HDMI Output einstellen um weitere 25 - 30 mA einzusparen. Auch lassen sich WiFi und Bluetooth Funktionen einstellen, die CPU drosseln und die LEDs auf den Board deaktivieren. Im Extremfall lassen sich geschätzt weitere 200 bis 300 mA einsparen (vgl. Rush 2018, vgl. Geerling 2019). Weitere Besonderheiten des Boards sind zwei Micro HDMI Port, USB 2 und USB 3 Schnittstellen, Gigabit-Ethernet Anschluss und die Möglichkeit mit Bluetooth und Wlan-Netzen zu kommunizieren. Ebenso, dass man per mircoSD Karte verschiedene Betriebssysteme darauf booten kann (vgl. Foundation o.D.). Zur Benutzerfreundlichkeit könnte man die LED, die Audiosignale oder ein 1080p Bild per Micro HDMI verwenden.

### ESP32

Das letzte Board kommt aus China von Espressif. Das ist der Nachfolger von äußerst beliebten ESP8266. Der ESP32 gehört zur Klasse der System-on-a-Chip (SoC). Darunter wird verstanden, dass große oder alle Funktionen eines Systems auf einen Chip untergebracht sind. Verglichen mit einem PC finden sich die meisten Bauteile, wie Mainboard, Grafikkarte oder Arbeitsspeicher, auf einen einzelnen Chip. Herr Lang von heise schreibt dazu: "Das macht die Systeme extrem klein und sie verbrauchen recht wenig Energie. Und somit sind SoCs die perfekte Basis für IoT-Produkte, wie Smartwatches oder smarte Heizungssteuerungen im privaten Bereich sowie etliche Produkte für die Industrie, wie zum Beispiel medizinisches Gerät" (Lang 2020a, (vgl. Dipl. Betriebswirt Otto Geißler 2018).

Es gibt nicht den einen Chip, sondern eine ganze Familie. Da sie intern keine allzu großen Unterschiede aufweisen, werden sie als Gesamtes betrachtet. Die Prozessoren bestehen aus einem oder zwei Kernen von Typ Xtensa LX6. Dazu gibt es 4/8/16 MB Flash Speicher und 320 bis 512 KB SRAM. Die kleinsten Boards sind 13 mm lang und 19 mm breit. Die größten 18 mm lang und 31,4 mm breit. Im Durchschnitt sind sie 10 Gramm schwer. Entweder haben sie 38 oder 55 Pins. Dazu zählen unter anderem 18 Analog-zu-Digital Wandler, 16 Pulsweitenmodulation Output Kanäle und 10 kapazitive Touch Sensor Pins. Der Preis liegt zwischen 4 und 9 Euro (vgl. Espressif o.D.(d), vgl. Espressif o.D.(e)). Die ESP32 bieten einige Energiemodi an. Den Active, Modem Sleep, Light Sleep, Deep Sleep und Hibernation. Im Active Mode sind alle Funktionen wie WiFi, Bluetooth oder der CPU gleichzeitig aktiv. Da liegt der Stromverbrauch bei ca. 80 - 240 mA. Im Modem Sleep sind WiFi und Bluetooth deaktiviert und die CPU passt die Taktfrequenz an (3 - 20 mA). Der Light Sleep ist dem Modem Sleep sehr ähnlich, nur dass die Taktfrequenz konstant gesetzt wird (0,8 mA) - beim Deep Sleep wird auch die CPU deaktiviert. Das führt zu 10 μA Energieverbrauch und nun ist nur noch die Echtzeituhr aktiv und der Co Prozessor. Zuletzt zum Hibernation-Modus, auch Winterschlaf auf Deutsch übersetzt. Dort ist nur die Echtzeituhr aktiv. Der Verbrauch beträgt dann bis zu 2,5 μA. Das führt zur einer Batterielebensdauer, in idealer Umgebung, von über 54 Jahren. Um den ESP32 zu wecken gibt es drei Möglichkeiten: Timer Wake Up, Touch Wake Up und External Wake Up. Beim Timer Wake Up wird nach einer zuvor festgelegten Zeitperiode, der ESP32 aufgeweckt. Die Touch Funktion wird von den kapazitiven GPIOs unterstützt. Bei Berührung des entsprechenden GPIO

Kanal, wird der ESP32 wach. Beim Externen Modus wird ein Signal von außen verarbeitet (Tastendruck, Temperaturänderung, etc.) und führt zu Änderung des Energiemodus (vgl. lastminuteengineers o.D., vgl. Helmut 2018, vgl. Rui Santos 2019b).

Interessant ist noch zu erwähnen, dass die ESP32 ein eignes lokales Netz aufmachen können, per Access Point. Die Benutzerfreundlichkeit lässt sich durch eine eingebaute LED erweitern. Zusätzlich sollte man die integrierte Funktionalität eines Webserver nicht außer Acht lassen.

### Entscheidung: ESP32

Aufgrund der Merkmale nach denen entschieden wird (Leistung, Stromverbrauch und Energiesparoptionen, Preis, Größe der Boards, Anzahl der Pins), fällt die Wahl ganz klar auf den ESP32. Er ist das kleinste Board, hat einen geringen Stromverbrauch und viele weitere Energiesparoptionen, um jede Anwendung energieeffizient zu betreiben. Dazu ist der ESP32 das günstigste Board von all den verglichenen - hat viele Pins, darunter auch interessante, wie die kapazitiven GPIOs, die viele weitere Anwendungsfälle erlauben. Und genügend Speicherplatz mit mindestens 4 MB Flash. Auch der integrierte Webserver und die Access Point-Alternative bieten weitere aussichtsreiche Optionen für die Gestaltung des Interface. Auch wenn der Raspberry Pi deutlich mehr Leistung bietet, erkennbar am großen Unterschied zwischen den beiden RAM (max. 512 KB zu 8 GB) wird diese nicht benötigt. Da man durch das Baukastenprinzip die ESP32 vielfach erweitern kann. Somit ist ein besseres Leistungs/Nutzen Verhältnis gegenüber den Raspberry Pi möglich. Der Arduino Uno hat keine Möglichkeit, außer durch WLAN erweiterne Shields, sich zu einem Netz zu entwickeln. Da ist das Produkt von Espressif lukrativer.

Nun folgt eine Tabelle (siehe 3.1), die die einzelnen Kriterien grafisch darlegt.

| | Arduino Uno | Raspberry Pi 4 Model B | ESP32 |
|---|---|---|---|
| Leistung | ★ ★ ★ | ★ ★ ★ ★ ★ | ★ ★ ★ |
| Stromverbrauch | ★ ★ ★ | ★ | ★ ★ ★ |
| Energiesparoptionen | ★★ | ★★ | ★ ★ ★ ★ ★ |
| Preis | ★ ★ ★ | ★ | ★ ★ ★ ★ ★ |
| Größe der Boards | ★ ★ ★ | ★★ | ★ ★ ★ ★ ★ |
| Anzahl der Pins | ★ ★ ★★ | ★ ★ ★★ | ★ ★ ★★ |

ein Stern - sehr schlecht, fünf Sterne - sehr gut

Abb. 3.1: Hardware Bewertung

## 3.2 Auswahl der Programmiersprache

Der ESP32 unterstützt standardmäßig die Arduino Mischsprache aus C und C++. Durch die Änderung der Firmware auf NodeMCU ist zudem Lua RTOS möglich. Selbst eine abgewandelte Form von Python, nämlich MicroPython kann auf den ESP32 betrieben werden.

Bewertungskriterien für die Programmiersprache sind:

- gibt es eine große Community, die viele Bibliotheken oder ähnliches selbst zur Verfügung stellt
- und lässt es sich effizient programmieren im Hinblick auf leistungs- und Speicherplatz-Optimierungen.

### Lua RTOS

Lua RTOS (Real-Time Operating System) ist ein Echtzeit-Betriebssystem, welches speziell für eingebettete Systeme entwickelt wurde, mit minimalen Anforderungen an Flash und RAM Speicher. Der Lua Interpreter ist mit der Version 5.4.3 ausgestattet. Damit kommt er der neuesten Version von Lua mit der Version 5.4.0 sehr nahe. Aktuell ist diese Sprache für ESP32, ESP8266 und PIC32MZ Boards verfügbar, kann jedoch auf jedes andere 32-Bit Board portiert werden. Für eine Skriptsprache ist Lua bemerkenswert effizient und schnell. Die Lua Gemeinschaft gehört nicht zu den großen Communitys im Entwicklerbereich. Das lässt sich am PYPL-Index ablesen für die beliebtesten Programmiersprachen im Januar 2021. Dort belegt Lua, mit unter einem Prozent, die hinteren Plätze (vgl. whitecatboard o.D., vgl. PYPL 2021).

### MicroPython

Das Ziel von MicroPython ist es, Python 3 auf Mikrocontrollern und kleinen eingebetteten Systemen implementieren zu können. MicroPython ist sehr ähnlich zum regulären Python. Wenn man Erfahrung mit Python hat, wird man mit MircoPython keine Probleme haben. Der größte Unterschied ist, dass MicroPython designt wurde, um unter eingeschränkten Bedingungen zu funktionieren. Daher kommt es nicht mit allen Standard Bibliotheken, sondern nur mit einer kleinen Untermenge. Braucht man eine, nicht bereits enthaltende, Bibliothek, muss man gegebenenfalls Anpassungen vornehmen, zu der regulären Python 3 Bibliothek. Python gilt im, allgemeinen, als sehr einsteigerfreundlich und ist dennoch sehr effizient. Im PYPL-Index belegt Python, mit über 30 Prozent, den ersten Platz. Das lässt auf eine sehr große Community schließen. Genauso wie bei Lua muss im Vorfeld die Firmware angepasst werden (vgl. Tim 2020, vgl. Rui Santos 2018a, vgl. PYPL 2021).

### C/C++

Im ESP32 ist ein Arduino Core implementiert. Das führt dazu, dass man die gängigen Entwicklungsumgebungen und Programmiersprachenfeatures, wie bei jedem anderen Arduino Projekt, verwenden kann. Bei Arduino wird eine Mischform aus C/C++ verwendet. Durch ihre Hardwarenähe lässt sich viel in der Speicherverwaltung und Laufzeit aus eigener Hand optimieren. Es gehört zu den oberen beliebtesten Sprachen und hat auch dank der Arduino Projekte eine sehr große Bastler-Community (vgl. Espressif o.D.(a), vgl. PYPL 2021).

### Entscheidung: C/C++

Aufgrund der Communitygröße musste die Wahl zwischen MicroPython oder C/C++ fallen. Da Unklarheit bestand welche Bibliotheken noch benötigt werden, und man etwas verunsichert war, wie diese in MicroPython umgesetzt werden müssen. Ebenso die große Arduino Community bestärkte unsere Entscheidung für den Arduino Core und deren Sprache C/C++.

Zum Überblick in einer tabellarischen Form:

| | Lua RTOS | MicroPython | C/C++ |
|---|---|---|---|
| Communitygröße | ★ | ★★★★★ | ★★★★ |
| Effizientes programmieren | ★★ | ★★★ | ★★★★ |

ein Stern - sehr schlecht, fünf Sterne - sehr gut

Abb. 3.2: Programmiersprache Bewertung

## 3.3 Auswahl der Architektur

Neben den Komponenten, und wie diese entwickelt werden, spielt die Architektur ebenso eine sehr große Rolle. Bei der Architektur geht es darum, wie innerhalb des Netzes kommuniziert werden soll. Bewertet werden eine klassische Server/Client Architektur, Publish-Subscribe per MQTT und ESP-Now.

Als Bewertungsgrundlage wurde folgendes definiert:

- ressourcenschonender Datenaustausch, um die Überlastungen im Netz zu vermeiden
- und dynamischer Wachstum des Netztes bzw. Skalierbarkeit.

### Server/Client

Die Teilnehmer im Netz haben unterschiedliche Rollen - einmal Server und einmal Client. Über Protokolle sind die Abläufe und Interaktionen definiert. Zu den bekanntesten Protokollen zählen UDP, TCP HTTP oder FTP. In der Regel bietet ein Server einen Dienst an und dieser wird von einem oder mehrere Clients genutzt. Zu den typischen Server/Client Modellen zählen Anwendungen wie der Zugriff auf einen Webserver oder die Abwicklung des E-Mail-Verkehrs (vgl. Elektronik-Kompendium 2021, vgl. D.-I. ( A. D. Dipl.-Ing. (FH) Stefan Luber 2019).

### MQTT

Message Queue Telemetry Transport oder kurz MQTT ist ein leichtgewichtiges Protokoll, welches sich “als wichtigstes Internet of Things-Standardprotokoll etabliert“ (Raschbichler 2017).

Der zentrale Aspekt von MQTT ist eine ereignisgesteuerte Publish/Subscribe-Architektur. Es gibt einen zentralen Server (Broker). Auf diesen können, per

publish, Daten gesendet werden. Daten empfängt man über die abonnierten Kanäle, die man im Vorfeld subscribed. Es werden sogenannte Topics abonniert, die wie ein Pfad aufgebaut sind. Anhand der Abb. 3.3 kann man den Ablauf sehr gut nachvollziehen.

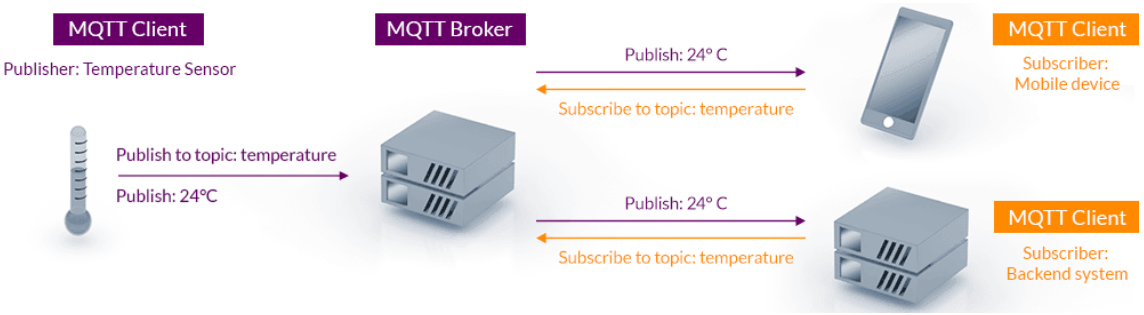


Abb. 3.3: Ablauf MQTT (MQTT o.D.)

Links ist ein Client mit einem Temperatur Sensor. Dieser möchte Daten in dem Topic *temperature* senden. Diesen Vorgang nennt man Publish. Nun kann jedes Endgerät, welches Zugriff auf den Broker und das entsprechende Topic *temperature* abonniert hat, auf die Daten zugreifen und anzeigen lassen. Um zu garantieren das die Daten auch ankommen, gibt es verschiedene Quality of Service (QoS). Bei QoS = 0 werden die Daten genau einmal gesendet, ohne Rücksicht auf eine Bestätigung des Empfängers. Bei der nächsten Qualitätsstufe werden die Daten solange gesendet, bis der Empfänger mindestens einmal eine positive Rückmeldung liefert. Bei der letzten Stufe, der QoS = 2 wird zugesagt, dass eine Nachricht genau einmal gesendet und auch beim Empfänger ankommt. Die Implementierung für einen MQTT Client für den ESP32 ist bereits abgeschlossen und wurde erfolgreich getestet. Bei der Broker-Lösung handelt es sich um eine Beta-Lösung. Zum Beispiel werden die Qualitätsstufen höher 0 nicht unterstützt und er kann momentan aus Speichergründen nur mit maximal acht Clients gleichzeitig kommunizieren (vgl. MQTT o.D., vgl. Raschbichler 2017, vgl. tuanpmt o.D., vgl. martin-ger o.D.).

### ESP-Now

ESP-Now ist ein schnelles und leichtgewichtiges Protokoll, welches von Espressif entwickelt wurde. Es erlaubt mehrere ESP-Module miteinander, ohne WiFi, kommunizieren zu lassen. Ohne Verwendung eines Routers läuft die Kommunikation sehr ähnlich zum energiesparenden 2,4 GHz Protokoll ab, welches sehr

oft in kabellosen Mäusen verwendet wird. Es werden 250 Byte große Pakete versendet. Dazu erlaubt es verschlüsselte und nicht verschlüsselte Kommunikationen zu führen, die für jeden Teilnehmer individuell definiert werden. Auch verschiedene Topologien mit one-way oder two–way Kommunikationen sind möglich. Als Nachteil lässt sich anmerken, dass die Skalierbarkeit nicht gegeben ist. Man muss seinen Kommunikationspartner, über die MAC-Adresse, im Vorfeld festlegen (vgl. Espressif o.D.(c), vgl. Rui Santos 2020b).

### Entscheidung: Server/Client

ESP-Now klingt sehr interessant, wegen der weitreichenden Kommunikationsmöglichkeit ohne WiFi. Die fehlende Skalierbarkeit ist ein nicht zu verachtender Punkt. Würde man einen bereits bestehenden MQTT-Broker verwenden, wäre dies aufgrund der Leichtgewichtigkeit, eine sehr interessante Lösung. Da der Prototyp, ein Gesamtprodukt in Form einer Serverrolle und Clientrolle abgeben soll, ist der Broker auf Basis des ESP32 noch zu unausgereift. So habe man sich letztendlich, für das altbekannte Server/Client-Modell entschieden, aufgrund der zahlreichen unterstützten Protokolle, anbei die beiden anderen Kommunikationsmöglichkeiten, auch sehr interessant sind. Jedoch lässt sich festhalten, ist ein Broker bereits vorhanden, dann wäre diese Lösung der Server/Client-Rolle vorzuziehen. Daher kommt in der Übersicht auch die 5 Sterne Bewertung bei MQTT unter den beiden Gesichtspunkten. Muss der Broker über den ESP32 abgebildet werden, dann würde es für die Skalierbarkeit vier Sterne und drei Sterne für den Datenaustausch geben, aufgrund der fehlenden zwei Quality of Service-Stufen.

| | Server/Client | MQTT | ESP-Now |
|---|---|---|---|
| ressourcenschonender Datenaustausch | ★ ★ ★ | ★ ★ ★ ★ ★ | ★ ★ ★★ |
| Skalierbarkeit | ★ ★ ★ ★ ★ | ★ ★ ★ ★ ★ | ★ |

ein Stern - sehr schlecht, fünf Sterne - sehr gut

Abb. 3.4: Architektur Bewertung

## 3.4 Wege zum Internetzugang

Internet, Near Field Communication (NFC) und Bluetooth - alle haben eins gemeinsam. Sie können Daten übertragen. Das Internet ist noch in zwei Kategorien

aufzuteilen: WLAN und kabelgebundenes LAN. Das wichtigste Argument bei der Wahl des Kanals für die Kommunikation ist der weitreichende Zugriff. NFC oder besser bekannt als Infrarotstrahlung hat, wie der Name vermuten lässt, eine sehr geringe Reichweite von 1 Meter. Bei besonderen Bedingungen sind bis zu 10 Meter möglich. Ähnlich sieht es mit Bluetooth aus, welches je nach Endgerät eine Reichweite zwischen 10 und 100 Metern aufweist. WLAN-Router senden in einem Umkreis von ca. 100 Metern und bei kabelgebunden Netzen, hängt es stark vom Kabel ab. Über Glasfaser hat man eine Reichweite von bis zu 70 Kilometer. Die ESP32 haben keinen Kabelanschluss, so fällt die weitreichendste Möglichkeit flach. Aufgrund dessen das WLAN und LAN-Netze perfekt harmonieren und die Daten in die jeweiligen Netze des nächstbesten Routers reichen, sind Kommunikation über die ganze Welt möglich. Ein weiterer Vorteil ist, dass das Internet sehr viele andere Daten bereithält, die man für sich nutzen kann. Zum Beispiel möchte man Vorhersagen über das Wetter treffen, kann auf eine Wetter-API zugegriffen werden, für zusätzliche Daten.

In diesem Abschnitt soll auf den Unterschied zwischen Station Mode und Access Point eingegangen werden. Dazu wird abgewogen welcher dieser beiden Modi genutzt werden soll und wie die ESP32 Module Zugang zum Internet bekommen. Zur Auswahl stehen: WPS und der WiFiManager.

Bewertet wird nach der Einfachheit des Verbindens mit dem Internet und Besonderheiten die sie zusätzlich anbieten.

### Station Mode oder Access Point

Verbindet sich der ESP32 zu einem bereits bestehendem WiFi Netzwerk, zum Beispiel durch den Router, nennt man diesen Station (siehe Abb. 3.5). Er bekommt vom Router eine IP-Adresse zugewiesen und kann mit allen anderen Geräten innerhalb des Netzwerks Daten austauschen.

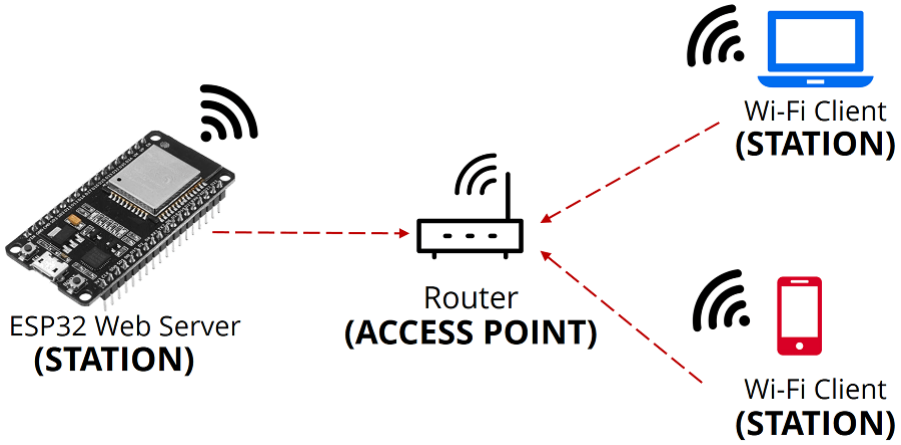


Abb. 3.5: Station Mode (Rui Santos 2018b)

Neben dem Station Mode kann der ESP32 selbst anstelle des Routers in der Mitte stehen (siehe Abb. 3.6). Dafür muss er einen Access Point aufmachen. Da dieser alleine dasteht, ohne Anschluss zu einem verkabelten Netzwerk, nennt man diesen Access Point auch Soft Access Point. Durch diesen erzeugt er ein neues WiFi Netzwerk, mit eigenem Namen (SSID) und bei Bedarf einem Passwort. Somit kann jedes internetfähige Gerät auf das neue lokale Netzwerk zugreifen. In der Dokumentation ist der Zugriff nur für maximal zehn Geräte gleichzeitig gestattet (vgl. Espressif o.D.(f), vgl. Rui Santos 2018b, vgl. lastminuteengineers o.D.).

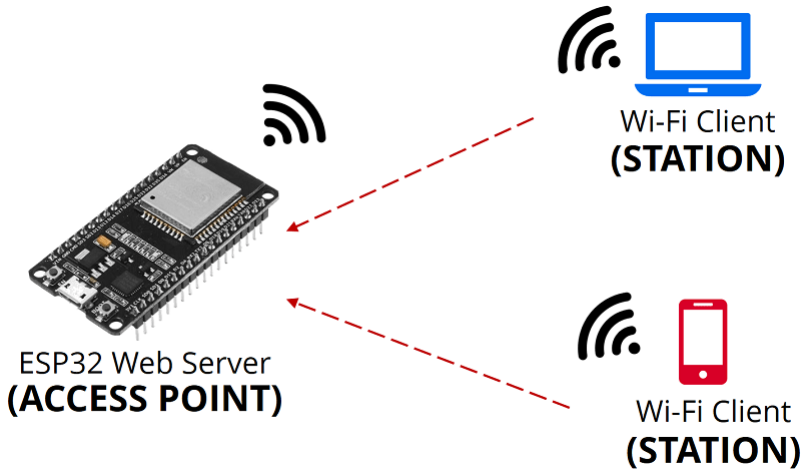


Abb. 3.6: Access Point (Rui Santos 2018b)

### WiFi Protected Setup

WiFi Protected Setup (WPS) ist ein von der Wi-Fi Alliance entwickelter Standard und 2006 veröffentlicht worden. Dieser soll den Verbindungsaufbau zu drahtlosen lokalen Netzen mit Verschlüsselung vereinfachen. Dazu gibt es verschiedene Wege, wie dieser Prozess abgearbeitet werden kann. Die erste ist die Push-Button Configuration. Bei dieser muss zuerst am Router die WPS-Taste betätigt werden und man hat zwei Minuten Zeit am Endgerät die entsprechende Aktion durchzuführen. Dann gibt es den Zugriff über eine Pin-Eingabe. Dort wird vom Router oder vom Endgerät ein Pin generiert, der auf dem anderen Gerät, zur Authentifizierung, eingegeben werden muss. Eine weitere Möglichkeit ist über NFC. Befindet sich das Endgerät in der Nähe des Routers, tauscht es mit ihm die Verbindungsdaten aus. Zu guter Letzt können die Verbindungsdaten per USB-Stick ausgetauscht werden, auf dessen der Zugriffsschlüssel gespeichert ist. Diese Methodik setzt voraus, dass man selbst bereits im Netz registriert ist. Ist dies erfüllt, bietet es eine Browser-Schnittstelle an, um den WPS Prozess in die Gänge zu leiten (vgl. wintotal 2020, vgl. Rusen 2017).

### WiFiManager

Der WiFiManager ist eine Bibliothek, die das Einwählen in drahtlose Netze erleichtern soll. Das funktioniert über das Zusammenspiel mit Access Point und Station Mode. Der Ablauf ist folgender: Startet man den ESP32 versucht es in den Station Mode überzugehen, indem sie die Verbindungsdaten der letzten gespeicherten Access Points ausprobiert. Ist dies nicht erfolgreich, startet er im Access Point Modus und setzt einen Webserver auf. Nun kann man mit jedem internetfähigen Gerät, sich am Access Point anmelden und hat Zugriff auf den Webserver. In der Zwischenzeit scannt der Access Point die Umgebung auf andere Access Points. Diese neu gefunden Access Points, meistens Router, werden nun im Browser angezeigt. Dort kann man sich beim Router, über die Zugangsdaten, authentifizieren. War die Authentifikation erfolgreich, werden die Daten gespeichert und der ESP32 wird neugestartet. Dann beginnt der Prozess von vorne, mit dem einzigen Unterschied, dass er sich über den Station Mode anmelden kann. Möchte man die gespeicherten Daten löschen, muss man software-seitig ein Feld dafür schaffen, indem man die Daten entfernen kann. Man kann auch hardware-seitig die Daten entfernen, indem man das Gerät neu bootet. Dies ist aber nicht ohne weitere Jumper-Kabel möglich. Die Browseransicht lässt sich

beliebig anpassen und beispielhaft um eine MQTT Broker Anmeldung ergänzen (vgl. tzapu o.D., vgl. Rui Santos 2018c).

### Entscheidung: Station Mode und WPS

Die erste Entscheidung fiel auf den Station Mode. Der Access Point bietet sich für den Server, als zentrale Rolle, an. Jedoch ist eine Anforderung, dass man weitreichenden Zugriff hat. Beim Access Point ist man auf die Reichweite von jenem diesem beschränkt. Mit dem Station Mode, hat man bereits eine Verbindung zum Internet und hat somit theoretisch weltweiten Zugriff. Der WiFiManager bieten all das und noch mehr, was der WPS auch kann. Einzig allein, dass man die Daten nur sehr schlecht löschen kann, ist ein großer Minuspunkt. Weiß man im Voraus, dass man das Netz nicht mehr wechseln wird, bietet sich der WiFiManager an. Da der Prototyp auch einen Netzwechsel standhalten können soll, haben wir uns für die WPS-Methodik entschieden. Von diesen vier Methoden, die genannt werden, blieb nur die Push-Button Configuration als machbare Lösung. Dynamisch einen Pin ein- oder auszugeben ist ohne Peripherie nicht möglich. Genauso fehlt eine USB-Schnittstelle. NFC muss erst als Modul angebracht werden, damit diese Möglichkeit überhaupt erst in Frage kommt.

# 4 Umsetzung des Prototypen

Für den Prototypen haben wir uns für das WiFi kit 32 mit OLED Bildschirm von Heltec Automation und den diymore ESP32-CAM entschieden. Das OLED Display soll ein weiterer Faktor im Feedbacksystem spielen und die Kameras dienen als Platzhalter für jede Art von Sensoren oder Modulen, die angebaut werden können. Die Kamera, die verbaut wurde, ist eine OV2640. Die erlaubt es Bilder mit 1600 x 1200 Pixel aufzunehmen oder mit 15 Bilder pro Sekunde zu streamen. Die Auflösung kann nach unten verändert werden, z.B. 800 x 600 oder 1024 x 768. Selbiges gilt auch für den Videostream. Dazu wird von Espressif ein Beispiel-Sketch mitgegeben, indem diese Kamera verwendet wird. Da wird bereits eine Oberfläche angeboten, indem man die Auflösungstufen einstellen kann. Die Werte wie Kontrast und Helligkeit können ebenfalls angepasst werden. Dazu gibt es die Option, die es erlaubt eine Gesichtserkennung durchzuführen. Es erscheint bei einem erkannten Gesicht eine Box, um das jeweilige Gesicht (vgl. Rui Santos 2019c, vgl. Arducam o.D., vgl. Espressif o.D.(b)). Ziel ist es eine Home-Build Kameraanlage zu entwickeln, in der jederzeit weitere Kameras hinzugefügt oder entfernt werden können. Beim Interface soll der Webserver eine entscheidende Rolle übernehmen.

## 4.1 Aufbau des Backend

Der erste Schritt ist das die Boards eine Verbindung zum Internet aufbauen, um mit anderen Teilnehmer zu kommunizieren. Sobald das Board an einer Stromquelle abgeschlossen wird, startet der WPS-Vorgang. Gleichzeitig muss am Router der selbe Vorgang gestartet werden.

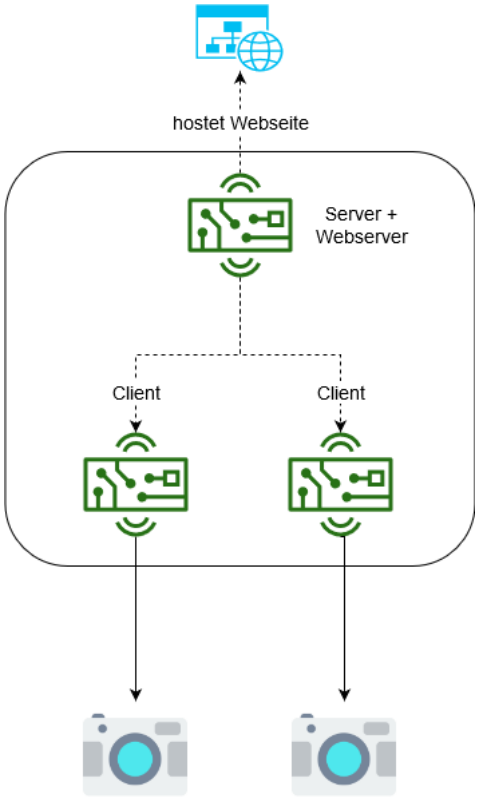


Abb. 4.1: Server/Client Architektur

Die Kommunikationswege für die Kameraanlage sind in Abb. 4.1 zu erkennen. Ein ESP32 dient als Server und stellt als Dienst einen Webservice zur Verfügung. Die Clients, weitere ESP32, tauschen ihre IP-Adressen über UDP/TCP mit dem Server aus und bauen das Netz auf. Der Prozess des Austausches von IP-Adressen läuft folgendermaßen ab. Der Server versendet per UDP-Broadcast eine Kennung, z.B. *"BA2021"*. Ein Client prüft, ob die UDP-Nachricht eine bestimmte Kennung besitzt - stimmt diese mit der erwarteten Kennung (*"BA2021"*) überein, so weiß der Client, dass es sich um eine Nachricht vom Server handelt. Daraufhin speichert der Client die Server-IP als solche ab. Nachdem er die Server-IP hat, sendet der Client seine IP-Adresse per TCP an den Server. Die Server-IP darf öffentlich bekannt sein, da auch über diese auf das Frontend zugegriffen wird. Deswegen wird sie per UDP-Broadcast versendet. UDP ist deutlich schneller als TCP, weil TCP ein verbindungsorientiertes Transportprotokoll ist. Das heißt es muss zuerst eine virtuelle Verbindung zum Gegenüber aufbauen. Dies geschieht über die drei Schritte Verbindungsaufbau, Datenübertragung und Verbindungsabbau. Dadurch kann garantiert werden, dass die übertragene Nachricht sicher beim Gegenüber ankommt. Darauf wird bei UDP verzichtet

und das macht es zu einem verbindungslosen Transportprotokoll. Deshalb ist es vorteilhaft, die Rückmeldung von Client zu Server über TCP zu gestalten, da das Ankommen der Nachricht zum Server garantiert wird. So wird die IP vom Client direkt an den Server geschickt und nicht, wie beim Broadcast, an jeden Netzteilnehmer. Der Broadcast Prozess ist nur eine kleine Zeitperiode aktiv, um das Netz nicht unnötig zu überlasten.

Nachdem alle Kommunikationspartner ihre IP-Adressen ausgetauscht haben, werden die Client-IP Adressen in einem Array, auf dem Server, gespeichert. Möchte man den Server auf einen neuen Standort verfrachten, die Kameras aber nicht - dann hat man ein Problem. Sobald der Server keinen Strom mehr bekommt, verliert er alle Daten, auch die IP-Adressen der Kameras. Man müsste, jede Kamera neu mit dem Server verbinden. Um das zu vermeiden werden die Daten persistent auf den Flash-Speicher gesichert. Ein Flash-Speicher ist ein nicht-flüchtiger Datenträger. Das bedeutet, wenn das Board von einer Energiequelle abgezogen wird, bleiben die Daten dennoch gespeichert. Der RAM ist ein flüchtiger Datenträger. Sobald bei ihm die Energie entfernt wird, sind die Daten verschwunden. Somit eignet sich ein Flash-Speicher, um die letzten Zustände einzelner LEDs oder Sensorinformationen zu speichern. Oder im unseren Fall, die IP-Adressen der Kameras. Eine Limitierung gibt es dennoch: der Flash-Speicher hat eine begrenzte Anzahl von Schreibzyklen. Diese liegt zwischen 100.000 und 1.000.000 Schreiboperationen. Dies ist jedoch zu vernachlässigen, da der Prototyp nur in der Anfangsphase der Entwicklung eine wichtige Rolle spielt.

Beim ESP32 wird mittels der EEPROM Bibliothek auf den Flash-Speicher zugegriffen. Man kann damit bis zu 512 Bytes dauerhaft speichern. Der Speicher ist byteaddressiert und ermöglicht es, jeden Byte einzeln zu addressieren. So besteht die Möglichkeit 512 mal einen Wert zwischen 0 und 255 zu speichern (vgl. Christophe 2020, vgl. Rui Santos 2020a). Die maximale Anzahl der permanent, gespeicherten IP-Adressen beträgt zehn. Diese Beschränkung wurde von den Access Points übernommen. Auf den Flash-Speicher wäre theoretisch Platz für bis zu 32 IP-Adressen. Ebenso kann man externe Speicherkarten, wie zum Beispiel microSD, als permanenten Speicher verwenden. Dazu braucht man ein extra Modul, welches SD-Karten lesen und schreiben kann. Bei den ESP32-CAM ist diese von Haus aus mit dabei (vgl. Rui Santos 2019a).

Der genaue Ablauf wird im Kapitel 4.3 nochmal aufgezeigt.

## 4.2 Gestaltung des Frontend

Die Aufgabe des Interface ist es die Aktivität im Netz darzustellen und die Kommunikation zwischen den Teilnehmern zu steuern. Das wird über das Frontend, einer klassischen Webseite, abgebildet. Man erhält Zugriff über die IP-Adresse des Servers. Das erlaubt es, mit jedem internetfähigem Gerät, auf den Server zuzugreifen und den Inhalt zu beeinflussen.

Die Webseite wird klassisch mit HTML, CSS und JavaScript programmiert. Aufgrund der Speichergröße der Boards wird nur das Nötigste dargestellt. Des Weiteren wird es Responsive entwickelt, so dass die Webseite auf jedem Endgerät optimal angezeigt werden kann. Die Kamera-Inhalte werden dynamisch per JavaScript über die Manipulation des HTML-DOMs erzeugt, da erst zur Laufzeit bekannt ist, wie viele Kameras aktuell verbunden sind.

Der Code wird in einem String gepackt. Im Gegensatz zu Webprojekten - bei denen HTML, CSS und JavaScript Code in drei Dateien aufgeteilt werden - sollte dort alles in einer Datei (*index.h (Header-Datei)*) entwickelt werden. Die Entwicklung wird dadurch nicht komplizierter, da die Webseite nur das wichtigste darstellt, aufgrund des geringen Speicherplatzes. Weiterhin wird der String, mit dem Code der Webseite, mit dem Schlüsselwort *PROGMEM* versehen. Das führt dazu, dass der String nicht im SRAM, sondern gleich im Flash-Speicher abgelegt wird. Aufgrund der merklichen Größe von Webseiten, würde er schnell den SRAM blockieren. Da weitere Berechnungen stattfinden, würde das den SRAM überlasten und die Laufzeit erheblich einschränken oder gar, durch einen Absturz, stilllegen. (vgl. Arduino o.D.(b)).

Als erste Design Idee wurde mit einem Add und einem Remove Button geplant (siehe Abb. 4.2). Der Add Button sollte die Clients mit dem Server verbinden und den Kamera-Inhalt darstellen. Der Remove Button wird das entsprechende Clientbild wieder entfernen. Die Kamera soll beeinflussbar sein, indem Einstellungen wie Helligkeit oder Kontrast verändert werden können.

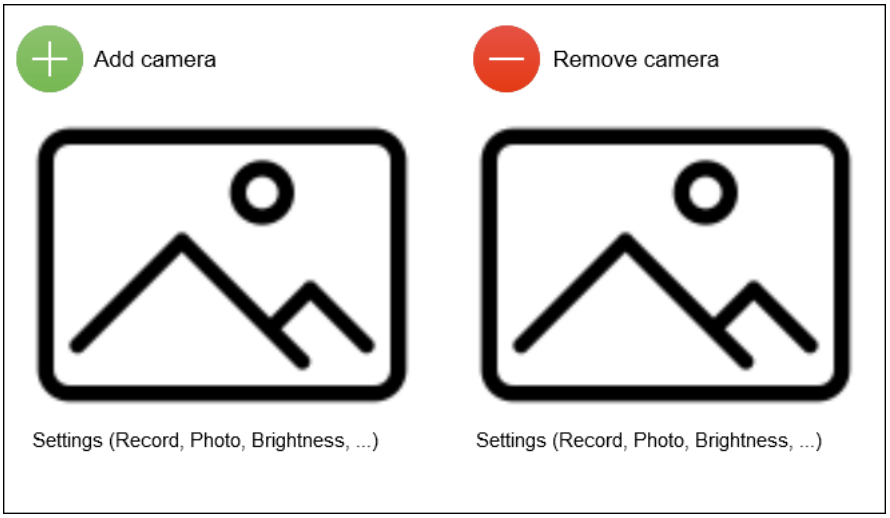


Abb. 4.2: Erstes Webseiten Design

Möchte man Funktionen aus dem C/C++ Part aufrufen, geht das nur über die Änderung der URI, genauer gesagt, dem Browser-Pfad. Durch die Änderung kann der Webserver Funktionen aus der Mischsprache aufrufen. Das heißt bei Klick auf den Add Button, wird an der URI der Zusatz */add/* hinzugefügt. Bei Remove dasselbe nur mit */remove/*.

Die anfängliche Idee mit nur einem Remove Button wurde letztendlich verworfen und unter jeder Kamera ein Remove Button hinzugefügt. Dazu wird die URI erweitert um den Index der Kamera Position. So hat die erste Kamera, stets den Pfadzusatz */remove/0/*. Das Ganze wurde praktiziert, um die Benutzerfreundlichkeit zu steigern. Es ist ersichtlicher bei einem Remove Button unter der Kamera, dass genau diese Kamera, bei drücken auf dem Remove Button, entfernt wird. Beim ersten Modell hätte man noch ein Textfeld benötigt, das die Zahl vom Kunden als Input entgegennimmt. Ohne weitere Erklärung, ist es für einen Nicht-Informatiker, nicht logisch, warum die erste Kamera über den Wert 0 indexiert wird.

Als weiteres Feedbacksignal wird der Add Button, je nach Zustand, angepasst. Die Zustände sind: normal, positiv und negativ. Ist der Add Prozess erfolgreich, konnte eine Kamera hinzugefügt werden, wird der Rand des Buttons grün. War dagegen die Suche nicht erfolgreich wird der Rand rot. Im Normalmodus nimmt er die Farbe des Buttons an (siehe Abb. 4.3). Zweck dieser Aktion ist es, dass der Kunde eine nachvollziehbare Reaktion angezeigt bekommt und so Fehlerquellen ausschließen kann. So eine Fehlerquelle könnte sein, dass der Client noch keine Verbindung zum Internet aufgebaut hat.

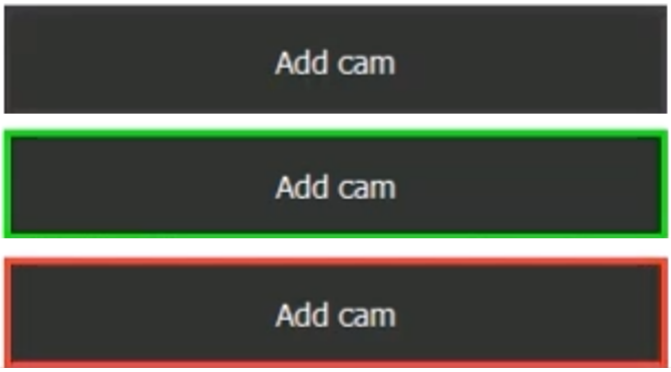


Abb. 4.3: verschiedene Feedbacksignale des Add Button

## Steigerung der Benutzerfreundlichkeit

Die Funktionalität des Webservers haben wir bereits genutzt, um die Benutzerfreundlichkeit zu steigern. Mit ihm haben wir ein Interface und können jederzeit das Netz betrachten und steuern. Das Interface selbst ist ebenso benutzerfreundlicher geworden, indem der Add Prozess ein visuelles Feedback bekommen hat. Eine Sache fehlt noch. Wir haben keinerlei Information, ob sich die Hardwareboards erfolgreich mit dem Internet verbunden haben. Bei der Hardware-Auswahl habe man immer darauf geachtet, ob eine LED dabei ist. Diese werden wir nun als weiteres visuelles Feedbacksignal nutzen. Sobald der ESP32 mit dem Internet verbunden ist, wird die eingebaute LED für wenige Sekunden blinken. Für eines der ESP32 Module haben wir uns eine OLED-Bildschirm Variante entschieden. Dieser Bildschirm eignet sich, um Informationen anzuzeigen. Zum Beispiel könnte ein WiFi-Bild aufleuchten, um zu signalisieren, dass die Verbindung zum Internet erfolgreich war. Oder um die IP-Adresse des Servers (siehe Abb. 4.4). Denn man bekommt erst eine IP-Adresse zugewiesen, wenn die Verbindung zum Internet steht. Dies hat wiederum den Vorteil, das man die Server IP-Adresse immer am Server ablesen kann und dadurch Zugriff auf die Webseite bekommt.


Abb. 4.4: Feedback des OLED-Bildschirm

## 4.3 Ablauf und Funktionsweise

Zuallererst müssen die Boards eine Verbindung mit dem Internet eingehen. Dies geschieht in diesem Fall über WPS. Nachdem sie verbunden sind, blinkt die integrierte LED am Client für ein paar Sekunden. Auf dem OLED Bildschirm des Servers wird die Server IP-Adresse angezeigt. Der Webserver hostet das Interface, eine Webseite. Auf die Webseite kommt man, indem man die Server IP-Adresse im Browser eingibt. Mit dieser kann das Netz beobachtet und gesteuert werden. Zur Steuerung gehört einmal das Hinzufügen eines neuen Clients, welches über den Add Button gestartet werden kann. Dieser Prozess startet den UDP-Broadcast und den Austausch der IP-Adressen zwischen Client und Server über TCP. Die Webseite stellt dynamisch, je nach Anzahl der verbundenen Kameras, Inhaltsboxen für deren Inhalt zur Verfügung. Dazu benötigt die Server-seite nur die IP-Adresse der Clients. Der Videostream wird direkt von der client-seite auf dem Frontend des Servers dargestellt und die bereits existierende Oberfläche angezeigt. Beim Löschen einer Kamera wird sie aus der Liste vom Server entfernt und auf diese kann erstmal nicht mehr zugegriffen werden. Für eine erneute Verbindung zu einem Server muss sie über den RST-Pin zurückgesetzt oder die Energiequelle abgenommen werden. Wird der Server von der Energiequelle abgekoppelt, speichert er die IP-Adressen der Clients und zeigt beim Start wieder alle verbundenen Kameras an.

Zu den Besonderheiten: der WPS Vorgang, aller Boards, kann ein wenig Zeit

beanspruchen. Da, wenn ein Fehler auftritt, ca. zwei Minuten gewartet werden muss, um erneut zu starten. Ein weiteres Feature ist, dass man mit einem Klick auf den Add Button mehrere Kameras gleichzeitig hinzufügen kann. Getestet wurde dies mit vier Kameras gleichzeitig. Aufgrund dessen, dass nur vier Kameras zur Verfügung standen, konnte keine größere Anzahl getestet werden.

## 4.4 Bewertung der Anforderungen

Der Prototyp wird auf die Anforderungen aus Kapitel 2 bewertet. Die erste Eigenschaft die gefordert wurde, war das der Prototyp ein skalierbares Netz aufbauen kann. Insgesamt konnte ein Netz mit einem Server und vier Clients aufgebaut werden. Es konnten mehrere Clients gleichzeitig hinzugefügt werden. Auch aus dem Netz löschen funktioniert. Einzig die Serverleistung limitiert die maximale Anzahl der Teilnehmer auf zehn ein. Somit ist der Mindestwert erreicht. Es könnten auch mehr Geräte angebunden werden, da der Mindestwert vom Access Point übernommen wurde. Jedoch fehlen praktische Ergebnisse, aufgrund der geringen Anzahl an ESP32, die zum Testen vorlagen. Die Modularität zeichnet sich dadurch aus, dass man verschiedene Sensoren oder anderweitige Module anbringen kann. Es wurde in dieser Arbeit lediglich die Kamera getestet. Im Internet gibt es viele Tutorials, die alle möglichen Sensoren anbinden. Bei Verwendung andere Sensoren, müsste man die Webseite umschreiben und die Funktionen des Sensors implementieren. Also lässt sich festhalten, dass der Prototyp sehr universell einsetzbar ist. Der Stromverbrauch, wurde mit einem Multimeter gemessen. Im Betriebszustand verbraucht der ESP32 ungefähr 100 mA. Die Blockbatterie wäre nach 12 Stunden entladen. Das ist ein schlechter Wert. Dies liegt aber auch am nicht optimalen Use-Case der abgebildet wurde. Eine Kamera muss ständig mit Energie versorgt werden. Dazu waren die Boards auch mit dem Internet angeschlossen, was nochmals den Stromverbrauch in die Höhe schraubt. Hätte man smarte Mülleimer getestet, die alle paar Stunden einen Messwert zurückliefern und den Rest der Zeit schlafen, hätte man eine deutlich bessere Energiebilanz aufweisen können. Das zeigt das die Kameras besser dauerhaft mit Strom versorgt werden. Die Bedienung ist sehr intuitiv, aber nicht komplett selbsterklärend. Eine kleine Bedienungsanleitung wäre hilfreich. So ist zum Beispiel nicht klar, dass bei Zuführen von Energie, die ESP32 sofort versuchen eine Verbindung über WPS aufzubauen. Hat man jedoch diese Hürde überwunden, und ist bei der Webseite gelandet, so ist diese sehr selbs-

terklärend und verständlich aufgebaut. Auch die Leuchtsignale der LED und OLED Bildschirm bei Internetzugang oder das persistente Speichern der Daten auf den Server hinterlassen einen positiven Eindruck. Die Zugriffsreichweite ist auf den Router begrenzt. Diese erlauben den Zugriff in der Regel in einem Umkreis von ca. 100m. Dies ist für den Prototypen ausreichend. Jedoch kann man mit dem ngrok Tunnel Service auf schnellen Wegen einen weltweitreichenden Webserver aufbauen. Da dies nicht gestestet wurde kann dazu keine Auskunft gegeben werden (vgl. microcontrollerslab 2019).

Das Ganze nochmal tabellarisch aufgezeigt:

| | Home-Build Kameraanlage |
|---|---|
| Skalierbarkeit | ★ ★ ★★ |
| Modularität | ★ ★ ★★ |
| Stromverbrauch | ★ |
| Benutzerfreundlichkeit | ★ ★ ★★ |
| weitreichender Zugriff | ★ ★ ★★ |

ein Stern - sehr schlecht, fünf Sterne - sehr gut

Abb. 4.5: Prototyp Bewertung

# 5 Fazit und Ausblick

Ziel dieser Arbeit war es einen kostengünstigen und universellen Prototypen zu entwickeln. Dazu wurden die Anforderungen definiert und verschiedene Hardwareplattformen auf diese Aspekte untersucht und ausgewählt. Sukzessiv bewertete man für die ausgewählte Hardwareplattform ESP32 die bestmögliche Programmiersprache, Architektur und Internetoption. Danach wurde der Prototyp in die Tat umgesetzt. Eingegangen wurde auf die Entwickung des Backend, des Frontend und die implemtierten Features im Bereich Benutzerfreundlichkeit. Das Ergebnis ist sehr vielversprechend. Der Prototyp bestätigt fast alle Anforderungen, die anfangs gestellt wurden. Es wurde ein Produkt kreiert, welches die Anforderungsanalyse, auf einem kostengünstigen und umweltschonenden Weg unterstützen kann. Eine zusätzliche Erkenntnis der Arbeit war es, dass man seinen Use-Case genau kennen sollte. Das Effizienz/Flexibilitäts-Konflikt tritt hier zum Vorschein. Möchte man maximale Effizienz, schränkt das die Flexibilität stark ein. Ist zum Beispiel die maximale Anzahl der Netzteilnehmer bekannt, so eignet sich ESP-Now mehr, da auf den WiFi-Dienst verzichtet werden kann. Das spart enorm an Energieleistung. Der Prototyp dagegen ist deutlicher flexibler, damit er ein dynamisches Netz aufbauen kann. Das steigert leider den Energieverbrauch. Solche Erkenntnisse können in der Praxis mit diesem Prototypen gewonnen werden.

Als Ausblick, um den Prototypen auszubauen, könnte man es um den ngrok Dienst erweitern. Desweiteren lässt sich die Home-Build Kameraanlage mit weiteren Modulen (z.B. Lichtschranke) zu einer Home-Build Sicherheitsanlage ergänzen. Auch andere Szenarien können abgebildet werden. Die in der Einleitung erwähnten smarten Mülleimer oder eine Bewässerungsanlage für viele verteilte Pflanzenorte wären die ersten Ideen. Für größere Projekte ist der ESP32 als Server zu schwach, da wäre ein Umrüsten auf den Raspberry Pi die bessere Wahl. Die Entwicklung von interessanten Projekten wie den WiFiManager, den ESP32 als MQTT-Broker oder MicroPython sollte weiter verfolgt und unterstützt werden. Mit diesem Prototypen ist es sehr einfach möglich, nachhaltige

Prozesse zu entwickeln. Egal ob in der Industrie, oder als Hobbybastler - seine Verwendung findet er in vielfältigen Domänen.

# Literaturverzeichnis

Arducam (o.D.). *OV2640 – Specs, Datasheets, Cameras, Features, Alternatives*. URL: https://www.arducam.com/ov2640/. Abgerufen am 19.02.2021.

Arduino (o.D.[a]). *Arduino Uno Rev3*. URL: https://store.arduino.cc/arduino-uno-rev3. Abgerufen am 25.01.2021.

– (o.D.[b]). *PROGMEM*. URL: https://www.arduino.cc/reference/de/language/variables/utilities/progmem/. Abgerufen am 03.02.2021.

Carolo, Lucas (Dez. 2020). *MCU or SBC?* URL: https://all3dp.com/2/arduino-vs-raspberry-pi/. Abgerufen am 25.01.2021.

Christophe (11. Sep. 2020). *ESP8266. How to read, write, erase the EEPROM. Calculate the space needed in bytes*. URL: https://diyprojects.io/esp8266-how-read-write-erase-the-eeprom-calculate-space-needed/. Abgerufen am 03.02.2021.

David, Christopher (2019). *Guide to reduce the Arduino Power Consumption*. URL: https://diyi0t.com/arduino-reduce-power-consumption/. Abgerufen am 25.01.2021.

Dipl. Betriebswirt Otto Geißler, Ulrike Ostler (13. März 2018). *Was ist ein System-on-a-Chip (SoC)?* URL: https://www.datacenter-insider.de/was-ist-ein-system-on-a-chip-soc-a-693472/. Abgerufen am 26.01.2021.

Dipl.-Ing. (FH) Stefan Luber, Dipl.-Ing. (FH) Andreas Donner (19. Juni 2019). *Was ist das Client-Server-Modell?* URL: https://www.ip-insider.de/was-ist-das-client-server-modell-a-940627/. Abgerufen am 28.01.2021.

Dipl.-Ing. (FH) Stefan Luber, Nico Litzel (1. Sep. 2016). *Was ist eine Smart City?* URL: https://www.bigdata-insider.de/was-ist-eine-smart-city-a-599409/. Abgerufen am 10.12.2020.

Eames, Alex (25. Juni 2019). *How much power does the Pi4B use? Power Measurements*. URL: https://raspi.tv/2019/how-much-power-does-the-pi4b-use-power-measurements. Abgerufen am 26.01.2021.

Elektronik-Kompendium (28. Jan. 2021). *Client-Server-Architektur*. URL: https://www.elektronik-kompendium.de/sites/net/2101151.htm. Abgerufen am 28.01.2021.

Espressif (o.D.[a]). *Arduino core for the ESP32*. URL: https://github.com/espressif/arduino-esp32. Abgerufen am 27.01.2021.

– (o.D.[b]). *CameraWebServer*. URL: https://github.com/espressif/arduino-esp32/tree/master/libraries/ESP32/examples/Camera/CameraWebServer. Abgerufen am 19.02.2021.

– (o.D.[c]). *ESP-NOW*. URL: https://www.espressif.com/en/products/software/esp-now/overview. Abgerufen am 28.01.2021.

– (o.D.[d]). *ESP32 Series of Modules*. URL: https://www.espressif.com/en/products/modules/esp32. Abgerufen am 26.01.2021.

– (o.D.[e]). *SoCs*. URL: https://www.espressif.com/en/products/socs. Abgerufen am 03.02.2021.

– (o.D.[f]). *Wi-Fi*. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html#_CPPv416wifi_ap_config_t. Abgerufen am 29.01.2021.

Foundation, Raspberry Pi (o.D.). *Raspberry Pi 4 Tech Specs*. URL: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/. Abgerufen am 26.01.2021.

Geerling, Jeff (2019). *Power Consumption Benchmarks*. URL: https://www.pidramble.com/wiki/benchmarks/power-consumption. Abgerufen am 26.01.2021.

Hannes Heusel, Heiko Jochum (29. Juli 2020). *Die GPIO-Schnittstelle*. URL: https://www.inf-schule.de/vernetzung/raspi/grundlagen/gpio. Abgerufen am 26.01.2021.

heise-online (26. Jan. 2021). *Raspberry Pi – Mini-Computer auf ARM-Basis*. URL: https://www.heise.de/thema/Raspberry-Pi. Abgerufen am 26.01.2021.

Helmut (11. Juli 2018). *ESP32 mit Batteriebetrieb*. URL: https://arduino-hannover.de/2018/07/11/esp32-mit-batteriebetrieb/. Abgerufen am 26.01.2021.

Kobbe, Tobias (28. Sep. 2017). *Stromverbrauch Arduino and Wemos Boards*. URL: https://arduino-projekte.info/stromverbrauch-arduino-wemos-boards/. Abgerufen am 25.01.2021.

Lang, Mirco (22. Mai 2020a). *ESP32: Was ist das? Was kann das?* URL: https://www.heise.de/tipps-tricks/ESP32-Was-ist-das-Was-kann-das-4471527.html. Abgerufen am 26.01.2021.

Lang, Mirco (13. Mai 2020b). *Was ist Arduino?* URL: https://www.heise.de/tipps-tricks/Was-ist-Arduino-4035461.html. Abgerufen am 25.01.2021.

lastminuteengineers (o.D.). *Insight Into ESP32 Sleep Modes and Their Power Consumption*. URL: https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/. Abgerufen am 26.01.2021.

martin-ger (o.D.). *uMQTTBroker*. URL: https://github.com/martin-ger/uMQTTBroker. Abgerufen am 25.02.2021.

Mazzari, Vanessa (26. Sep. 2016). *Die Qual der Wahl beim Arduino Board*. URL: https://blog.generationrobots.com/de/auswahl-arduino-board/. Abgerufen am 25.01.2021.

microcontrollerslab (2019). *Accessing ESP32 web server from anywhere in the world ( ESP8266 compatible)*. URL: https://microcontrollerslab.com/accessing-esp32-web-server-anywhere-world-esp8266/. Abgerufen am 07.02.2021.

MQTT (o.D.). *Why MQTT?* URL: https://mqtt.org/. Abgerufen am 28.01.2021.

PYPL (Jan. 2021). *Die beliebtesten Programmiersprachen weltweit laut PYPL-Index im Januar 2021*. URL: https://de.statista.com/statistik/daten/studie/678732/umfrage/beliebteste-programmiersprachen-weltweit-laut-pypl-index/. Abgerufen am 27.01.2021.

Ramos, Igor (2. Sep. 2013). *Arduino Power Consumption Normal and Sleep*. URL: https://www.gadgetmakersblog.com/arduino-power-consumption/. Abgerufen am 25.01.2021.

Raschbichler, Florian (13. Juni 2017). *MQTT - Leitfaden zum Protokoll für das Internet der Dinge*. URL: https://www.informatik-aktuell.de/betrieb/netzwerke/mqtt-leitfaden-zum-protokoll-fuer-das-internet-der-dinge.html. Abgerufen am 28.01.2021.

Rui Santos, Sara Santos (2018a). *Getting Started with MicroPython on ESP32 and ESP8266*. URL: https://randomnerdtutorials.com/getting-started-micropython-esp32-esp8266/. Abgerufen am 27.01.2021.

– (2018b). *How to Set an ESP32 Access Point (AP) for Web Server*. URL: https://randomnerdtutorials.com/esp32-access-point-ap-web-server/. Abgerufen am 29.01.2021.

– (2018c). *WiFiManager with ESP8266 – Autoconnect, Custom Parameter and Manage your SSID and Password*. URL: https://randomnerdtutorials.com/wifimanager-with-esp8266-autoconnect-custom-parameter-and-manage-your-ssid-and-password/. Abgerufen am 29.01.2021.

Rui Santos, Sara Santos (2019a). *ESP32 Data Logging Temperature to MicroSD Card*. URL: https://randomnerdtutorials.com/esp32-data-logging-temperature-to-microsd-card/. Abgerufen am 03.02.2021.

– (2019b). *ESP32 Deep Sleep with Arduino IDE and Wake Up Sources*. URL: https://randomnerdtutorials.com/esp32-deep-sleep-arduino-ide-wake-up-sources/. Abgerufen am 26.01.2021.

– (2019c). *ESP32-CAM Video Streaming and Face Recognition with Arduino IDE*. URL: https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/. Abgerufen am 19.02.2021.

– (2020a). *ESP32 Flash Memory – Store Permanent Data (Write and Read)*. URL: https://randomnerdtutorials.com/esp32-data-logging-temperature-to-microsd-card/. Abgerufen am 03.02.2021.

– (2020b). *Getting Started with ESP-NOW (ESP8266 NodeMCU with Arduino IDE)*. URL: https://randomnerdtutorials.com/esp-now-esp8266-nodemcu-arduino-ide/. Abgerufen am 28.01.2021.

Rusen, Ciprian Adrian (18. Sep. 2017). *Simple questions: What is WPS (Wi-Fi Protected Setup) and how does it work?* URL: https://www.digitalcitizen.life/simple-questions-what-wps-wi-fi-protected-setup/. Abgerufen am 29.01.2021.

Rush, Chris (26. März 2018). *How to save Power on your Raspberry Pi*. URL: https://learn.pi-supply.com/make/how-to-save-power-on-your-raspberry-pi/. Abgerufen am 26.01.2021.

Tim (7. Aug. 2020). *Python and MicroPython - Compare and Access*. URL: https://core-electronics.com.au/tutorials/Python_and_MicroPython.html. Abgerufen am 27.01.2021.

tuanpmt (o.D.). *esp_mqtt*. URL: https://github.com/tuanpmt/esp_mqtt. Abgerufen am 25.02.2021.

tzapu (o.D.). *WiFiManager*. URL: https://github.com/tzapu/WiFiManager. Abgerufen am 29.01.2021.

whitecatboard (o.D.). *Lua-RTOS-ESP32*. URL: https://github.com/whitecatboard/Lua-RTOS-ESP32. Abgerufen am 27.01.2021.

wintotal (5. Feb. 2020). *Was ist WPS? Funktion einfach erklärt*. URL: https://www.wintotal.de/was-ist-wps/#3_Moegliche_Probleme_bei_der_Nutzung_von_Wi-Fi_Protected_Setup. Abgerufen am 29.01.2021.

# Abbildungsverzeichnis