Mannheim, den 18.05.2026

Werner-von-Siemens-Schule Mannheim
Schuljahr: 2025/26
Klasse: E2FS1
Fach: LBT3
Lehrkraft: Hr. Baumgärtner

Gruppe 1:
Lars Fetsch
Nico Huchatz
Philipp König
Daniel Funk

IoT-Projekt
Raum-/Zugangskontrolle

Dokumentation

Inhaltsverzeichnis

Dokumentation

1. Einführung

Im Rahmen unseres Projektes haben wir ein IoT-basiertes Zugangskontrollsystem entwickelt. Ziel war die Umsetzung einer sicheren und modularen Lösung zur Zugangskontrolle mithilfe verschiedener Technologien, wie zum Beispiel RFID, ESPs und MQTT sowie der Integration einer zentralen Backend-Infrastruktur. Des Weiteren wurde eine KI-gestützte, kamerabasierende Personenerkennung integriert, um Bewegungen im Eingangsbereich erfassen zu können.

1.1 Projektbeschreibung

Um Zugang zu einem Raum zu erhalten, muss ein Anwender zunächst berechtigt sein. Jeder autorisierte Benutzer erhält hierfür einen RFID-Chip mit einer eindeutigen UID. Zur Überprüfung der Berechtigung wird der RFID-Chip an einen RFID-Sensor gehalten, welcher über ein Steckbrett mit einem ESP32-Mikrocontroller verbunden ist. Der ESP32 erfasst die UID des Chips und sendet die Daten mittels MQTT an einen HiveMQ-Broker.

Die weitere Verarbeitung der Daten erfolgt innerhalb eines Flows in Node-RED. Alle für den Zugang berechtigten UIDs werden in einer PostgreSQL-Datenbank gespeichert. Node-RED sowie die Datenbank laufen auf einem zentralen, abgesicherten Linux-Backend innerhalb von Docker-Containern.

Nach dem Empfang der UID überprüft Node-RED, ob die übermittelte UID in der Datenbank vorhanden und damit autorisiert ist. Ist dies der Fall, wird über MQTT ein Signal an einen zweiten ESP32 gesendet, an welchem ein LED-Band angeschlossen ist. Bei erfolgreicher Authentifizierung leuchten die LEDs grün auf. Ist die UID nicht in der Datenbank vorhanden, wird der Zugriff verweigert und das LED-Band signalisiert dies durch eine rote Beleuchtung.

Zusätzlich wurde eine kamerabasierte Personenerkennung integriert. Hierbei erkennt eine YOLO-Kamera Personen, welche eine zuvor definierte Linie überschreiten. Die erkannten Bewegungen werden ebenfalls über MQTT verarbeitet. Mithilfe eines Zählmechanismus wird die aktuelle Anzahl der Personen innerhalb des Raumes ermittelt und über das LED-Band visualisiert. Dadurch kann nachvollzogen werden, wie viele Personen sich aktuell im Raum befinden.

1.2 Projektziel

Ziel des Projekts war die Entwicklung eines modularen und sicheren Zugangskontrollsystems auf Basis moderner IoT-Technologien. Dabei sollte eine Kombination aus Hardware- und Softwarekomponenten entstehen, welche eine zuverlässige Authentifizierung von Benutzern ermöglicht und gleichzeitig flexibel erweitert werden kann.

Ein zentraler Bestandteil des Projekts war die Verarbeitung von RFID-basierten Zugangsberechtigungen. Autorisierte Benutzer sollten anhand ihrer RFID-UID erkannt und Zugriffe

entsprechend freigegeben oder verweigert werden. Zusätzlich sollte das System in der Lage sein, Personenbewegungen innerhalb eines Raumes zu erfassen und visuell darzustellen.

Darüber hinaus bestand das Ziel auch darin, verschiedene Technologien aus den Bereichen Netzwerktechnik, IT-Sicherheit, Datenbanken, Containerisierung und IoT-Kommunikation praktisch miteinander zu kombinieren. Die einzelnen Komponenten sollten dabei über MQTT miteinander kommunizieren und zentral über ein abgesichertes Backend verwaltet werden.

Neben der technischen Umsetzung, stand ebenfalls die praktische Teamarbeit samt organisatorischer und administrativer Komponente im Fokus unserer Projektarbeit. Hierzu gehörten die gemeinsame Planung, die Aufgabenverteilung sowie der Austausch innerhalb des Teams sowie die schrittweise Integration der einzelnen Systemkomponenten hin zu einem funktionierenden Gesamtergebnis.

1.3 Projektumfeld

Das Projekt wurde grundsätzlich im Rahmen eines schulischen Teamprojekts umgesetzt. Ziel war die praktische Anwendung verschiedener Technologien aus den Bereichen IoT, Netzwerktechnik, Datenbanken und IT-Sicherheit innerhalb eines gemeinsamen Gesamtsystems. Für die Bearbeitungszeit diente der schulische Anteil als solide Basis, weitere und vor allem kleinteilige und auch konzentrierte Arbeiten mussten jedoch aufgrund der Projektgröße und des Anspruchs zwangsläufig außerhalb der Unterrichtszeiten (z.B. im Betrieb wenn möglich, zu Hause, teils auch als mobiles Arbeiten) durchgeführt werden.

Die Umsetzung erfolgte arbeitsteilig innerhalb unseres vierköpfigen Teams (die Aufgabenverteilung wird später konkret aufgeschlüsselt). Dabei wurden die unterschiedliche Aufgabenbereiche wie die Einrichtung der Backend-Infrastruktur, die Entwicklung der Node-RED-Flows, die MQTT-Kommunikation, die Einrichtung der ESPs, die RFID-Anbindung sowie die kamerabasierte Personenerkennung aufgeteilt.

Als Hardwarekomponenten kamen unter anderem ESP32-Mikrocontroller, RFID-Sensoren, ein LED-Band sowie eine Kamera mit YOLO-basierter Personenerkennung zum Einsatz. Die zentrale Server-Infrastruktur wurde auf einem eigenen Linux-Server umgesetzt und mithilfe von Docker-Containern betrieben.

Für die Kommunikation zwischen den einzelnen Komponenten wurde ein MQTT-Broker von HiveMQ verwendet. Die Verwaltung der Zugangsberechtigungen erfolgte über eine PostgreSQL-Datenbank auf dem Backend. Zusätzlich wurde besonderer Wert auf die Absicherung der Infrastruktur durch HTTPS, Firewall-Regeln und abgesicherte Benutzerzugänge gelegt.

2. Projektplanung

Die Umsetzung des Projekts erforderte bereits im Vorfeld eine strukturierte Planung der technischen Komponenten sowie eine technische, organisatorische und administrative Zusammenarbeit innerhalb des Teams. Ziel war die Entwicklung eines modularen und erweiterbaren Systems, bei

welchem sowohl die einzelnen Hardwarekomponenten als auch die Backend-Infrastruktur zuverlässig miteinander kommunizieren können.

Neben den funktionalen Anforderungen spielte insbesondere die Auswahl geeigneter Technologien eine wichtige Rolle. Dabei mussten sowohl Aspekte der Sicherheit als auch der Erweiterbarkeit und einfachen Verwaltung berücksichtigt werden.

Im Projektverlauf gab es auch immer wieder Anpassungen, Verbesserungen und Neuplanungen, was eine entsprechende Organisation und Kommunikation nötig machte.

2.1 Anforderungen

Für die Umsetzung des Projekts wurden zunächst verschiedene funktionale sowie nicht-funktionale Anforderungen definiert. Die funktionalen Anforderungen beschreiben die konkreten Aufgaben und Funktionen des Systems. Dazu gehörten unter anderem die Erkennung von RFID-Chips, die Überprüfung von Zugangsberechtigungen sowie die visuelle Darstellung des Zugriffsstatus über ein LED-Band. Zusätzlich sollte das System Personenbewegungen innerhalb eines Raumes erkennen und verarbeiten können. Die Kommunikation zwischen den einzelnen Komponenten sollte dabei über MQTT erfolgen.

Neben den funktionalen Anforderungen wurden ebenfalls verschiedene nicht-funktionale Anforderungen berücksichtigt. Hierzu zählten insbesondere Aspekte der Sicherheit, Zuverlässigkeit und Erweiterbarkeit des Systems. Die Kommunikation der Komponenten sollte verschlüsselt erfolgen und der Zugriff auf zentrale Dienste abgesichert werden. Darüber hinaus sollte die Systemarchitektur modular aufgebaut sein, sodass einzelne Komponenten unabhängig voneinander erweitert oder ausgetauscht werden können. Zusätzlich wurde auf eine persistente Datenspeicherung geachtet, um Konfigurations- und Zugangsdaten dauerhaft sichern zu können.

2.2 Auswahl der Technologien

Für das Projekt wurden verschiedene Technologien ausgewählt, welche die Anforderungen des Systems möglichst effizient erfüllen sollten. Dabei wurde insbesondere auf Erweiterbarkeit, einfache Verwaltung sowie eine zuverlässige Kommunikation zwischen den einzelnen Komponenten geachtet.

Zur Containerisierung der zentralen Backend-Dienste wurde Docker eingesetzt. Dadurch konnten einzelne Dienste wie Node-RED und PostgreSQL voneinander getrennt betrieben werden. Gleichzeitig ermöglicht Docker eine einfache Verwaltung, Wiederherstellung und Erweiterung der Systemkomponenten.

Für die Verarbeitung der Datenströme und die Umsetzung der Logik wurde Node-RED verwendet. Durch die visuelle Flow-Programmierung eignet sich Node-RED besonders für IoT-Projekte und ermöglicht eine schnelle Entwicklung der Kommunikations- und Steuerungslogik.

Die Kommunikation zwischen den Geräten erfolgt über MQTT. Hierfür wurde der MQTT-Broker HiveMQ eingesetzt. MQTT eignet sich aufgrund seines geringen Ressourcenverbrauchs besonders

für IoT-Geräte und ermöglicht eine zuverlässige Echtzeitkommunikation zwischen den einzelnen Komponenten.

Zur Absicherung der Webanwendungen wurde nginx als Reverse Proxy eingesetzt. nginx übernimmt dabei unter anderem die Weiterleitung der Anfragen an interne Dienste sowie die Bereitstellung verschlüsselter HTTPS-Verbindungen.

Für die Personenerkennung kam eine YOLO-basierte Kamera zum Einsatz. YOLO ermöglicht die Echtzeiterkennung von Personen innerhalb eines Kamerabildes und eignet sich dadurch zur Erfassung von Bewegungen und Linienüberschreitungen innerhalb des Systems.

2.3 Projektorganisation

Die Umsetzung des Projekts erfolgte arbeitsteilig innerhalb des Teams. Die einzelnen Aufgabenbereiche wurden aufgeteilt, sodass verschiedene Komponenten parallel entwickelt und anschließend in das Gesamtsystem integriert werden konnten. Gleichzeitig gab es je nach Bedarf auch eine gegenseitige Unterstützung und gemeinsame Bearbeitung von Teilbereichen.

Zur Kommunikation und allgemeinen Zusammenarbeit im Team wurde eine eigene Messenger-Gruppe eingerichtet, ergänzend fand teils auch ein Austausch via Discord in der Freizeit bzw. Teams im Betrieb oder sogar in persönlichen Treffen in der Freizeit statt. Ein für frühere Schulprojekte gemeinsam genutztes und abgesichertes Wiki ermöglichte darüber hinaus auch die Dokumentation von einzelnen, oft zentral abzurufenden Informationen zum Projekt. Diese Plattform wurde teils auch für den gemeinsamen Datenaustausch genutzt. Darüber hinaus wurden die einzelnen, individuellen Arbeitsfortschritte im Logbuch des schulischen Moodle-Kurses dokumentiert.

Die ungefähre Arbeitsaufteilung im Team kann schwerpunktmäßig wie folgt beschrieben werden:

Teammitglied	**Aufgabenbereiche/-schwerpunkte**
Lars	MQTT-Kommunikation, Beschaffung der LED-Komponenten, Konfiguration der ESP32-Geräte sowie Unterstützung bei der MQTT- und ESP-Anbindung
Nico	Einrichtung und Konfiguration der YOLO-Kamera einschließlich Vorbereitung der Anbindung an das Gesamtsystem
Philipp	Unterstützung bei der ESP32-Integration sowie Entwicklung und Anbindung der Flows in Node-RED
Daniel	Einrichtung der Backend-Infrastruktur (Linux-Server, Sicherheitskonzept, Docker, Node-RED und PostgreSQL), Beschaffung und Bereitstellung der RFID-Komponenten sowie Schwerpunkt Dokumentation

Die Integration der einzelnen Komponenten erfolgte im Ergebnis sodann schrittweise im Rahmen einzelner und letztlich auch gemeinsamer Testläufe. Dabei wurden insbesondere die Kommunikation über MQTT sowie das Zusammenspiel zwischen RFID-System, Backend und Personenerkennung überprüft.

Auch beim Erstellen der Dokumentation und Präsentation wurden parallel individuelle Arbeiten durchgeführt, welche später in je eine zentrale Datei zusammengefügt wurden. Somit konnte jedes Teammitglied seine Arbeitsbereiche, teils eben auch in Abstimmung bei Überschneidungen, bearbeiten.

3. Technische Umsetzung

Im Folgenden soll die Systemarchitektur näher und detaillierter, inklusive Unterpunkte für die einzelnen, wesentlichen Komponenten, beschrieben werden.

3.1 Systemarchitektur

Die Systemarchitektur des Projekts basiert auf einer modularen IoT-Infrastruktur, bei welcher die einzelnen Komponenten über MQTT miteinander kommunizieren. Ziel des Aufbaus war eine klare Trennung der verschiedenen Aufgabenbereiche sowie eine einfache Erweiterbarkeit des Systems.

Im Zentrum der Kommunikation befindet sich der MQTT-Broker HiveMQ. Über diesen tauschen die einzelnen Komponenten ihre Daten und Statusinformationen aus. Die ESP32-Mikrocontroller übernehmen dabei die Kommunikation mit den angeschlossenen Hardwarekomponenten wie dem RFID-Sensor sowie dem LED-Band.

Zur Überprüfung der Zugangsberechtigungen sendet der RFID-ESP32 die ausgelesene UID eines RFID-Chips über MQTT an das Backend-System. Die Verarbeitung der Daten erfolgt anschließend innerhalb eines Node-RED-Flows. Node-RED läuft gemeinsam mit einer PostgreSQL-Datenbank auf einem zentralen Linux-Server innerhalb von Docker-Containern.

Die PostgreSQL-Datenbank speichert die autorisierten RFID-UIDs. Node-RED überprüft die empfangenen UID-Daten und entscheidet anhand der Datenbankeinträge, ob ein Zugriff erlaubt oder verweigert wird. Das Ergebnis wird anschließend erneut über MQTT an einen zweiten ESP32 übertragen, welcher die entsprechende LED-Steuerung übernimmt.

Zusätzlich wurde eine kamerabasierte Personenerkennung integriert. Hierbei verarbeitet eine YOLO-Kamera Bilddaten und erkennt Personenbewegungen anhand definierter Linienüberschreitungen. Die erkannten Ereignisse werden ebenfalls über MQTT an das Gesamtsystem übertragen und innerhalb von Node-RED weiterverarbeitet.

Die zentrale Backend-Infrastruktur wurde auf einem Linux-Server umgesetzt und durch verschiedene Sicherheitsmechanismen abgesichert. Dazu gehören unter anderem HTTPS-Verschlüsselung, ein Reverse Proxy mittels nginx, Firewall-Regeln sowie die Containerisierung der Dienste durch Docker.

Die folgende Abbildung zeigt die vereinfachte Gesamtarchitektur des Systems sowie die Kommunikation zwischen den einzelnen Komponenten:

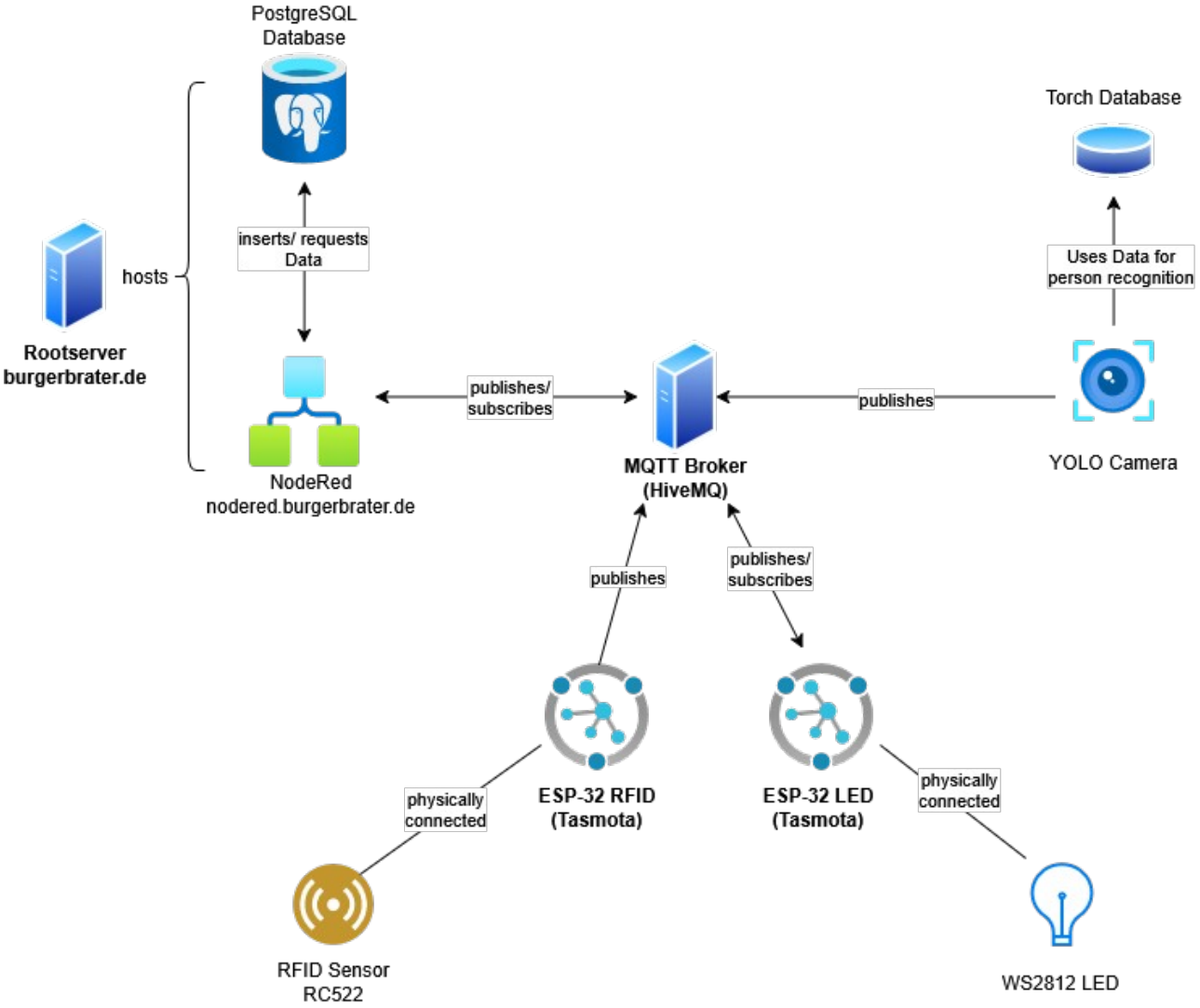

Abb. 1: Grafische Darstellung der Gesamtarchitektur

3.2 Sicherheitskonzept

Da innerhalb des Projekts verschiedene netzwerkbasierte Komponenten miteinander kommunizieren, spielte die Absicherung der Infrastruktur eine wichtige Rolle. Ziel des Sicherheitskonzepts war der Schutz des zentralen Backend-Systems sowie der Kommunikationswege zwischen den einzelnen Komponenten.

Der Zugriff auf den Linux-Server erfolgt ausschließlich über SSH mit schlüsselbasierter Authentifizierung. Zusätzlich wurde der direkte Root-Login deaktiviert und der standardmäßige SSH-Port angepasst, um automatisierte Angriffe auf den Server zu erschweren.

Zur weiteren Absicherung des Systems wurde eine Firewall mittels UFW (Uncomplicated Firewall) eingerichtet. Dabei wurden ausschließlich die für den Betrieb notwendigen Netzwerkports

freigegeben. Zusätzlich wurde mit Fail2Ban ein Schutzmechanismus gegen wiederholte Fehlanmeldungen implementiert. Verdächtige Zugriffsversuche werden hierbei automatisch erkannt und temporär blockiert.

Für den sicheren Zugriff auf die Webanwendungen wurde HTTPS eingesetzt. Die HTTPS-Verschlüsselung erfolgt über nginx als Reverse Proxy in Verbindung mit Let's Encrypt-Zertifikaten. Dadurch können Webdienste wie Node-RED verschlüsselt und abgesichert bereitgestellt werden.

Auch die Kommunikation über MQTT wurde abgesichert. Der eingesetzte HiveMQ-Broker unterstützt verschlüsselte TLS-Verbindungen sowie eine Benutzer- und Rechteverwaltung. Dadurch können Zugriffe auf bestimmte Topics gezielt eingeschränkt werden.

Zusätzlich wurde Node-RED nicht direkt öffentlich erreichbar betrieben. Der Zugriff erfolgt ausschließlich über den Reverse Proxy von nginx. Darüber hinaus wurde der Zugriff auf die Node-RED-Oberfläche durch eine Benutzeranmeldung geschützt.

Auch die eingesetzten ESP32-Geräte wurden abgesichert. Die Konfiguration der Geräte erfolgt über die lokale Weboberfläche von tasmota, welche durch Benutzername und Passwort geschützt wurde. Dadurch sollte verhindert werden, dass unbefugte Personen Änderungen an den Geräteeinstellungen oder der MQTT-Konfiguration vornehmen können. Zusätzlich wurden die Geräte ausschließlich innerhalb des lokalen Netzwerks betrieben und in die abgesicherte MQTT-Kommunikation eingebunden.

Durch die Kombination dieser Maßnahmen konnte eine grundlegende Absicherung der Infrastruktur sowie der Kommunikationswege innerhalb des Systems umgesetzt werden.

3.3 Hardwarekomponenten

Nachdem das grobe Konzept für die Projektumsetzung einmal erarbeitet war, konnte sich um das Heraussuchen und Bestellen der eingesetzten, technischen Hardware gekümmert werden. Bei den unmittelbar von uns beschafften, also extra für das Projekt sodann gekauften, Hardwarekomponenten handelt es sich um folgende Produkte:

Produkt	Beschreibung
2x AZDelivery 3X ESP32 D1 Mini USB-C	Die beiden (später mit tasmota geflashten) ESPs
1x BOJACK Breadboard-Kit	Zwei Steckbretter mit Kabeln für das Anschließen von LED-Band und RFID-Sensor an die jeweiligen ESPs
2x Hama USB-A auf USB-C Datenkabel	Anschließen der ESPs an Client / Stromquelle
1x AZDelivery RFID-Kit RC522	RFID-Sensor RC522 mit RFID-Chip und RFID-Karte
1x W2812b LED-Band	LED-Band zum Zurechtschneiden auf individuelle Länge

Die gelieferten ESPs waren zunächst mit den Aufsteck-PINs einzeln geliefert worden. Hier musste zunächst eine Option fürs Löten gefunden werden.

Für die Anbindung an YOLO wurde eine handelsübliche Kamera verwendet. YOLO selbst läuft lokal auf dem Laptop eines Teamkollegen.

Für die zentrale Backend-Infrastruktur fiel die Wahl des Anbieters auf netcup als einer der etablierten, sicheren und professionellen Internet Service Provider in Deutschland. Mit Blick auf potenzielle Erweiterungen des Projekt (einfachere Skalierbarkeit) und der Verfügbarkeit von dedizierten Ressourcen hierfür, wurde sich für das Modell des eigenen Root-Servers entschieden. Konkret wurde der RS 1000 G12 mit 8GB DDR5 RAM, 4 dedizierten Kernen und 256 GB NVMe ausgewählt. Auf der Softwareseite des Servers kam das Betriebssystem Debian 13 „Trixie“ zum Einsatz.

Schlussendlich sei auch noch erwähnt, dass KI als modernes, allgemeines Hilfsmittel Einzug in verschiedene Anwendungsbereiche, Arbeitsschritte und Entwicklungsstufen hielt.

3.3.1 RFID-Technologie

An dieser Stelle soll nun zunächst ganz kurz etwas auf das theoretische Basiswissen zum Thema RFID eingegangen sowie anschließend der Bogen zu unserem konkreten Anwendungsfall geschlagen werden.

RFID steht für „Radio Frequency Identification“ und beschreibt ein Verfahren zur kontaktlosen Identifikation von Objekten über Funkwellen. Dabei kommunizieren ein RFID-Lesegerät sowie ein RFID-Tag miteinander. Der RFID-Tag enthält eine eindeutige Kennung, die sogenannte UID (Unique Identifier). Mithilfe dieser UID kann ein Objekt oder Benutzer eindeutig identifiziert werden.

In RFID-Systemen existieren verschiedene Frequenzbereiche, welche sich hinsichtlich Reichweite, Datenrate und Einsatzgebiet unterscheiden. Low-Frequency-Systeme arbeiten typischerweise im Bereich von 125–134 kHz und werden beispielsweise in Zeiterfassungs- oder Zugangssystemen eingesetzt. High-Frequency-Systeme arbeiten im Bereich von 13,56 MHz und unterstützen unter anderem NFC-basierte Kommunikation. Ultra-High-Frequency-Systeme ermöglichen größere Reichweiten und werden beispielsweise im Bereich der Logistik oder Warensicherung eingesetzt.

Im Rahmen des Projekts wurde ein RFID-RC522-Modul von AZDelivery verwendet. Das Modul arbeitet im High-Frequency-Bereich bei 13,56 MHz und unterstützt die Kommunikation mit NFC-kompatiblen RFID-Tags. Als RFID-Tags wurden RFID-Chips beziehungsweise Karten mit eindeutiger UID eingesetzt.

Zur Authentifizierung hält ein Benutzer seinen RFID-Chip in die Nähe des RC522-Lesegeräts. Das Lesegerät erzeugt dabei ein elektromagnetisches Feld, über welches der RFID-Tag kontaktlos mit Energie versorgt wird. Anschließend überträgt der RFID-Tag seine UID an den RFID-Sensor. Die Reichweite der Kommunikation liegt typischerweise im Bereich weniger Zentimeter.

Unser RFID-Sensor ist mit einem ESP32-Mikrocontroller verbunden. Nach dem Auslesen der UID wird diese vom ESP32 verarbeitet und über MQTT an das zentrale Backend-System übertragen.

Dort erfolgt anschließend die Überprüfung der Zugangsberechtigung anhand der in der PostgreSQL-Datenbank gespeicherten RFID-UIDs.

Das im Projekt verwendete System basiert auf einer reinen UID-basierten Authentifizierung und dient primär Demonstrations- und Lernzwecken. Professionelle Zugangssysteme verwenden darüber hinaus zusätzlich kryptographische Verfahren zur sicheren Authentifizierung der RFID-Tags.

3.3.2 ESPs

Für das Projekt verwenden wir zwei Tasmota-geflashte ESP-32 Mikrocontroller.
Um eine ESP-32 mit Tasmota zu flashen, benötigen wir zunächst den richtigen USB-Treiber, über den eine serielle Kommunikation (UART) möglich ist (Treiber: https://www.silabs.com/software-and-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads) und ein UBS zu USB-C Kabel, welches Daten übertragen kann. Zudem sollte das Kabel nur mit 3,3V die ESP-32 mit Strom versorgen.

Das Flashen der ESP-32 ist so weit grundsätzlich relativ simpel, über den Tasmota-Webinstaller. Dazu eignet sich jedoch nicht jeder Browser, denn manche Browser blockieren den Zugriff auf USB-Schnittstellen. Wir haben in unserem Fall den Chrome Browser verwendet.

Webinstaller: https://tasmota.github.io/install/

Da es bei uns jedoch vermehrt zu Problemen kam, haben wir uns entschieden, einer der ESP-32 manuell per Windows Terminal zu flashen. Dafür benötigt man die richtige tasmota32.factory.bin Datei, die alle Daten zum Flashen von Tasmota beinhaltet: https://ota.tasmota.com/tasmota32/

Anschließend benötigt man Python, um ein Python Modul zu installieren, welches das manuelle Flashen von Tasmota über das Terminal ermöglicht:
https://www.python.org/ftp/python/3.14.4/python-3.14.4-amd64.exe

Über folgenden Python Befehl installiert man das benötigte Tool „esptool“:
py -m pip install esptool

Bevor wir die ESP-32 flashen, sollten wir die aktuellen Daten der ESP-32 löschen, um Datenrendundanz zu vermeiden:
py -m esptool --chip esp32 --port COM3 erase_flash

COM3 = USB Port. Kommt drauf an, an welchem Port die ESP-32 angeschlossen ist.
Das kann man unter dem "Geräte Manager" in Windows herausfinden.
Nun flashen wir die ESP-32 mit Tasmota:
py -m esptool --chip esp32 --port COM3 write_flash 0x0 tasmota32.factory.bin

Nun greifen wir auf die Tasmota Webkonsole zu: https://tasmota.github.io/install/.

Ist die ESP-32 noch nicht mit dem WLAN verbunden sein, so klicke auf "Change Wi-Fi" und anschließend auf "Visit Device".

Sollte die ESP-32 sich noch in einem anderen Netz befinden, so kann man auch über die Webkonsole die ESP-32 mit einem anderen WLAN-Netz verbinden.
Dazu geht man unter "Logging & Console", nachdem man den Tasmota Webinstaller offen hat und gibt in die Konsole folgendes ein:

Backlog SSID1 <WLAN SSID>; Password1 <PASSWORT>

Nun kommen wir zur Verkabelung der ESPs mit den jeweiligen Sensoren:
Eine der ESP-32 ist per Jumper-Kabel an den RC522 RFID Sensor angeschlossen, die andere ESP-32 per Jumper-Kabel an einem WS2812b LED-Streifen.

Der RC522 RFID-Sensor ist mit insgesamt sieben Jumper-Kabeln an einer der zwei ESP-32 verbunden:

VCC	Spannung von 3,3 Volt
GND	Ground (Erdung)
SCK	Serial Clock (Taktleitung)
MISO	Master in Slave Out (Daten vom Reader zur ESP)
MOSI	Master out Slave in (Daten von der ESP zum Reader)
SDA / SS	Slave Select (Aktiviert den Reader)
RST	Reset (Setzt den Reader zurück)

Die WS2812b LED Streifen sind mit insgesamt drei Jumper-Kabeln an der anderen ESP-32 angeschlossen:

VCC	Spannung von 3,3 Volt
GND	Ground (Erdung)
Data	Dateneingang zur Steuerung der LEDs

Pins, über die Daten gesendet oder empfangen werden, müssen noch in Tasmota programmiert werden. Diese nennt man dann GPIO, das steht für General Purpose Input/Output. Über Tasmota wählt man zur jeweiligen GPIO die Peripheral Function bzw. Hardware-Funktion aus, damit die Daten im richtigen Format gesendet / empfangen werden können.

In unserem Fall möchten wir Daten an die beiden ESP-32 per MQTT-Broker senden und empfangen. Damit die ESPs Daten senden und empfangen können, müssen diese im WLAN-Netz eingebunden werden. Hierfür verwenden wir einen mobilen Hotspot, mit dem sich die beiden ESPs verbunden haben. Über die IP- Adresse der ESPs kommt man auf die jeweilige Tasmota Webkonsole, auf der man sämtliche Konfigurationseinstellungen vornehmen kann.

Damit der Zugriff auf die Konsole eingeschränkt ist, haben wir für beide Tasmota-geflashten ESPs ein HTTP-Passwort gesetzt. Für jede ESP-32 können wir unter dem Button „MQTT“ einen MQTT-Nutzer hinzufügen und ein Topic auswählen, über das dies ESPs entweder Daten empfangen (subscriben) oder senden (publishen).

Da die ESP-32 mit den W2812b LED-Streifen nur Daten empfängt und verarbeitet, verwenden wir hierfür einen MQTT-Nutzer, der nur Zugriff hat, Daten zu empfangen (subscriben).

Die ESP-32 mit dem anhängenden RC522 RFID Sensor sendet ausschließlich nur Daten, daher verwenden wir für diese ESP einen Nutzer, der nur Zugriff hat, Daten zu senden (publishen).

3.3.3 Besonderheit des RFID-ESPs

Der Einsatz von RFID stellte uns zunächst vor ungeplante Herausforderung und verzögerte weitere Schritte zu Beginn des Projekts. Konkret ging es hier um eine Besonderheit mit der zu verwendenden Tasmota-Version. Der eingesetzte RFID-ESP musste überhaupt erst einmal in der Lage sein den RC522-Sensor ansprechen zu können.

Nach anfänglichen Tasmota-Flashversuchen stellte sich heraus, dass die Standard-Version von Tasmota für den ESP32 keine Kompatibilität mit dem RC522-Sensor besitzt. Um unseren RFID-Sensor mit dem ESP verbinden zu können, brauchte es zwingend eine spezielle Tasmota-Version, in welcher sich die GPIOs so konfigurieren lassen, dass der konkrete RC522-Sensor bei der Konfiguration auswählbar ist. Ein beispielhaftes Setting muss folgendermaßen aussehen:

RC522-Verkabelung	Beispielhafte GPIOs am ESP	Konfiguration in Tasmota
SCK	GPIO18	SPI CLK
MISO	GPIO19	SPI MISO
MOSI	GPIO23	SPI MOSI
SDA	GPIO5	RC522 CS
RST	GPIO22	RC522 Rst

In den vorgegebenen Versionen des Tasmota-Flashers sind die Konfigurationen „RC522 CS“ und „RC522 Rst“ nicht auswählbar. Damit ist mit den „Standard-Versionen“ von Tasmota auch keine Anbindung des RC522-Sensors möglich gewesen.

Zunächst wurde über die offizielle GitHub-Seite des Herstellers nach einer bereits gebauten Lösung für das Problem gesucht (https://tasmota.github.io/docs/ bzw. Unterseiten zum damaligen Aufruf https://tasmota.github.io/docs/Download/). Eine spezielle tasmota32-sensors.bin sollte die gewünschte Anbindung schließlich bieten, auch wenn mehrere Forenbeiträge dazu verschiedene Aussagen trafen. Diese Version wurde schließlich heruntergeladen und der RFID-ESP damit über den Online-Flasher von Tasmota installiert. Leider stellte sich trotz seriös wirkender Anleitung heraus, dass der RC522-Sensor auch mit dieser Version nicht ansprechbar war.

Letztlich brachte ein offizieller Beitrag in den Tasmota-Docs doch die Gewissheit (https://tasmota.github.io/docs/MFRC522/), es musste eine selbst kompilierte Version erstellt werden, indem individuelle Konfigurationsanweisungen eingebaut werden mussten:

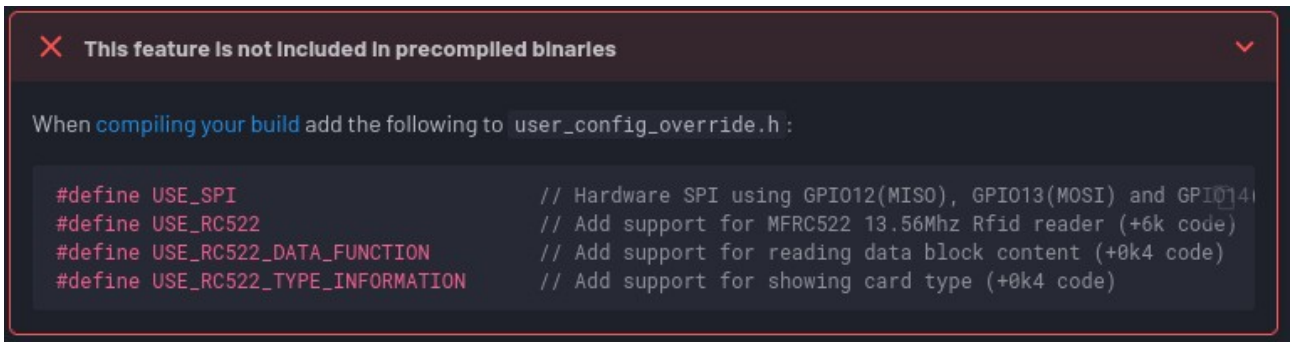

Abb. 2: Screenshot auf GitHub zu den individuellen Konfigurationsanweisungen beim selbst kompilieren für den RC522-Einsatz

Wie oben teils schon angeklungen, gibt es mehrere Möglichkeiten des Kompilierens und auch Flashens einer eigenen Tasmota-Version. So lässt sich beispielsweise eine wohl bewährte Version zum Kompilieren bei GitHub finden (https://github.com/benzino77/tasmocompiler), womit dies über eine grafische Oberfläche möglich ist und hier die individuellen Konfigurationsanweisungen berücksichtigt werden können.

Das Flashen mittels esptool ist bereits oben ausführlicher beschrieben worden. Die erstellte, selbst kompilierte tasmota32.bin konnte dann durch ein manuelles Flashen erfolgreich auf den ESP32 geladen werden:
esptool --chip esp32 --port /dev/ttyUSB0 --baud 460800 write_flash 0x10000 tasmota32.bin

Somit war auch der RFID-Sensor ansteuerbar und der RFID-ESP funktionierte endlich final.

3.4 MQTT-Broker

Als MQTT-Broker verwenden wir HiveMQ, da diese eine einfache und übersichtliche Oberfläche anbieten. Der Datenverkehr zwischen den ESPs und dem Broker ist über HiveMQ TLS-verschlüsselt und daher werden Daten über dem Port 8883 versendet und empfangen. Die TLS-Verschlüsselung bietet eine höhere Sicherheit, sodass kein Dritter die Daten mitlesen und verändern kann.

Auf dem Broker existieren zwei Nutzer:

broker_pub	Dieser hat nur die Berechtigung, Daten auf den Broker senden zu können
nodered_backend	Dieser hat die Berechtigung, Daten vom Broker empfangen und senden zu können

Zudem sind die beiden Nutzer jeweils mit einem 20-stelligen Passwort gesichert, sodass die Nutzer von einer Brute-Force Attacke geschützt sind.

Zur besseren Übersicht und für einen dezentraleren Aufbau, haben wir uns dazu entschieden, zwei Topics für den Broker zu erstellen. Ein Topic für den RC522 RFID Sensor (tele/esp32-rfid/SENSOR) und ein Topic für die WS2812b LED Streifen (cmnd/esp32-led/<command>).

3.5 Backend

Für die zentrale Verarbeitung und Verwaltung der Systemkomponenten wurde eine eigene Backend-Infrastruktur auf Basis eines Linux-Servers umgesetzt. Das Backend übernimmt unter anderem die Bereitstellung der Dienste, die Verarbeitung der MQTT-Daten sowie die Speicherung und Verwaltung der RFID-Zugangsdaten. Zusätzlich wurden verschiedene Sicherheitsmaßnahmen zur Absicherung des Systems umgesetzt. Im Folgenden soll dies systematisch dargestellt werden.

3.5.1 Server-Infrastruktur und Grundkonfiguration

Als zentrale Backend-Infrastruktur wurde ein virtueller Linux-Server des Anbieters netcup eingesetzt (RS 1000 G12 mit Debian 13 „Trixie“). Der Server dient als zentrale Plattform für die Bereitstellung der einzelnen Dienste sowie für die Verarbeitung und Speicherung der Daten innerhalb des Systems.

Nach der Bereitstellung des Servers erfolgte zunächst die grundlegende Konfiguration des Systems, wobei auf entsprechende Sicherheitseinstellungen besonders hoher Wert gelegt wurde. Hierzu gehörten unter anderem die Aktualisierung der installierten Pakete sowie die Einrichtung eines separaten Benutzerkontos für die Administration des Servers.

Für den Fernzugriff auf den Server wurde SSH verwendet. Dabei wurde der Zugriff ausschließlich über eine schlüsselbasierte Authentifizierung ermöglicht (SSH keys). Zusätzlich wurde der direkte Root-Login deaktiviert und der standardmäßige SSH-Port angepasst, um automatisierte Angriffe auf den Server zu erschweren. An allen Stellen wo Passwörter für einen Login benötigt werden, sind eigene, lange, sichere Passwörter gesetzt (zufällig generiert, 15-20 Zeichen lang).

Zur weiteren Absicherung der Infrastruktur wurde eine Firewall mithilfe von UFW eingerichtet. Dabei wurden ausschließlich die für den Betrieb notwendigen Netzwerkports freigegeben. Zusätzlich kam mit Fail2Ban ein Schutzmechanismus zum Einsatz, welcher wiederholte Fehlanmeldungen automatisch erkennt und entsprechende Zugriffe temporär blockiert.

```
burgerbrater@iot-server:~$ sudo ufw status
[sudo] password for burgerbrater:
Status: active

To                         Action      From
--                         ------      ----
51123                      ALLOW       Anywhere
80                         ALLOW       Anywhere
443                        ALLOW       Anywhere
51123 (v6)                 ALLOW       Anywhere (v6)
80 (v6)                    ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)

burgerbrater@iot-server:~$
```

Abb. 3: Screenshot SSH-Sitzung zeigt aktivierte Firewall-Regeln mittels UFW

Durch diese Maßnahmen konnte eine grundlegende Absicherung des Backend-Systems umgesetzt werden.

3.5.2 Domain, nginx und Reverse Proxy

Für den externen Zugriff auf die bereitgestellten Dienste wurde eine eigene Domain verwendet. Hierzu wurden entsprechende DNS-Einträge beim Domainanbieter konfiguriert, sodass einzelne Dienste über Subdomains erreichbar sind.

Zur Bereitstellung der Webanwendungen kam nginx als Reverse Proxy zum Einsatz. nginx übernimmt dabei die Weiterleitung eingehender Anfragen an die intern laufenden Dienste innerhalb der Docker-Container. Dadurch mussten die einzelnen Dienste nicht direkt öffentlich erreichbar betrieben werden.

Für Node-RED wurde beispielsweise die Subdomain „nodered.burgerbrater.de“ eingerichtet. Eingehende Anfragen werden hierbei zunächst von nginx verarbeitet und anschließend an den intern laufenden Node-RED-Dienst weitergeleitet.

Zusätzlich wurde HTTPS mithilfe von Let’s Encrypt-Zertifikaten eingerichtet. Dadurch erfolgt die Kommunikation zwischen Browser und Server verschlüsselt. Die Zertifikate wurden automatisiert über Certbot erstellt und in nginx eingebunden.

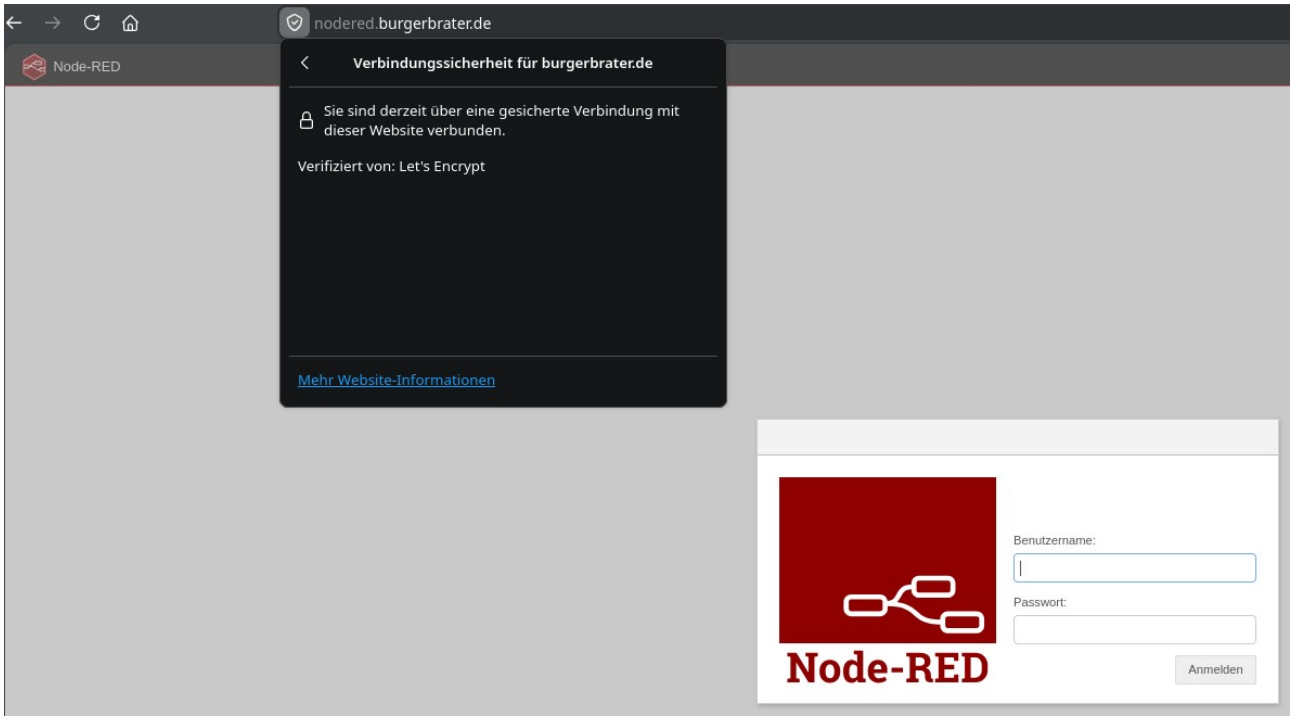

Abb. 4: Screenshot von nodered.burgerbrater.de im Browser mit HTTPS und Node-Red Anmeldemaske

Durch die Kombination aus Reverse Proxy und HTTPS konnte eine sichere sowie zentral verwaltete Bereitstellung der Webdienste umgesetzt werden. In dieser Umgebung konnten die Teamkollegen sodann ihre weitere Bearbeitungen fortsetzen. Auch hier ist der Zugriff natürlich durch einen Login abgesichert.

Auch eine kleine Startseite für das Projekt (burgerbrater.de) wurde erstellt. Diese dient jedoch auch eher zu Demonstrationszwecken und ist nicht zwingend für eine Gesamtfunktionalität erforderlich.

3.5.3 Containerisierung mit Docker

Zur Bereitstellung der zentralen Dienste wurde Docker eingesetzt. Docker ermöglicht die containerisierte Ausführung einzelner Anwendungen und sorgt dafür, dass Dienste unabhängig voneinander betrieben werden können.

Für das Projekt wurden unter anderem Node-RED sowie die PostgreSQL-Datenbank innerhalb separater Docker-Container ausgeführt. Dadurch konnten die einzelnen Komponenten sauber voneinander getrennt und zentral verwaltet werden.

Die Verwaltung der Container erfolgte mithilfe von Docker Compose. Hierbei wurden die benötigten Dienste innerhalb einer YAML-Datei definiert. Dadurch konnten die einzelnen Container sowie deren Netzwerke, Ports, Sicherheitseinstellungen und persistenten Speicherbereiche zentral konfiguriert und automatisiert gestartet werden.

Abb. 5: Screenshot SSH-Sitzung zeigt laufende Docker-Container

Zusätzlich wurden Docker-Volumes eingesetzt, um wichtige Daten dauerhaft zu speichern. Dadurch bleiben beispielsweise Node-RED-Flows sowie Datenbankinhalte auch nach einem Neustart oder einer Aktualisierung der Container erhalten.

Durch die Nutzung von Docker konnte die Infrastruktur übersichtlich, modular und einfach erweiterbar umgesetzt werden. Gleichzeitig erleichtert die Containerisierung die Wartung sowie die Wiederherstellung einzelner Dienste und auch eine potenzielle Erweiterung in der Zukunft.

3.5.4 Bereitstellung von Node-RED

Zur Verarbeitung der MQTT-Datenströme sowie zur Umsetzung der Steuerungslogik wurde Node-RED eingesetzt. Node-RED wurde innerhalb eines Docker-Containers bereitgestellt und über Docker Compose in die bestehende Backend-Infrastruktur integriert.

Der Zugriff auf die Node-RED-Weboberfläche erfolgt über den zuvor eingerichteten Reverse Proxy von nginx. Hierfür wurde eine eigene Subdomain eingerichtet und zusätzlich eine HTTPS-Verschlüsselung umgesetzt. Dadurch konnte die Weboberfläche sicher und zentral bereitgestellt werden.

Zusätzlich wurde die Node-RED-Oberfläche durch eine Benutzeranmeldung abgesichert. Dadurch sollte verhindert werden, dass unbefugte Personen Änderungen an den Flows oder der Konfiguration vornehmen können.

Für die dauerhafte Speicherung der Node-RED-Konfigurationen sowie der erstellten Flows wurden persistente Docker-Volumes verwendet. Dadurch bleiben die Daten auch nach Neustarts oder Aktualisierungen der Container erhalten. Optional können hier auch lokale „Backups“ auf dem Server durchgeführt werden, indem der Inhalt des Volume-Ordners kopiert und damit zwischengespeichert wird.

3.5.5 Bereitstellung von PostgreSQL-Datenbank

Für die Speicherung der RFID-Zugangsdaten wurde eine PostgreSQL-Datenbank eingesetzt. Die Datenbank wurde ebenfalls innerhalb eines Docker-Containers betrieben und in die bestehende Backend-Infrastruktur integriert.

Innerhalb der Datenbank wurden die autorisierten RFID-UIDs gespeichert. Node-RED greift auf diese Datenbank zu, um empfangene RFID-UIDs zu überprüfen und Zugriffsentscheidungen zu treffen.

Zur Verwaltung der Daten wurde eine eigene Datenbanktabelle erstellt, in welcher unter anderem die UID der RFID-Chips sowie zusätzliche Informationen gespeichert werden können. Die Kommunikation zwischen Node-RED und PostgreSQL erfolgt innerhalb des internen Docker-Netzwerks.

```
burgerbrater@iot-server:/opt/iot-backend$ docker exec -it postgres psql -U nodered -d access_control
psql (16.13 (Debian 16.13-1.pgdg13+1))
Type "help" for help.

access_control=# SELECT * FROM rfid_users;
 id |      name       |    uid     | active |         created_at
----+-----------------+------------+--------+----------------------------
  1 | Testkarte       | TEST123456 | t      | 2026-05-07 11:55:34.426369
  2 | Zugangskarte 1 | 0AE10C06   | t      | 2026-05-08 16:30:54.738435
(2 rows)

access_control=#
```

Abb. 6: Screenshot SSH-Sitzung zeigt PostgreSQL-Tabelle

Auch für PostgreSQL wurden persistente Speicherbereiche eingerichtet, sodass Datenbankinhalte dauerhaft gespeichert bleiben und Neustarts der Container keine Datenverluste verursachen.

Durch die Nutzung von PostgreSQL konnte eine strukturierte und zuverlässige Verwaltung der RFID-Zugangsdaten umgesetzt werden. Perspektivisch können auch hier weitere Anpassungen vorgenommen werden. Beispielsweise sollte unter „name“ auch der Name des Karteninhabers zu finden sind. Für den hier genutzten Demonstrationszweck und auch zu Testzwecken, wurde auf eine genauere Spezifikation verzichtet. Es soll dennoch grundsätzlich angemerkt sein, dass hier für eine Erweiterung hin in Richtung eines Produktivsystems weitere Schritte nötig wären.

3.6 Node-RED

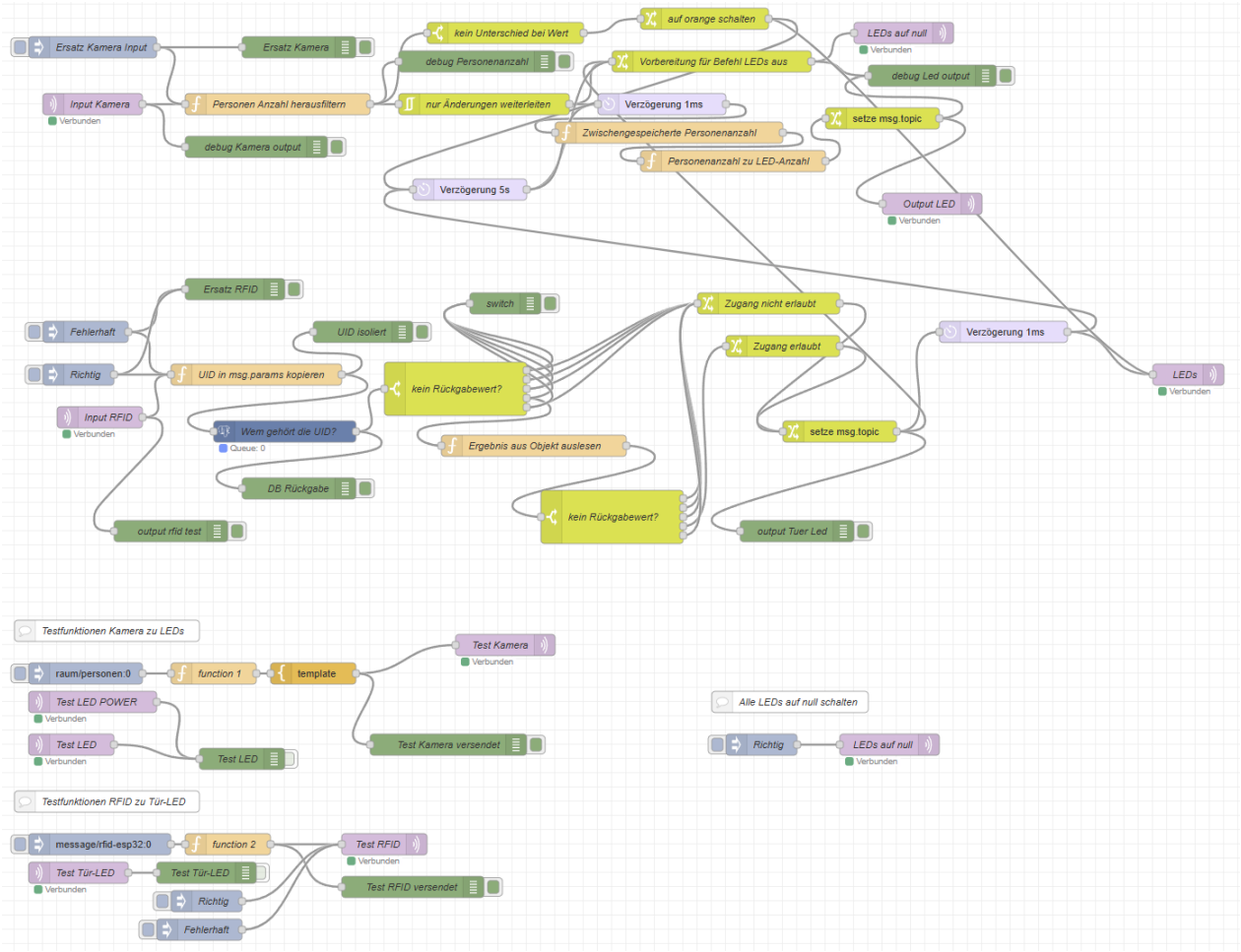

Abb. 7: Screenshot des finalen Aufbaus in Node-RED

Der Aufbau von Node-RED teilte sich anfangs in zwei unabhängige Flows, die durch ein zusammenlegen der LEDs überschneidende Elemente erhielten. Das Zusammenlegen der LEDs brachte zusätzliche Probleme mit sich. So musste nach einer visuellen Rückgabe des Ergebnisses des RFID-Scans wieder auf die Personenanzahl umgeschaltet werden. Dafür muss die Anzahl der Personen, die im Raum befindlich sind, zwischengespeichert werden, welches keine Funktion eines der Standartknoten in Node-RED ist. Lösung war schließlich in Funktionsknoten die Befehle flow.set([name: string], [value: any]) und flow.get([name: string]) zu verwenden, welche einen Wert unter einem Namen abspeichern können und im Anschluss mithilfe dieses Namens wieder aufrufen können.

Einen eigenen Block erhielten Testfunktionen zum Simulieren der Ein- und Ausgabe der MQTT-Signale der Kamera, des RFID-Scanners und des LED-Streifens.

3.6.1 Flow Kamera zu LED

Über einen MQTT-Knoten wird über das Topic „raum/personen“ ein String empfangen, der mehrere Informationen enthält. Aus diesem wird mithilfe eines Funktionsknotens die Personenanzahl herausgelesen und für spätere Verwendung über flow.set() abgespeichert. Nun folgt eine Aufteilung des Flows in einen Knoten, der nur ein Signal weitergibt, wenn die Personenanzahl gleich des letzten Wertes ist und einem der nur Wertänderungen weitergibt.

Gehen wir vorerst auf den ein, der Änderungen der Personenanzahl weitergibt. Das Signal wird hier wieder in zwei Knoten weitergeleitet. Ersterer ist ein Change-Knoten, der das Topic auf „cmnd/esp32-led/Led“ ändert und den Payload ändert auf einen String, der aus einer Wiederholung des Strings „000000“ besteht. Diese Wiederholung entspricht der Anzahl der LEDs auf dem LED-Streifen, der die Personenanzahl zählt und hat die Funktion alle Lampen zu reseten, bevor die Personenanzahl erneut durchgegeben wird. Dies wird anschließend über einen MQTT-Knoten an den Esp32 mit dem entsprechenden LED-Streifen weitergeleitet.

Der zweite Knoten ist ein Delay-Knoten, der das Signal um 1ms verzögert, um zu gewährleiten, dass das „Reset-Signal“ zuerst verschickt wird. Nach dieser Verzögerung wird in einem Funktionsknoten die Anzahl der Personen mithilfe von flow.get() erneuert, welches in diesem Fall nicht nötig wäre, aber für weitere Knoten, die ein Signal auf diesem Wege verschicken werden, benötigt wird. Ein weiterer Funktionsknoten wandelt die Personenanzahl in einen String mit der entsprechenden Anzahl an „0000FF“ um und ergänzt „000000“ bis die Gesamtanzahl beider 20 beträgt. Dies wird zur Folge haben, dass auf dem LED-Streifen eine der Personenanzahl entsprechende Anzahl an Lämpchen blau leuchten werden. Nun wird noch das Topic in einem Change-Knoten auf „cmnd/esp32-led/Led“ gesetzt und anschließend über einen MQTT-Knoten weitergeleitet an die LEDs.

Der Flow, der nur ein Signal weitergibt, wenn keine Änderung der Personenanzahl stattgefunden hat im Vergleich zum letzten Signal, leitet ein Signal an einen Change-Knoten weiter. Dieser Ändert das Topic auf „cmnd/esp32-led/Color“ und den Payload auf „FFA500“. Dies entspricht der Farbe Orange. Diese wird anschließend an den LED-Streifen weitergegeben als visuelle Warnung, dass eine Person den Raum verlassen hat, aber die Personenanzahl gleichgeblieben ist, was nur passiert, wenn die Personenanzahl auf 0 stand und trotzdem eine Person den Raum verlässt.

Damit im Anschluss wieder die Anzahl der sich im Raum befindlichen Personen angezeigt wird, wird mit einer Verzögerung von 5s durch einen Delay-Knoten wieder ein Signal an den Knoten zum „Reseten“ der LEDs und an den Knoten, der nach 1ms Verzögerung das Auslesen der Personenanzahl und damit das Anzeigen dieser veranlasst, gesendet.

3.6.2 Flow RFID-Scanner zu LED

Auch dieser Flow erhält ein Signal durch einen MQTT-Knoten, aber über das Topic „tele/esp32-rfid/SENSOR“, welches von dem Esp32 der mit dem RFID-Scanner verbunden ist, befüllt wird. Zunächst folgt ein Auslesen des Wertes aus dem JSON-Objekt, dass durch den RFID-Scanner versendet wurde. Diese UID wird in einen Array an nullter Stelle gespeichert und anschließend nicht mehr im Payload, sondern in Params weitergegeben. Der Array sowie die weitergabe in msg.params wird vom nächsten Knoten benötigt. Dieser ist ein Postgres-Knoten, der mit der entsprechenden Datenbank verknüpft ist. Mit der SQL-Abfrage „SELECT name FROM rfid_users

Where uid = $1 AND active = true;“ wird hier abgefragt, welchem Namen die UID (hier durch $1 aus den Params eingefügt) entspricht. Die Rückgabe erfolgt in einem Array.

In einem Switch-Knoten wird überprüft, ob der Rückgabewert überhaupt ein Array ist oder null, undefiniert oder etwas anderes.

Wenn es sich um einen Array handelt, wird innerhalb eines Funktionsknotens die Länge des Arrays überprüft. Wenn sie 0 sein sollte, wird im Payload null weitergegeben. Falls nicht, wird der Name aus dem Array ausgelesen und als String im Payload weitergegeben. In einem weiteren Switch-Knoten wird nun überprüft, ob der Payload ein leerer String ist, ob er null ist oder etwas anderes als ein string.

Falls er ein String sein sollte, der nicht leer ist, wird in einem Change-Knoten der Payload auf „00FF00“ gesetzt, welches der Farbe Grün entspricht.

Falls in einem der vorherigen Switch-Knoten der Rückgabewert nicht den Vorgaben entsprochen hat, wird der Payload über einen Change-Knoten stattdessen auf „FF0000“ (Rot) gesetzt.
Der Flow wird nach diesen beiden Change-Knoten wieder zusammengeführt in einen weiteren Change-Knoten, der das Topic auf „cmnd/esp32-led/Color“ setzt.

Hier wird der Flow wieder geteilt und einmal in den Change-Knoten des anderen Flows auf den das „Reset“ der LEDs folgt und einen Delay-Knoten mit 1ms Verzögerung. Dieser lässt anschließend den gesamten LED-Streifen über ein MQTT-Signal in der vorher festgelegten Farbe leuchten lässt und nach weiteren 5 Sekunden Verzögerung „Resetet“ er die LEDs wieder und lässt sie die vorher gespeicherte Personenazahl wieder anzeigen. Dies passiert wieder über die entsprechenden Knoten des Flows von Kamera zu LEDs.

3.6.3 Testfunktionen

Jeder der vorherigen Flows hat Inject-Knoten, mit deren Hilfe man die Eingabe eines MQTT-Signals simulieren kann. Aber es gibt auch Testfunktionen, die tatsächlich nur die Kamera oder den RFID-Scanner simulieren und über MQTT die Signale an die entsprechenden Flows schicken. Hier kann eine zufällige Personenzahl eingespeist werden, aber auch eine richtige und eine falsche RFID-UID sowie zufällig ausgewählt, ob die UID richtig oder falsch sein soll.

Des weiteren gibt es gibt es MQTT-Knoten, die die Signale, die an die LEDs gesendet werden, empfangen und an Debug-Knoten weitergeben, um das Versenden dieser zu überprüfen.
Zuletzt haben wir noch einen Knoten, mit dem man manuell alle LEDs ausschalten kann.

3.7 YOLO-Kamera

YOLO ist ein bereits trainiertes Objekterkennungs-KI-Modell von Ultralytics. Für unser Projekt benutzen wir das Modell YOLO26n. Dieses und weitere können direkt über die Homepage von Ultralytics heruntergeladen werden.

Mit dem Modell alleine kommen wir allerdings nicht weit. Für unseren Raumzähler benötigen wir weitere Komponenten:
- Python 3.8+
- Pytorch 1.8+
- Eine Kamera
- YOLO 11+ Modell

Als Erstes installieren wir Python und Pytorch. Diese benötigen wir zum schreiben unseres Codes und initialisieren des YOLO26n.pt Modells (die Dateinamenserweiterung .pt steht für Pytorch).

Der Nächste Schritt ist das schreiben des Raumzähler selbst, hierfür hat Ultralytics bereits eine Bibliothek für uns bereitgestellt.

1. Zwei Zeilen bilden den Grundstein des Programms, hiermit wird das trainierte Modell geladen:
from ultralytics import YOLO
model = YOLO("yolo26n.pt")

2. Zusätzlich benötigen wir weitere Bibliotheken:
import cv2
import ssl
import math
import paho.mqtt.client as mqtt

- cv2 wird für die Kameraanbindung, die Bildverarbeitung und die Anzeige des Videofensters verwendet
- ssl wird für die verschlüsselte TLS-Verbindung zum MQTT-Broker benötigt
- math wird für mathematische Berechnungen verwendet, zum Beispiel für Abstände zu einer Linie
- paho.mqtt.client dient zum Senden von Nachrichten an einen MQTT-Broker

3. Im nächsten Schritt werden die grundlegenden Einstellungen für das Programm festgelegt:
MODEL_PATH = r"C:\Cam\yolo26n.pt"
SOURCE = 0
LINE_P1 = (300, 400)
LINE_P2 = (200, 100)
LINE_BUFFER = 30
MIN_MOVE_PIXELS = 5

Model_path ist der Speicherort des Modells.
Source ist unsere Kamera (z.B 0=Webcam meines Laptops, 1=Externe Kamera).
Line zeichnet eine Linie durch die Kamera welche dies in einzelne Bereiche abtrennt um das betreten des Raumes zu erkennen.
Line_buffer gibt der linie einen „puffer“ um die eine stabile Erkennung zu ermöglichen.
Min_move_pixels ist die minimale Distanz um Bewegung über die Linie als „Eintreten“ zu erkennen.

4. Als nächstes benötigen wir Variablen um Informationen zwischen einzelnen Kamera-Bildern zu speichern:
Room_count = 0
last_positions = {}

person_states = {}
crossed_ids = set()

Room_count ist unser Raumzähler der ebenfalls per MQTT übertragen wird.
last_position = letzter Ort der erkannten Person.
person_states = in welchen Teil des Bildes er sich befindet.
crossed_ids = verhindert mehrfacherkennung einer Person.

5: Anschließend wird das YOLO-Modell geladen und die Kamera startet:
model = YOLO(MODEL_PATH)

Mit dieser Zeile wird das trainierte Modell geladen, damit Personen im Kamerabild erkannt werden können.

6: Danach wird die Verbindung zum MQTT-Broker aufgebaut:
mqtt_client = mqtt.Client(callback_api_version=mqtt.CallbackAPIVersion.VERSION2)
mqtt_client.username_pw_set("Benutzername", "Passwort")
mqtt_client.tls_set(tls_version=ssl.PROTOCOL_TLS_CLIENT)
mqtt_client.connect(„Server“)
mqtt_client.loop_start()

Hier wird der MQTT-Client erstellt und mit den Zugangsdaten verbunden.
Durch tls_set() wird die Verbindung verschlüsselt aufgebaut, damit die übertragenen Daten geschützt sind.
loop_start() startet im Hintergrund den Nachrichtenversand und Empfang.

7. Im Hauptteil wird jedes Kamerabild analysiert:
for result in model.track(...):

Dabei erkennt das Modell Personen und verfolgt sie über mehrere Bilder hinweg.
Aus den erkannten Boxen wird der Mittelpunkt jeder Person berechnet, damit geprüft werden kann, auf welcher Seite der Linie sie sich befindet.

8. Wenn eine Person die Linie überquert, wird der Zähler angepasst:

```
if current_zone == "right":
    Room_count -= 1
else:
    Room_count += 1
```

Wechselt eine Person von einer Seite auf die andere, erkennt das Programm daraus einen Eintritt oder Austritt. Der Raumzähler wird anschließend erhöht oder verringert und per MQTT gesendet.

9. Zum Schluss wird das Bild angezeigt und das Programm beendet sich mit q:

```
cv2.imshow("YOLO Room Counter", frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
   break
```

So kann man das Ergebnis zum Testen direkt live sehen. Nach dem Beenden werden alle Fenster geschlossen und die Verbindung zum MQTT-Broker getrennt.

4. Fazit

Im Rahmen des Projekts konnte ein funktionsfähiges IoT-basiertes Zugangskontrollsystem entwickelt und erfolgreich umgesetzt werden. Dabei wurden verschiedene Hardware- und Softwarekomponenten miteinander kombiniert und erfolgreich in ein gemeinsames Gesamtsystem integriert.

Neben der technischen Umsetzung standen insbesondere die praktische Anwendung moderner IoT-Technologien, die Absicherung der Infrastruktur sowie die Zusammenarbeit und Koordination innerhalb des Teams im Mittelpunkt des Projekts.

Aufgetretene Probleme und Herausforderungen konnten im Projektverlauf auf individuellen sowie gemeinsamen Ebenen gelöst bzw. überwunden werden. Das Projekt unterlag auch einem dynamischen Wandel und bot mehrere Wege ans Ziel zu kommen. Letztlich könnte man mit mehr Zeit und verhältnismäßigem Aufwand auch gut Erweiterungen implementieren bzw. entwickeln.

4.1 Projektergebnis

Die einzelnen Komponenten wie RFID-Sensorik, ESP32-Mikrocontroller, MQTT-Kommunikation, Node-RED sowie die PostgreSQL-Datenbank konnten erfolgreich miteinander verbunden und in ein gemeinsames Gesamtsystem integriert werden.

Die Überprüfung von RFID-basierten Zugangsberechtigungen funktionierte im Rahmen der Testläufe zuverlässig. Autorisierte RFID-Chips wurden korrekt erkannt und verarbeitet, während unberechtigte Zugriffe entsprechend abgelehnt wurden. Die visuelle Darstellung über das LED-Band ermöglichte dabei eine direkte Rückmeldung des Systemstatus.

Zusätzlich konnte die YOLO-basierte Personenerkennung erfolgreich integriert werden. Personenbewegungen innerhalb des überwachten Bereichs wurden erkannt und zur Darstellung der aktuellen Personenzahl genutzt.

Neben der eigentlichen Funktionalität konnte ebenfalls eine abgesicherte Backend-Infrastruktur, auch mit Blick auf Sicherheits- und Verwaltungsaspekte, umgesetzt werden.

4.2 Probleme und Lösungen

Im Verlauf des Projekts traten verschiedene technische sowie organisatorische Herausforderungen auf. Insbesondere die erstmalige Einrichtung und Absicherung der zentralen Server-Infrastruktur erforderte eine intensive Einarbeitung in Themen wie Linux-Administration, Docker, Reverse Proxy und HTTPS-Konfiguration.

Ebenfalls sollte sich die initiale Ersteinrichtung des RFID-ESPs mit dem Flashen einer Sensor-ansteuerbaren tasmota-Version als besondere Herausforderung darstellen. Hier beanspruchten sehr viele, mühsame und teils auch komplexe Versuche den anfänglichen Projektverlauf. Dies kostete

nicht nur Mühen, sondern auch teils wertvolle Zeit. Aber auch hier konnte sich schrittweise an eine entsprechende Lösung herangetastet werden.

Auch die Kommunikation zwischen den einzelnen Komponenten musste schrittweise getestet und angepasst werden. Dabei war insbesondere die Abstimmung zwischen MQTT-Kommunikation, Node-RED-Flows sowie der Datenbankanbindung von Bedeutung. Durch schrittweise Testläufe konnten auftretende Probleme identifiziert und behoben werden. In Node-RED gab es so beispielsweise zur Beginn die Probleme, dass die MQTT-Knoten nichts empfangen und versenden konnten. Dies muss durch einen Schreibfehler passiert sein, da sie bei erneutem Aufsetzen funktionierten.

Weiter gab es das Problem, des Zusammenlegens der LED-Rückgaben. Dadurch dass wir beide LED-Rückgaben auf einen LED-Streifen leiteten, musste die Personenanzahl zwischengespeichert und wieder abgerufen werden, damit nach anzeigen, ob die UID fehlerhaft oder richtig war wieder die Personenanzahl angezeigt werden konnte. Dies löste sich durch die Befehle flow.set() und flow.get() die in Funktions-Knoten eingebettet werden können und mit denen man Werte zwischenspeichern und wieder abrufen kann.

Der Postgres-Knoten war eine weitere Herausforderung, da dieser nicht intuitiv zu bedienen war. Nach Versuchen mit mehreren verschiedenen Postgres-Knoten und Recherchen zum Bedienen dergleichen, war es uns möglich eine Anleitung für einen zu finden und ihn der Anleitung und unseren Wünschen entsprechend einzubauen.

Eine weitere Herausforderung bestand in der Integration der unterschiedlichen Hardware- und Softwarekomponenten innerhalb eines gemeinsamen Systems. Da mehrere Teammitglieder parallel an verschiedenen Teilbereichen arbeiteten, war eine kontinuierliche Abstimmung innerhalb des Teams notwendig.

Durch die modulare Architektur des Systems sowie die schrittweise Integration der einzelnen Komponenten konnten die auftretenden Probleme jedoch erfolgreich gelöst werden. Ein Nachteil ist im Ergebnis sicherlich der Zeitfaktor gewesen. Auch viel private Zeit und Mühen wurden investiert und nicht alle ursprünglich geplanten Punkte umgesetzt werden. Ein klarer Vorteil der entsprechend im Projekt zu findenden Komplexität ist jedoch auch sicher der entstandene Lerneffekt sowie eine spannende Teamarbeit.

4.3 Ausblick

Das entwickelte System bietet verschiedene Möglichkeiten zur zukünftigen Erweiterung. So könnte beispielsweise eine detailliertere Benutzerverwaltung implementiert werden, um Berechtigungen flexibler verwalten zu können. Auch könnte man beim Thema RFID ansetzen und sich hier vom reinen UID-Konzept technisch lösen.

Darüber hinaus wäre eine Erweiterung um zusätzliche Sensorik oder weitere Zugangspunkte möglich. Auch die Integration zusätzlicher Visualisierungs- und Monitoring-Lösungen könnte zukünftig umgesetzt werden (z.B. Einsatz von Grafana, weitere DB (vgl. InfluxDB) etc.).

Im Bereich der Personenerkennung könnten weitere Funktionen ergänzt werden, beispielsweise eine genauere Auswertung von Bewegungsdaten oder zusätzliche Analysefunktionen (z.B. individuelle Personenerkennung).

Durch die modulare Architektur sowie die containerisierte Backend-Infrastruktur lässt sich das System grundsätzlich flexibel erweitern und auch an zukünftige Anforderungen anpassen.