Hochschule für Angewandte Wissenschaften Hamburg

Fakultät Life Sciences

Entwicklung eines Asset-Tracking-Systems mit automatisierter Zubehörzuordnung für medizinische Einrichtungen

Bachelorarbeit

im Studiengang Medizintechnik

vorgelegt von

Jan Linus Norden

[redacted]

In Hamburg

am 07.07.2024

1. Gutachter: Prof. Dr. Udo van Stevendaal (HAW Hamburg)

2. Gutachter: Prof. Dr. Thomas Schiemann (HAW Hamburg)

Danksagung

Hiermit möchte ich mich bei all denjenigen herzlich bedanken, die mich während der Anfertigung meiner Bachelorarbeit unterstützt und motiviert haben.

Mein besonderer Dank gilt meinen beiden Gutachtern, die sich dazu bereit erklärt haben, mich die letzten Schritte zum Studienabschluss zu begleiten und mir ihre wertvolle Zeit und ihr Wissen zur Verfügung gestellt haben.

Ebenso möchte ich meiner Familie und all meinen Freund:innen danken, die mich durch ihre immerwährende Unterstützung und Motivation ermutigt haben, auch in herausfordernden Zeiten nicht aufzugeben.

Erklärung zur Nutzung geschlechtsneutraler Sprache

In dieser Arbeit wird zur Vereinfachung und besseren Lesbarkeit weitgehend auf geschlechtergerechte Sprache verzichtet. Ungeachtet dessen sind immer alle Geschlechtsidentitäten mit einbezogen. Wo möglich, wurden neutrale Formulierungen verwendet, um allen Personen gerecht zu werden. Ich möchte betonen, dass die Nutzung der ungegenderten Sprache keine Exklusion darstellen soll, sondern lediglich der Lesefreundlichkeit dient.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel „Entwicklung eines Asset-Tracking-Systems mit automatisierter Zubehörzuordnung für medizinische Einrichtungen“ selbstständig und ohne unerlaubte fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

X [redacted]

Jan Linus Norden

Hamburg, 04.07.2024

Inhalt

Abkürzungsverzeichnis

API ..... *Application Programming Interface*

ATS ..... *Asset-Tracking-System*

BLE ..... *Bluetooth Low Energy*

EMV ..... E*lektromagnetische Verträglichkeit*

Espressif IDF ..... *Espressif IoT Development Framework*

F K ..... *Foreign Key*

GPS ..... *Global Positioning System*

JSON ..... *JavaScript Object Notation*

LoRaWAN ..... *Long Range Wide Area Network*

MAC-Adresse ..... *Media-Access-Control-Adresse*

MDR ..... *Medical Device Regulation*

MP ..... *Medizinprodukte*

MPDG ..... *Medizinprodukte Durchführungsgesetz*

MQTT-Broker ..... *Message Queuing Telemetry Transport*

P K ..... *Primary-Key*

REST ..... *Representational State Transfer*

RFID ..... *Radio Frequency Identification*

RTLS ..... *Real-Time-Location-System*

UART ..... *Universal Asynchronous Receiver-Transmitter*

UDI ..... *Unique Device Identifier*

UUID ..... *Universally Unique Identifier*

UWB ..... *Ultra Wide-Band*

VSCode ..... *Visual Studio Code*

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Aktuelle Rechtsnormen, insbesondere die Medizinprodukte Betreiberverordnung (MPBetreibV), stellen begründet hohe Sicherheitsanforderungen an die Betreiber von Medizinprodukt (MP), in der Hauptsache um Gefährdungen von Patienten zu vermeiden. Besonders Sicherheitsvorkehrungen, wie die regelmäßige Prüfung und Wartung der MP, ein korrekter Umgang mit Vorkommnissen sowie die Durchführung von Produktrückrufen, sind von hoher Relevanz.

Für die Erfüllung dieser Anforderungen benötigen medizinische Einrichtungen grundlegende Daten[1] der betriebenen MP. Diese werden bei Inbetriebnahme meist manuell in eine Verwaltungssoftware eingepflegt. In medizinischen Einrichtungen ab einer gewissen Größe stellt der aktuelle Einsatzort eines MPs einen der wichtigsten Datenpunkte dar. Er wird benötigt, um alle erforderlichen Arbeiten an den MP zeitnah zu ermöglichen, womit er für einen normgerechten und sicheren Betrieb unerlässlich ist.

Der Arbeitsalltag in medizinischen Einrichtungen ist oft von immenser Arbeitsdichte und hohem Arbeitstempo geprägt. Die medizinische Behandlung der Patienten steht im Vordergrund, organisatorische und administrative Belange rücken real immer wieder in den Hintergrund [1]. Langjährige Erfahrung in einer medizintechnischen Abteilung zeigt, dass dies auch zu einer unzureichenden Überwachung der Migration von MP und damit falschen Standortinformationen führt.

Sind MP schwer oder nicht auffindbar, kostet die Suche nach ihnen viel Zeit, ggf. fehlen sie für den aktuellen Einsatz am Patienten [2]. Überdies können aufgrund technischer Gegebenheiten erhebliche Komplikationen entstehen. Werden beispielsweise Geräte weiter am Patienten eingesetzt, die wegen des falsch hinterlegten Standorts von der medizintechnischen Abteilung nicht gefunden und vorschriftsmäßig geprüft und gewartet werden konnten, könnte daraus eine akute Patientengefährdung resultieren und der Betreiber würde sich strafbar machen.

Die nicht gegebene Aktualität von Daten kann nicht nur die Funktion eines einzelnen Gerätes beeinträchtigen. Wechseln Geräte des gleichen Typs aber anderer Version unbemerkt den Standort, birgt dies weitere Risiken. Maßgeblich ist hier die mögliche Inkompatibilität von Gerät und Zubehör sowie Unterschiede in Funktion und gebotener Einstellung, wie beispielsweise Alarmierungsgrenzen. Insbesondere zeitkritischen Situationen führen dazu, dass die Prüfung, ob das verfügbare Gerät dem Soll entspricht, vom medizinischen Personal nicht durchgängig zuverlässig zu leisten ist. Der Aspekt einer fehlerhafte Datenhaltung trägt auch in diesem Punkt ein hohes, möglicherweise kostenintensiven Risiko für Betreiber und Patienten in sich.

Um den beschriebenen Problemen entgegenzuwirken sowie die gesetzlichen Anforderungen besser zu erfüllen, setzen einige medizinische Einrichtungen sogenannte Asset-Tracking-Systeme (ATS) ein. Diese

[1] Wie z.B. Standort, Seriennummer, Chargennummer, Gerätetyp, Hersteller, Unique Device Identifier (UDI) etc.

ermöglichen es, MP und andere Objekte, die mit der entsprechenden Hardware oder Barcodes ausgestattet wurden, innerhalb der Einrichtungen zu orten. Werden die so gewonnenen Standortinformationen in bestehende Verwaltungssysteme eingespeist, wird der normgerechte Betrieb aufgrund der Aktualität des Gerätestandorts deutlich erleichtert [3].

Keines der derzeit am Markt verfügbaren ATS bietet allerdings die Erfassung von MP-Kombinationen an, wie sie eine Zubehörnutzung darstellt. Der Standort des Zubehörs könnte mit einigen der Systeme zwar verfolgt werden, eine Neuzuordnung bei Gerätewechsel ist jedoch nicht möglich. Bei der Erstausgabe eines Zubehörartikels wird das zugehörige Gerät zugeordnet, die weitere Migration des Zubehörs wird nicht verfolgt. MP, wie Hämodynamikmodule, EKG-Ableitungskabel oder SpO2-Sensoren, wechseln im klinischen Alltag ständig ihren Standort und werden damit regelmäßig an Geräten verwendet, denen sie in der Geräteverwaltungssoftware der Einrichtung nicht zugeordnet sind. Damit werden die aus der MPBetreibV abgeleiteten Anforderungen an die notwendigen Sicherheitsmaßnahmen vom Betreiber nicht erfüllt. In Bezug auf marktverfügbare ATS ist ergänzend darauf hinzuweisen, dass viele der Hersteller nicht-standardisierte Komponenten und proprietäre Funkverbindungen nutzen. Keiner der untersuchten Hersteller legt die Architekturen der ATS offen. Die Lösungen sind im Ergebnis teils kostenintensiv sowie sicherheitstechnisch schlecht bis nicht einschätzbar. [2, 4-17]

Ziel dieser Arbeit ist die Entwicklung eines innovativen ATS, das eine rechtsnormenkonforme Überwachung und Instandhaltung[2] von MP vereinfacht und damit eine Minimierung der Gefährdung von Patienten bewirken kann. Das Produkt wird im Kontext bereits existierender Asset Tracking Lösungen entwickelt. Mit der genauen Erfassung von MP-Kombinationen, der Registrierung der verbleibenden Sensor-Batterieladung und der offenen Softwarearchitektur werden spezifische Marktlücken adressiert. Hierfür soll ein grundlegendes Produktkonzept erstellt und mit verschiedenen Soft- und Hardwarekomponenten umgesetzt werden. Zum Abschluss der Bearbeitung soll ein funktionsfähiges Prototyp-System erstellt worden sein, welches einen Großteil der gestellten Anforderungen erfüllt und in der vorhandenen Form bereits begrenzt einsatzfähig ist.

[2] Mehr zu Rechtsnormen siehe Anhang A1

2. Anforderungsanalyse und Projektrahmen

Das im Rahmen dieser Arbeit zu entwickelnde Produkt wird mithilfe eines Lasten- und Pflichtenheftes umgesetzt. Dies ermöglicht eine organisierte und detaillierte Darstellung der Anforderungen, die an das Produkt gestellt werden. Alle folgenden Projektabschnitte verlaufen linear und basieren jeweils auf den vorangegangenen Ergebnissen. Anwendung findet das sog. Wasserfallmodell, da es sich sehr gut dafür eignet, einen festen Zeitplan zu erstellen und einzuhalten [18, 19]. Das Modell wird insofern erweitert, als dass eine prozessbegleitende Überprüfung der Abschnitte und ein eventuelles Zurückspringen und Korrigieren ebenfalls vorgesehen sind.

Das aktuelle Kapitel zielt auf die Erstellung des Lastenheftes ab. Dafür werden der derzeitige Stand der Technik in medizinischen Einrichtungen evaluiert und die gegenwärtigen am Markt verfügbaren Asset Tracking Produkte analysiert. Vor dem Hintergrund der gewonnen Ergebnisse sowie eines idealisierten Soll-Zustands der Nutzung von ATS erfolgt eine Bewertung der Produktidee mit anschließender Erstellung des Lastenheftes.

2.1. Ist-Zustand in medizinischen Einrichtungen

Der Ist-Zustand wird innerhalb des Projektrahmens beschrieben. Er basiert auf recherchierten Daten, ermittelten Informationen und ist durch Praxiserfahrungen gestützt.

In ihrer Veröffentlichung mit dem Titel „Best Care at Lower Cost“ der National Academy of Medicine beschreiben die Autoren die Ineffizienz des US-amerikanischen Gesundheitssystems. Eine repräsentative Datenanalyse der Ausgaben medizinischer Einrichtungen im Jahr 2009 ergab, dass rund 30 Prozent dieser Ausgaben, d. h. 750 Milliarden US-Dollar, durch ungerechtfertigte Behandlungen, Betrug und vor allem durch erhöhte Verwaltungskosten verursacht wurden, die hätten vermieden werden können. Die Autoren kommen zu dem Schluss, dass diese Problematiken vornehmlich durch technische Fortschritte schnell und effizient eingedämmt werden könnten. [20] Ein Vergleich des Gesundheitssystems und des Medizinproduktemarkts der USA und westeuropäischer Länder, einschließlich Deutschlands, legt nahe, dass die gewonnenen Erkenntnisse hierzulande ebenfalls als relevant einzuschätzen sind.

Seit der o. g. Datenerhebung haben sich etliche neue Technologien und Produkte in medizinischen Einrichtungen etabliert, darunter auch solche, die die administrative Arbeit vereinfachen. Ein gängiges Beispiel ist ein automatisiertes Patientendaten-Managementsystem mit digitaler Patientenakte, für mittlere bis große Einrichtungen inzwischen ein Standard [21-23]. Trotz dieser Entwicklungen lassen aktuelle Veröffentlichungen vermuten, dass ATS bisher noch selten eingesetzt werden[2, 24].

Eine Studie, die sich mit den Herausforderungen der Digitalisierung medizinischer Dienste in der Europäischen Union, insbesondere im Kontext der COVID-19-Pandemie befasst, kommt unter anderem zu ähnlichen Schlussfolgerungen, wie die Autoren der o. g. Veröffentlichung von 2009. Die

Gesundheitssysteme haben erhebliches unausgeschöpftes Potential Kosten zu sparen und die Versorgung der Patienten zu verbessern. Laut Alexandra-Mădălina Țăran et al. muss „[…] die Digitalisierung der medizinischen Dienstleistungen ein vorrangiges Ziel aller europäischen Staaten sein, und die Gesundheitspolitik und die Gesundheitsausgaben müssen diesen Prozess langfristig unterstützen.“ [25]. Dies wird durch persönliche Erfahrungen und Schilderungen aus der Praxis in verschiedenen medizintechnischen Abteilungen Hamburger Krankenhäusern bestätigt.

Um den aktuellen technischen Stand deutscher Krankenhäuser im Kontext des Einsatzes von ATS sachlich fundierter einordnen zu können, wurde eine Online-Befragung durchgeführt. Diese richtete sich an 1158 allgemeinmedizinischen sowie spezialisierten Einrichtungen unterschiedlicher Größe und geografischer Lage. Der Fragebogen umfasste 21 Single- und Multiple-Choice Fragen, meist mit Möglichkeit der offenen Antwort, die Daten zu der vorhandenen und geplanten Nutzung und Wahrnehmung von ATS abfragten. Ein Liste aller gestellten Fragen und Umfrageergebnisse befindet sich im Anhang unter A2.

Von den angeschriebenen Einrichtungen haben 33 den Fragebogen vollständig ausgefüllt. Lediglich 2 von diesen nutzen ATS. Von den 31 Einrichtungen, welche keines nutzen, planen 7 in naher Zukunft eines anzuschaffen. Weit über die Hälfte der antwortenden Einrichtungen, die weder ein ATS nutzen noch dessen Nutzung planen, nehmen die rein manuelle Pflege von Gerätestandorten und Zubehörzuordnung als ein Problem wahr. Nahezu dreiviertel würden sich den Einsatz von ATS in ihrer Arbeitsstätte wünschen.

Die vorhandenen Umfrageergebnisse können aufgrund der geringen Antwortrate, externer Einflussfaktoren und möglicher unbekannter Korrelationen nicht als wissenschaftlich repräsentativ für die Gesamtheit der deutschen Gesundheitseinrichtungen interpretiert werden. Die unterschiedlichen Spezialisierungen und Größen sowie die geographische Verteilung der antwortenden Institutionen lassen jedoch annehmen, dass die Ergebnisse dennoch eine gewisse Aussagekraft über den Stand der Nutzung von ATS in deutschen Gesundheitseinrichtungen beinhalten. Die Ergebnisse stützen die Annahme, dass ein großer Teil der medizinischen Einrichtungen Deutschlands einen relevanten Bedarf am vermehrten Einsatz digitaler Technik besitzt, wie sie ein ATS darstellt.

Die technischen Entwicklungen in anderen Bereichen, wie die Nutzung einer digitalen Patientenakte, lassen darauf schließen, dass der Grund für die geringe Verbreitung von ATS nicht auf mangelnde Technologieoffenheit zurückzuführen ist. Diese Vermutung deckt sich mit den aus der Umfrage gewonnenen Daten. Den Antworten des befragten medizintechnischen Personals zufolge sind die auschlaggebenden Gründe, die gegen eine Implementierung solcher Systeme sprechen, mehrheitlich die Anschaffungs- und Betriebskosten sowie der Aufwand der Installation und Wartung.

Aus den o. g. Studien in Verbindung mit der durchgeführten Umfrage ist zu erkennen, dass es in den Gesundheitssystemen Deutschlands und anderer EU-Mitgliedsstaaten sowie den USA einen erheblichen Bedarf für die weitere Digitalisierung und eine damit einhergehende Vereinfachung von Prozessen gibt, wie sie z. B. durch den Einsatz von ATS erzielt werden können. Die Kosten für solche Produkte sind dabei offensichtlich ein entscheidender Faktor, von dem die Verbreitung solch neuer Technologien abhängig ist.

2.2. Soll-Zustand der Nutzung von ATS in medizinischen Einrichtungen

Zusätzlich zu den objektiven Anforderungen fließen auch subjektive Beweggründe in die Ziele der geplanten Entwicklung mit ein, die auf individuellen Werten und Idealen sowie den genannten Erfahrungen basieren. Diese Ziele werden unter 2.2.1. „Abgrenzung der Arbeit“ genauer formuliert. Vorab werden die Vorteile der generellen Nutzung von ATS reflektiert.

Aus einer effizienten Einbindung von ATS in den Alltag medizinischer Einrichtungen sollten direkte Vorteile durch die Aktualität der Standort- sowie Zubehörnutzungsdaten resultieren. Die administrative Arbeit in medizintechnischen Abteilungen sollte merkbar vereinfacht werden, wodurch die Ressource Zeit effektiver eingesetzt werden könnte. Die Suche nach MP würde reduziert, die Arbeitsabläufe würden optimiert und deren Zeitaufwand insgesamt verringert.

Als weiterer Effekt ist der Einfluss auf das Beschaffungsmanagement und die Bestandsführung zu nennen. Gesundheitseinrichtungen beschaffen üblicherweise 10 – 30% mehr Ausrüstung, als tatsächlich benötigt wird, um den unbekannten Standort von MP im Arbeitsalltag auszugleichen [26, 27]. Diese Überbeschaffung könnte durch eine verbesserte Auffindbarkeit deutlich reduziert werden.

Mittelfristig wäre mit einer Kosten- und Zeitaufwandssenkung, einer verbesserten Patientenversorgung und -zufriedenheit sowie einer höheren Arbeitszufriedenheit bei Mitarbeitenden zu rechnen. Die Aussicht auf eine Entlastung bei vorhandenem Arbeitskräftemangel ist ebenfalls positiv zu bewerten. Nicht zuletzt würden die Risiken in Hinblick auf ein rechtssicheres Arbeiten erfolgreich minimiert werden können.

2.3. Abgrenzung der Arbeit

Die beschriebene Vielzahl an möglichen Funktionen, die mit ATS umgesetzt werden könnten, machen eine analytische Gliederung sowie Abgrenzung der in dieser Arbeit umzusetzenden Funktionen erforderlich. Für die Anforderungen an ATS wurden folgende Kernbereiche identifiziert:

- Qualitätssicherung
- Kostenoptimierung
- Umsetzung geltenden Rechts

Abb. 1 gliedert die bestehenden Anforderungen auf zwei Ebenen und ordnet Teilziele zu.

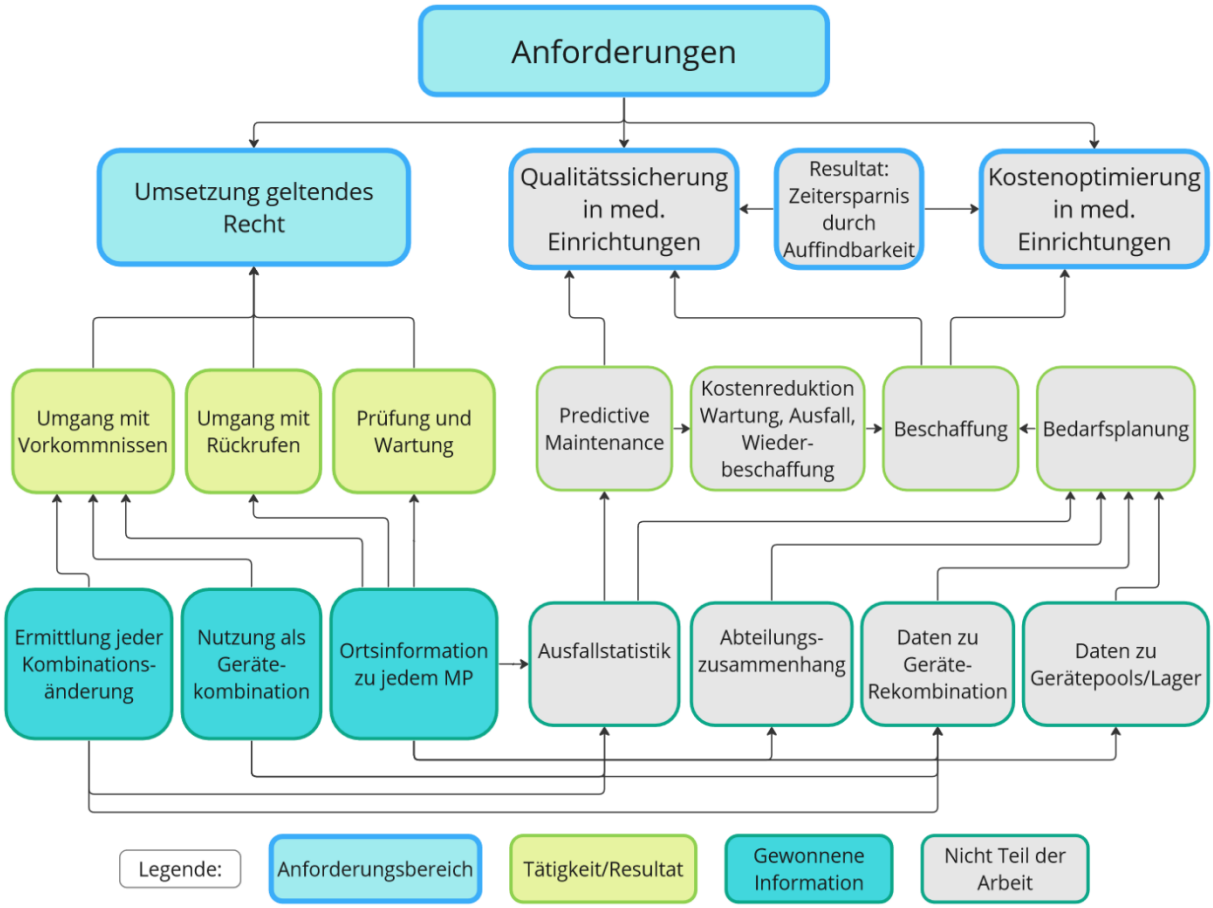


Abb. 1: Anforderungsbereiche ATS

Hypothetisch könnten die erfassten Daten für zusätzliche Bereiche herangezogen werden (hier grau hinterlegt). Die dafür erforderlichen Analysen und Interpretationen würden jedoch über den Projektrahmen dieser Arbeit hinausgehen.

Das in dieser Arbeit umzusetzende System begrenzt sich auf die Erfassung der in der Grafik türkis hinterlegten und als „Gewonnene Information“ gekennzeichnet Daten. Mit der Erfassung dieser Daten sollen grundlegende Informationen über die Ortung und zusätzlich die kombinatorische Nutzung der betriebenen MP gewonnen werden.

Damit sollen die aus den geltenden Rechtsnormen abgeleiteten Anforderungen an medizinische Einrichtungen ausdrücklich besser erfüllt werden können.

2.4 Bewertung der Produktidee

Die zugrundeliegenden Produktidee basiert auf dem Produktkonzept von auf dem Markt vorhandenen, für den Einsatz in medizinischen Einrichtungen konzipierten ATS. Sie fokussiert auf eine gezielte Optimierung und Ergänzung von Funktionen innerhalb des oben definierten Anforderungsbereichs.

Eine Vielzahl an Herstellern haben bereits Produkte dieser Art entwickelt und damit einhergehend Analysen und Bewertungen durchgeführt. Die Menge an verschiedenen ATS bietet Rückschlüsse auf einen wachsenden Markt für solche Produkte.

Ein übergreifender Blick auf den aktuellen Stand des Marktes erscheint an dieser Stelle daher passender als eine ausführliche wirtschaftliche Bewertung anhand üblicher Kriterien, wie Entwicklungs- und Herstellungskosten, Wettbewerbsvor- und nachteile, Ressourcen und Qualifikationen oder die Art des Vertriebs.

Zur Bewertung der formulierten Produktidee wird näher auf das Kriterium des Nutzens und die Frage eingegangen, welche innovativen Vorteile bzw. welche Alleinstellungsmerkmale diese besitzt.

2.4.1 Aktueller Stand des Marktes

In der Regel bestehen ATS aus zwei Typen von Hardwarekomponenten, die eine Ortsbestimmung ermöglichen. Dies sind Beacons bzw. Sender, die geortet werden und Hubs bzw. Empfänger, die die Ortung vornehmen. Einige Hersteller bieten zusätzlich spezielle Funktionen an, wie eine Beschleunigungsmessung oder Luftfeuchtigkeitsbestimmung, die mit in den Beacons verbauten Sensoren umgesetzt werden. Der wesentlichste Unterschied innerhalb der existierenden Systeme besteht jedoch in den angewandten Kommunikationstechnologien zur Ortung und Weiterleitung von Informationen. Für die Ortung verwenden Hersteller überwiegend Technologien, wie BLE, LoRaWAN, RFID, UWB oder GPS (siehe Abkürzungsverzeichnis). Für die Nachrichtenweiterleitung wird fast ausschließlich mit lokalen WiFi-Netzwerken gearbeitet.

Einige Produkte zeichnen sich dadurch aus, dass die angebotenen Empfänger/Hubs in der Lage wären, untereinander ein Mesh-Netzwerk aufzubauen, womit sie nur einen Access Point pro Einsatzbereich benötigen. [2, 4-17, 28, 29]

Die jeweils genutzten Kommunikationsmethoden haben eindeutige Vor- und Nachteile in Hinblick auf die Eignung für den Einsatz in verschiedenen medizinischen Bereichen. Die Tab. 1 zeigt einen Vergleich der verschiedenen Technologien.

Tab. 1: Eigenschaften RTLS-Technologien

Technologie	Echtzeit	Genauigkeit	Reichweite	Energieverbrauch	Anforderungen an Infrastruktur	Kosten
UWB	Ja	Hoch	Kurz bis mittel	Mäßig	Komplex	Hoch
Bluetooth	Ja	Niedrig bis hoch	Kurz	Niedrig	Einfach	Niedrig / mittel
WiFi	Ja	Mittel	Mittel bis lang	Mäßig	Einfach	Mittel
RFID	Nein	Niedrig bis hoch	Kurz bis mittel	Niedrig	Mäßig bis hoch	Niedrig
LoRaWAN	Nein	Niedrig bis hoch	Mittel bis lang	Niedrig	Mäßig bis hoch	Mittel
iBeacon	Nein	Niedrig	Kurz	Niedrig	Einfach	Niedrig / mittel

[29]

Die in der Tabelle dargestellten Informationen decken sich mit den Ergebnissen einer Studie von B. Rodrigo et al. [30]

Jede dieser Technologien ist unter den aktuell vertriebenen Systemen vorhanden. Die Hersteller wägen einhergehende Vor- und Nachteile ab und wählen die zu den jeweiligen Produktschwerpunkten passenden Lösungen. Dies zeigt, dass sich die verfügbaren Systeme durch ihre unterschiedliche Hardware stark in Funktionsweise, Kosten, Sicherheit und Wartungsintensität unterscheiden.

Wie eingangs erwähnt, bietet keines der angebotenen ATS die Funktion einer automatisierte Zubehörzuordnung. Abgesehen von der Möglichkeit der Ortung des Zubehörs werden keine Funktionen angeboten, die dazu in der Lage sind, den Aufwand der Datenhaltung für den Betreiber im Hinblick auf Zubehörartikel zu verringern.

Zusätzlich werden, je nach Hersteller, unterschiedliche, oft proprietäre Technologien eingesetzt, hauptsächlich in Bereichen wie der Netzwerktopologie, den verwendeten Sensoren oder der Datenhaltung und Verteilung. Dies limitiert die Möglichkeiten zur Einschätzung sicherheitstechnischer Belange und kann zusätzlich zur Folge haben, dass manche ATS wegen länderspezifischen Vorgaben nicht eingesetzt werden können (z. B. wegen einer Einschränkung der nutzbaren/öffentlichen Frequenzbänder). [2, 4-17]

2.4.2 Alleinstellungsmerkmale der projektierten Lösung

Im direkten Vergleich zu den oben genannten Lösungen, erlaubt der projektierte Lösungsansatz:

- die Nutzung frei am Markt verfügbarer, für elektromagnetische Verträglichkeit (EMV) zertifizierter Standardkomponenten im Bereich der Sensoren
- Open Source Komponenten zur Datenkommunikation und Datenhaltung
- frei verfügbare Bauelemente im Bereich der Hardwareentwicklung
- die Kopplung von MPs durch einfachen Nutzerinput
- verfügbare Quelltexte aller Komponenten zur sicherheitstechnischen Analyse

In Summe ergeben sich Vorteile in allen angestrebten Bereichen, auch auf die aus der eigenen Umfrage ersichtlichen möglichen Sicherheits- und Kostenbedenken.

Der angestrebte Prototyp erlaubt ebenfalls:

- eine offene Schnittstelle zur einfachen Anbindung an bestehende Verwaltungssysteme
- eine Einbindung der Systemkommunikation in das Krankenhaus-WLAN mit minimaler Last

Diese letzten beiden Punkte stellen jedoch keine vollständigen Alleinstellungsmerkmale dar, da sie von manchen Herstellern bereits umgesetzt worden sind.

2.4.3 Ergebnis der Bewertung

Diese Arbeit umfasst eine prototypische Umsetzung der Grundfunktionen eines ATS, das dazu in der Lage sein wird, die o. g. genannten Aspekte zu erfüllen.

Die unter 2.4.1 „Aktueller Stand des Marktes“ geschilderten Eigenschaften der am Markt erwerbbaren ATS zeigen, dass keines dieser Systeme alle geforderten Merkmale der projektierten Lösung gemeinsam erfüllt. Aufgrund der verwendeten Hardware sowie proprietärer Entwicklungsansätze, ist auch ein Anpassen von vorhandenen ATS für die Erfüllung der o. g. Aspekte nicht möglich. Das geplante ATS bietet die Chance, eine sinnvolle Ergänzung zu bestehenden ATS in einem für Außenstehende offen einsehbaren und damit nachvollziehbarem Rahmen umzusetzen. Für Betreiber und Hersteller wird es durch die Open Source Architektur möglich sein das in dieser Arbeit erstellte System weiterzuentwickeln oder Einzelaspekte in bestehende Systeme zu übernehmen.

2.5. Lastenheft der Asset Tracking Prototypisierung

Im Lastenheft wird das Entwicklungsziel der prototypischen Umsetzung präzisiert und damit der zu erfüllende Zweck sowie die funktionalen und nicht funktionalen Anforderungen an das Produkt konkret beschrieben.

Bei der Erstellung wurde sich an der Begriffsdefinition gemäß DIN 69901-5:2009-1 orientiert.

Um eine bessere Übersichtlichkeit zu gewährleisten, werden an dieser Stelle nur die besonders wesentlichen Anforderungen und Inhalte des Lastenhefts exemplarisch aufgeführt und erklärend kommentiert. Das vollständige Dokument befindet sich im Anhang unter A3.

Die einleitende Zweckbestimmung steckt den Projektrahmen ab und erklärt den Begriff ATS. Nachfolgend werden unter dem Punkt der funktionalen Anforderungen die grundlegenden Funktionen skizziert. Dies beinhaltet erste Forderungen an die Funktion und Architektur des Gesamtsystems und stellt damit den Grundpfeiler aller weiteren Anforderungen dar.

Die ersten dieser Anforderungen definieren die Funktionen der Ortung und (Zubehör-) Zuordnung:

1. *Das ATS soll die Ortung von Medizinprodukten (MP) ermöglichen. (siehe auch 2.2.)*[3]
2. *Das ATS soll die Speicherung des detektierten Ortes von MP ermöglichen.*
3. *Das ATS soll die Erkennung einer kombinatorischen Nutzung von MP ermöglichen. (siehe auch 2.3.)*
4. *Das ATS soll die Speicherung der Zuordnung mehrerer detektierter MP zueinander ermöglichen.*
5. *Das ATS soll eine Historie relevanter Datenänderungen zu Verfügung stellen. (siehe auch 2.4.1.)*

Basis sind die oben festgestellten Funktionsweisen gängiger ATS, die durch unterschiedliche Technologien und Konzepte das Orten von Sendern und damit MP ermöglichen. Dabei wird die Besonderheit der Zubehörzuordnung in den Anforderungen 3. und 4. Aufgegriffen und später spezifiziert. Details, wie die Ortungsgenauigkeit und Sendeintervalle, werden ebenfalls in späteren Anforderungen festgelegt.

Weiterhin werden einige essenzielle Anforderungen an die zu erhebenden Daten und Interaktion mit diesen formuliert:

6. *Das ATS soll die Datenweiterleitung innerhalb bestehender lokaler Netzwerke ermöglichen. (siehe auch 2.4.2.)*
7. *Das ATS soll darauf ausgelegt sein mit bestehenden Geräteverwaltungssoftwares zu interagieren und Daten austauschen zu können. (siehe auch 2.4.3.)*

[3] Die Verweise beziehen sich auf weitere im Lastenheft formulierte Anforderungen, die der jeweiligen Spezifizierung dienen.

8. *Es soll eine Webanwendung geben, welche die Interaktion mit den relevanten Daten des ATS ermöglicht. (Siehe auch 2.5.)*

Zuletzt wird die Skalierbarkeit des Systems gefordert, um die Möglichkeit des Einsatzes in Einrichtungen unterschiedlichster Größe sicherzustellen:

9. *Das ATS soll skalierbar sein, um mit einer variablen Anzahl von MP, Bereichen und Stationen umgehen zu können. (siehe auch 2.6.)*

Diese neun Anforderungen definieren den Rahmen des geforderten Funktionsumfangs.

Zusätzlich zu den funktionalen Anforderungen werden nichtfunktionale Anforderungen gestellt. Diese befassen sich mit Aspekten wie der Zuverlässigkeit, Wartbarkeit und Sicherheit der Systemkomponenten und adressieren insbesondere die Qualität des Gesamtsystems:

1. *Die in den medizinischen Einsatzbereichen verwendeten Hardwarekomponenten sollen gegen Umwelteinflüsse geschützt sein, wie sie in dem Einsatzbereich auftreten könnten (Staub, Feuchtigkeit, Desinfektionsmittel etc.)*
2. *Die Betriebsdauer aktiver Komponenten zur Positionserfassung soll ohne Wartung oder Batterietausch minimal 2 Jahre betragen.*
3. *Ein Batterietausch oder ein Aufladen von Komponenten zur Positionserfassung soll schnell, einfach und mit Standardkomponenten (z. B. standardisierten Batteriezellen) möglich sein.*
4. *Im Falle einer Wartung von Komponenten zur Positionserfassung sollen die Schutzklasse (z. B. IP67) sowie alle o. g. Eigenschaften weiterhin erhalten bleiben.*

3. Systemkonzept und Design

Das erstellte Lastenheft bildet die Voraussetzung für das detaillierte Konzipieren und Designen des Gesamtsystems. Am Ende dieses Kapitels soll die Umsetzung des im Lastenheft geforderten Produkts mithilfe der Entwicklung eines Pflichtenhefts möglichst vollständig geplant sein. Die im Lastenheft beschriebenen Anforderungen sind auf Umsetzbarkeit zu prüfen und ggf. zu überarbeiten.

Eine textliche Übernahme aller Details des Pflichtenheftes wäre sehr umfangreich und an dieser Stelle nicht zielführend, weshalb die Herleitungen und Erläuterungen ebenfalls exemplarisch auf die grundlegenden Inhalte begrenzt wird. Das vollständige Pflichtenheft befindet sich im Anhang unter A4.

3.1. Zulassungskriterien und rechtliche Pflichten

Ein entscheidender Aspekt bei der Entwicklung eines ATS für den Gebrauch in medizinischen Einrichtungen ist die Prüfung, ob es als MP zugelassen werden müsste. In diesem Fall wären deutlich strengere Anforderungen zu erfüllen und sehr aufwändige Zulassungsschritte zu durchlaufen, welche aus Rechtsnormen wie der Medical Device Regulation (MDR) und dem Medizinprodukte Durchführungsgesetz (MPDG) hervorgehen. Darüber hinaus muss erörtert werden, ob die Verwendung der Hardwarekomponenten an einem MP eine Veränderung desselben bedeuten und damit eine neue Kombination darstellen würde. Wäre das der Fall, so würde diejenige Person die die Veränderung vorgenommen hat rechtlich als Hersteller gelten.

Die Bewertung erfolgt anhand der Festlegung des Anwendungsbereiches der MDR. Der dafür relevante Auszug aus der Norm ist in Anhang unter A5 zu finden.

Ein Beacon zur Ortsüberwachung eines medizinischen Gerätes oder Zubehörs dient in seiner Funktion nicht einer medizinischen Zweckbestimmung. Nach MDR §1 und MDR §2 Abs. 2, Abs. 10 oder Abs. 11 stellt ein solcher Beacon damit weder ein Medizinprodukt noch ein Zubehör zu einem Medizinprodukt dar.

Bestünde ein Einfluss eines Beacons auf umliegende Geräte, hätte dieses eine direkte Auswirkung auf die Verwendung in medizinischen Einrichtungen. Im Rahmen der MDR wäre es gewollt, die kombinierte Nutzung der Komponenten des ATS mit MP zu bewerten. Ein marktreifes Produkt sollte vor Verkaufsstart ausgiebige Tests durchlaufen haben, in denen Aspekte, wie die EMV, getestet werden. Weitere wichtige Punkte, die vor konkretem Einsatz in Betrieben zu berücksichtigen wären, werden im Lastenheft unter Kap. 3. beschrieben.

3.2. Anforderungen an Hardwarekomponenten

Laut Marktanalyse setzt die Mehrheit der verfügbaren ATS eine Ortung mithilfe von Beacons und Hubs um. Abgesehen von der Verwendung von Barcodes und Scannern ist dieses Vorgehen innerhalb aller betrachteten Technologien nahezu alternativlos. Ausschlaggebend ist die umfassende Einsatzmöglichkeit der Komponenten in einer Vielzahl unterschiedlichster Einrichtungen mit verschiedener Ausrüstung. Dieses Vorgehen gewährleistet die erforderliche Autarkie bei der Datenerhebung und -übertragung am besten. Aus diesem Grund wird das prototypisierte ATS ebenfalls mit Beacons und Hubs umgesetzt.

Betrachtet man die verfügbaren Kommunikationstechnologien zeigt sich, dass es hier Spielraum in der Umsetzung gibt. Jede unter 2.4.1 Aktueller Stand des Marktes“ genannte Technologie bietet die Möglichkeit einer Umsetzung der geforderten Funktionen. In einer Analyse der Kosteneffizienz von Real-Time-Location-Systems (RTLS) und ATS in Gesundheitseinrichtungen kommt D. Banat zu dem Schluss, dass BLE die vielversprechendste Technologie sei, um solch ein System zu realisieren. Sie vereine eine präzise Ortung und geringe Störanfälligkeit mit sehr geringem Akkuverbrauch, was für diese Anwendung ideal sei. Andere gängige Technologien, wie WiFi, LoRaWAN, Infrarot usw. hätten laut den von Banat untersuchten Studien in Punkto Kosten oder Störanfälligkeit erhebliche Nachteile gegenüber BLE. [2] Zusätzlich zu D. Banats Einschätzung zeigen weitere Quellen, dass BLE aufgrund der niedrigen Kosten, langen Batterielebensdauer und ausreichenden Genauigkeit eine bevorzugte Wahl für RTLS- und Asset-Tracking-Anwendungen ist. Die Technologie bietet eine einfache Integration in bestehende Wi-Fi-Netzwerke sowie eine hohe Skalierbarkeit, was sie besonders für dynamische Umgebungen wie Krankenhäuser und industrielle Einrichtungen geeignet macht. [29, 31, 32] Diese Einschätzung deckt sich mit eigenen Erfahrungen, weshalb sich für die Umsetzung mit BLE-Komponenten entschieden wurde.

Ein weiterer Vorteil von BLE ist die breite Verfügbarkeit von Standardkomponenten unterschiedlichster Hersteller, aber identischer Funktionsweise. Betreiber, die das in dieser Arbeit beschriebene ATS einführen wollten, könnten somit weitgehend frei aus den am Markt verfügbaren BLE-Beacons und deren Bauformen wählen.

Um die geforderte Ortung von MP umzusetzen, müssen die Beacons und Hubs bestimmte Daten übertragen können. Alle Hardwarekomponenten müssen eindeutig identifizierbar sein, um eine korrekte Zuordnung in der Datenhaltung zu ermöglichen. Dafür ist die Übertragung ihrer jeweiligen Media-Access-Control-Adresse (MAC-Adresse) notwendig. Um die Ausfallrate so gering wie möglich zu halten, sollte der Batteriestand ebenfalls übermittelt werden.

Eine Untersuchung gängiger BLE-Beacons zeigt, dass die meisten von ihnen ihre Daten im iBeacon-Format, einem von wenigen Standardformaten senden können [33, 34]. iBeacon bezeichnet einen 2013

von Apple Inc. etablierten Funkstandard auf Basis von BLE, der für die Ortung in Innenräumen bestimmt ist [35]. Ein empfangender Hub kann iBeacon-Signale gut von anderen Bluetooth Signalen separieren. Dies ermöglicht ein Filtern nach Beacons mit dieser Signatur, inklusive eines einstellbaren Universally Unique Identifier (UUID), womit die zu verarbeitende Datenmenge erheblich reduziert wird.

Folgende Pflichtenheftanforderungen werden auf Basis dieser Kenntnisse für die geplante Entwicklung aufgegriffen:

- *Das ATS soll die Ortung von BLE-fähigen Sendern ermöglichen.*
- *Die Komponenten des ATS, die die Ortung ermöglichen, sollen aus einem BLE fähigen Sender (folgend Beacon) und einem BLE fähigen Empfänger (folgend Hub) bestehen.*
- *Die Beacons sollen austauschbare Standardkomponenten sein, sodass sie leicht durch einen anderen Typ ersetzt werden könnten.*
- *Die Beacons sollen den iBeacon Standard unterstützen (impliziert einen UUID).*
- *Die Beacons sollen eine eindeutige MAC-Adresse haben.*
- *Die Beacons sollen beim Senden folgende Daten an umliegende Hubs übermitteln: UUID, MAC-Adresse, Batteriestatus, […].*
- *Die für die Hubs verwendete Hardware soll aus austauschbaren Standardkomponenten bestehen, sodass sie leicht durch einen anderen Typ ersetzt werden könnten.*
- *Die Hubs sollen eine eindeutige MAC-Adresse haben.*
- *Die Hubs sollen permanent nach BLE-Geräten suchen.*
- *Die Hubs sollen bei Detektion eines Beacons mit korrekter Signatur die von dem Beacon gesendeten Daten empfangen.*

Es muss eine Entscheidung bezüglich der Datenweiterleitung getroffen werden. Empfängt ein Hub die Daten eines Beacons, müssen diese an nachfolgende Komponenten des ATS gesendet werden. Um die beschriebene Autarkie zu gewährleisten, sollen, für das prototypisierte System, Hubs verwendet werden, die den Aufbau eines WiFi-Meshs ermöglichen. Innerhalb dieses Meshs soll sich ein Hub als Basisknoten dieses Netzes mit einem lokalen Netzwerk/ Access-Point verbinden und so die Nachrichten weiterleiten. Die geforderte Skalierbarkeit als auch die minimale Belastung vorhandener Netzwerke sind dabei zu berücksichtigen.

- *Die Hubs sollen ein Wifi-Mesh untereinander aufbauen.*
- *Die Hubs sollen dazu in der Lage sein lückenlos nach Bluetoothgeräten zu suchen, während das WiFi-Mesh aufrecht erhalten wird.*
- *Die Hubs sollen sich mit einem am Einsatzort verfügbaren Access-Point verbinden können.*
- *Das WiFi-Mesh der Hubs soll eine Tiefe von mindestens 4 Knoten bei mindestens 5 Child-Nodes aufbauen können.*

- *Die Anzahl der mit den lokal verfügbaren WLAN verbundenen Hardwarekomponenten soll in jedem Bereich (z. B. Krankenhausstation) auf max. 2 begrenzt werden*

Weiterhin bedarf es einer Festlegung, anhand welcher Kriterien die Ortsbestimmung und die Erfassung der kombinatorischen Nutzung umgesetzt werden können. Für ersteres bedarf es der Feststellung der Empfangsstärke (RSSI) eines Beacons durch den empfangenden Hub. Wird ein Beacon von mehreren Hubs geortet, so soll er dem Hub mit dem stärksten Empfangssignal zugeordnet werden. Damit würde der Beacon, bis auf Ausnahmen, z.B. wegen aktiver Bewegung, dem räumlich nächsten Hub zugeordnet werden (weitere Details, siehe Kap. 3.3.).

Für die Erfassung der kombinatorischen Nutzung vom MP wird eine Möglichkeit der anwendergesteuerten Zuordnung benötigt. Unter gängigen BLE-Beacons gibt es einige mit einem Knopf ausgestattete Modelle, die den Status des Knopfdrucks zusätzlich zum iBeacon-Datenpaket übermitteln. Ein Knopfdruck bietet eine ideale Möglichkeit der einfachen Nutzerinteraktion, mit der die Erfassung von Kombinationen stattfinden kann. Der Zeitaufwand für das medizinische Personal bliebe minimal, wenn lediglich ein zeitnahes Betätigen zweier Knöpfe nötig wäre, um MP zu paaren.

Aus obigen Punkten resultieren folgende Anforderungen:

- *Die Beacons sollen über einen Knopf verfügen, der sie unabhängig von dem festgelegten Sendeintervall sofort senden lässt.*
- *Die Beacons sollen beim Senden folgende Daten an umliegende Hubs übermitteln: […], Buttonstatus.*
- *Die Hubs sollen die Empfangsstärke (RSSI) zu den jeweiligen Beacons erfassen.*

Diese Anforderungen schaffen eine Grundlage für die Datenerhebung und -weiterleitung. Auf dieser kann nachfolgend die Ausarbeitung der Softwarekomponenten erfolgen, in der die weitere Datenverarbeitung sowie Datenhaltung erörtert werden.

3.3. Anforderungen an Softwarekomponenten

Die korrekte Ortung von Beacons erfordert eine chronologische Abarbeitung aller erstellten Datensätze. Nur so kann gewährleistet werden, dass der ermittelte Standort korrekt ist. Die Zwischenverarbeitung der Beacondaten, welche die Hubs vornehmen, erfolgt aufgrund der geringen Datenmenge und einfacher Verarbeitungsschritte sehr schnell. Gleiches gilt für ein Weiterleiten der Datensätze per WiFi-Mesh zwischen den Hubs und an nachfolgende Komponenten. Zeitliche Differenzen wären bis zu diesem Zeitpunkt gering und nicht signifikant.

Beacons und der iBeacon Standard benötigen und liefern keine Erstellungszeit von gesendeten Nachrichten. Die erforderliche Zuordnung eines Zeitpunktes zu einem Beacon-Signal würde Hub-seitig eine Zeitsynchronisation eines jeden Hubs mit einem Time-Server bedingen, während die Zuordnung eines Zeitpunktes in einer nachfolgenden Informationsverarbeitung wesentlich einfacher an einem zentralen Punkt erfolgen kann.

Wann ein Beacon-Signal eingeht, bestimmt die Asynchronität des Sendens der einzelnen Beacons. Auch wenn für reale Nutzungsbedingungen von einer durchschnittlichen Gleichverteilung ausgegangen werden kann, sollte zur Auslegung der Spitzenlast der Architektur die maximal mögliche Anzahl gleichzeitiger Signale genutzt werden. Um einen Informationsverlust durch Überlast zu vermeiden und die Chronologie in allen weiteren Prozessschritten zu gewährleisten, bietet sich die Nutzung eines Zwischenspeichers in Form einer Message-Queue (MQ) an. Die Hubs können ihre Daten unmittelbar nach dem Empfangen an die MQ senden und die weiterführenden Programme können die Daten aus dieser chronologisch auslesen.

Ein open source Message Queuing Telemetry Transport Broker (MQTT-Broker), wie er unter anderem von der Eclipse Foundation [36] angeboten wird, bietet die Möglichkeit, eine hoch-performante MQ umzusetzen. Die nachfolgenden Softwarekomponenten können alle Daten sequenziell und mit einer der Gleichverteilung von Nachrichten entsprechenden Geschwindigkeit verarbeiten.

Für die weiterführende Datenverarbeitung ist der Entstehungszeitpunkt einer Nachricht relevant (s.o.). Eine MQ verarbeitet Nachrichten sequenziell. Ein Abruf der Informationen muss bei Berücksichtigung erforderlicher Verbindungsparameter nicht zum gleichen Zeitpunkt erfolgen. Im Falle von Verbindungsabbrüchen kann dieser ggf. auch mehrfach oder asynchron geschehen. Entsprechend wird ein Zeitpunkt für eine Nachricht spätestens beim Eintrag in die MQ benötigt.

Der offene Quellcode des o. g. MQTT-Brokers oder auch bereits im Netz verfügbare Erweiterungen erlauben die Anpassung an diese Anforderung. Der Broker soll jedem Datensatz einen Zeitstempel hinzufügen, der die Empfangszeit sekundengenau wiedergibt. Wird die so entstehende Nachricht aus Beacondaten und Zeitstempel als JavaScript Object Notation (JSON) weitergegeben, können die

weiterführenden Softwareelemente jeden Datenpunkt leicht identifizieren, einer genauen Uhrzeit zuordnen und die Weitergabe erfolgt in einem weit verbreiteten Standard-Format.

Ein MQTT-Broker dient primär zum Sammeln und Verteilen von Nachrichten. Er speichert Nachrichten nur unter bestimmten Bedingungen. [37] Für diese Umsetzung ist die Speicherung von Nachrichten für eine persistente Subscription jedoch wesentlich. Die verarbeitende Software muss sich mit einer Kennung am Broker identifizieren und erhält bei einer Neuverbindung alle neuen Nachrichten seit der zuletzt erfolgreichen Übermittlung an ebendiese Kennung.

Dieser Ablauf spiegelt sich in den Anforderungen wider:

- *Auf einem dedizierten Server soll permanent ein MQTT-Broker laufen.*
- *Die Hubs sollen eine Verbindung zum MQTT-Broker aufbauen können.*
- *Die von den Hubs empfangenen Daten sollen chronologisch unmittelbar nach dem Verarbeiten gesendet werden.*
- *In den Daten, die von den Hubs gesendet werden, sollen jeweilig festgestellte RSSI der Beacons enthalten sein.*
- *Die Daten, die von den Hubs gesendet werden, sollen als kommaseparierte Liste mit Name-Value Paaren an den MQTT-Broker gesendet werden.*
- *Der MQTT-Broker soll jeder empfangenen Nachricht, bevor sie in die MQ eingefügt wird, die Empfangszeit als Unix-Zeitstempel der aktuellen Uhrzeit hinzufügen.*
- *Der MQTT-Broker soll die um den Timestamp erweiterten Nachrichten im JSON-Dateiformat bereitstellen.*

Nach dem Einfügen der JSON-Nachrichten in die MQ müssen die in ihnen enthaltenen Daten weiterverarbeitet werden. Benötigt wird ein Programm (Microservice), das die Daten aus der MQ entnimmt, verarbeitet und anschließend an eine Datenbank (DB) zur permanenten Speicherung übergibt.

Eine Auflistung der unterschiedlichen Szenarien von Meldungen aus Beacon- und Hub-Informationen und deren Ablauf verdeutlicht die gestellten Anforderungen an die Verarbeitung. Es gibt:

- Beacons, die von nur einem Hub gemeldet wurden,
- Beacons, die von mehreren Hubs in unterschiedlicher Feldstärke gemeldet wurden,
- Änderungen der Empfangsfeldstärke, sowohl in einem Raum als auch bei Raumwechsel,
- Beacons, die gedrückt wurden (Button),
- Alle $t \leq 60$ Sekunden selbstständige Aktualisierungen eines jeden Beacons.

Aus diesen Meldungen sind Regeln zu folgenden Aspekten abzuleiten:

- Zuordnung eines Beacons zu einem Raum
- Raumwechsel
- Zubehörzuordnung
- Validierung der Zulässigkeit von Zubehörzuordnungen
- Trennung von Zubehör

Die häufigste Information – ein Beacon ist nach wie vor in einem Raum und hat seine Position nicht geändert – muss nicht alle t ≤ 60 Sekunden in der DB aktualisiert werden. Auch andere Datenänderungen sollten nicht bei jeder Änderung in die DB eingetragen werden, um unnötige DB-Anfragen zu vermeiden.

Um die Effizienz der Datenverarbeitung zu erhöhen, können die Datenpakete zunächst in einem lokalen Cache zwischengespeichert und miteinander verglichen werden. Erst bei abgeschlossener Zwischenverarbeitung werden die Daten an die DB weitergegeben. Dies kann entweder als direkte Insert/ Update Statements oder als Übergabe an Stored Procedures geschehen.

Innerhalb dieses Microservices sollen die Positionsbestimmung der Beacons sowie die Erkennung einer kombinatorischen Nutzung stattfinden. Die Zuordnung von Beacons zu MPs erfolgt innerhalb der DB durch eine Zuordnung über ein Pflegeinterface:

- *Auf einem dedizierten Server soll ein verarbeitender Microservice laufen.*
- *Der Microservice soll die Daten aus der MQ abrufen und entsprechend ihrer Zweckbestimmung verarbeiten.*
- *Die aus der MQ ausgelesenen Daten sollen so verarbeitet werden, dass die Verarbeitungsergebnisse zunächst in einem lokalen Zwischenspeicher (folgend MemCache) gespeichert werden.*
- *Der MemCache soll zur unterbrechungsfreien Verarbeitung zwischengespeicherte Daten auch im Falle eines Neustarts des Microservice halten.*
- *Bei der Verarbeitung neu empfangener Daten sollen diese zunächst mit den im MemCache befindlichen Daten verglichen werden.*
- *Eine DB-Anfrage soll erst dann gestellt/ abgesetzt werden, wenn eine relevante Datenänderung, wie ein Hubwechsel oder eine Erstzuweisung, festgestellt wurde.*
- *Die Zuweisung eines Beacons zu einem Hub/ Raum soll anhand des Vergleichs der Empfangsfeldstärke an unterschiedlichen Hubs erfolgen (höchste RSSI gewinnt).*

- *Wenn der Knopf eines Beacons betätigt wurde, soll mithilfe des Microservices und Stored Procedures (SP) geprüft werden, ob eine Zuordnung zu einem (weiteren) Beacon nötig ist (z. B. Zubehör zu Gerät).*
- *Die Beacon zu Beacon Zuordnung soll nur dann stattfinden, wenn sich die Beacons eindeutig an demselben Ort befinden.*
- *Die Beacon zu Beacon Zuordnung soll nur dann stattfinden, wenn beide Knöpfe innerhalb eines Zeitintervalls von $t \leq 60$ Sekunden betätigt wurden.*

Die Datenhaltung bildet das Herzstück des ATS. Sie speichert alle relevanten Informationen über Beacons, Hubs, MP, Räume etc. sowie eine Historie der Datenänderungen. Diese strukturierte Datenspeicherung ermöglicht einen effizienten Zugriff auf die Informationen, die anschließend für unterschiedlichste Berechnungen und Analysen (s. o. Abb. 1), wie auch Visualisierungen genutzt werden können.

Die Datenhaltung wird mithilfe einer relationalen, Structured Query Language (SQL)-fähigen DB erfolgen, einer weit verbreiteten Art der Datenhaltung, die sich mit frei verfügbaren Tools, wie MySQL oder Maria-DB, umsetzen lässt. Diese erlaubt überdies die o. g. Nutzung von Stored Procedures und Triggern mit anschließender Datenverarbeitung, was mit korrekt gesetzten Indizes wiederum für eine schnelle Datenverarbeitung sorgt.

- *Es soll eine zentrale, relationale SQL-Datenbank (DB) geben.*
- *Auf die DB soll von mehreren Clients aus gleichzeitig zugegriffen werden können.*
- *Die DB soll für jede betrachtete Soft- und Hardwarekomponente die relevanten Daten speichern.*
- *In der DB soll eine relationale Datenstruktur zur Abbildung folgender Daten umgesetzt werden: Beacons, Hubs, MP, Räume, Bereiche, […].*
- *Die folgenden Datenänderungen sollen für mindestens 24 Monate gespeichert werden und nachträglich einsehbar sein: Standortänderung MP, Änderung MP-Kombination, […].*
- *Die Historie dieser Datensätze soll den Anfangs- und Endzeitpunkt des Vorgangs enthalten. […]*

Um die in den vorherigen Elementen des prototypisierten ATS erhobenen, verarbeiteten und gespeicherten Daten sinnvoll nutzen zu können, muss eine mögliche Interaktion mit ihnen gewährleistet sein. Wie sich aus der eigenen Umfrage ableiten lässt, wäre eine auf unterschiedlichen Geräten verfügbare Webanwendung diesbezüglich eine geeignete Lösung. Entscheidend ist, dass die darstellenden Webelemente auch ohne ein Aktualisieren der Website stets aktuell sind. Ansonsten würden aufgrund der permanenten Datenerhebung nur veraltete Daten angezeigt werden.

Die Umsetzung soll nach dem Model View ViewModel Entwurfsmuster erfolgen. Dies erlaubt eine Trennung zwischen Darstellung und Logik der Benutzerschnittstelle (UI) sowie ebenfalls des Software-

Backends. Im Webbrowser wird das ViewModel Layer durch JavaScript-Frameworks abgebildet, welche den Zustand einer HTML5-Benutzeroberfläche verwalten können.

Zusätzlich wird ein Generator benötigt, der statische und serverseitige Apps rendert. Über diesen Basisaufbau ließen sich auch über den Projektrahmen hinausgehenden Weiterentwicklungen, wie die Darstellung asynchroner Änderungen/ automatischer Aktualisierungen von Ansichten, Live-Karten o. ä., abbilden.

Zuletzt benötigen die generierten Webansichten Zugriff auf die in der DB gesicherten Daten. Hierfür wird üblicherweise eine Webschnittstelle bzw. ein Application Programming Interface (API) eingerichtet, das jegliche DB-Anfragen übernimmt. Auch wenn im Rahmen dieser Arbeit sicherheitsrelevante Komponenten, wie eine Nutzerverwaltung, nicht umgesetzt werden, lassen sich über die Nutzung dieses Aufbaus bereits Sicherheitsmechanismen, wie Cross-Origin Resource Sharing (CORS), berücksichtigen.

Die Entkopplung von DB-Zugriffen und Webanwendung erlaubt zudem eine unabhängige Entwicklung beider Komponenten.

- *Die Datenhaltung und Interaktion mit den Daten des ATS soll mit einem Client-Server-System umgesetzt werden.*
- *Alle notwenigen DB-Anfragen, welche für die Interaktion mit den Webansichten benötigt werden, sollen mithilfe einer API umgesetzt werden (Backend).*
- *Die Webanwendung soll mehrere Ansichten bereitstellen, welche folgende Inhalte darstellen sollen (Frontend): „MP-Suchansicht“ […], „MP-Serviceansicht“ […], „Raum-Serviceansicht“ […], […].*
- *Alle in den Webanwendungen gezeigten Daten/Ansichten sollen mithilfe von asynchronen Elementen umgesetzt werden (z. B. über Node.js/Vue.js).*
- *Alle Ansichten der Webanwendungen sollen sich an modernen UI-Designs orientieren, um die Nutzung zu erleichtern.*
- *Die API soll alle notwendigen Funktionen zur Dateninteraktion definieren (mit: GET, POST, PUT, DELETE), welche von dem Client aufgerufen werden können sollen.*

Mithilfe dieser Anforderungen wird es dem nutzenden Personal ermöglicht, mit den gewonnenen Daten zu interagieren und sie bei Bedarf anzupassen. Das Hinzufügen neuer Beacons, Räume, Bereiche etc. lässt sich über Serviceansichten ermöglichen, während Tabellen und Suchleisten den Standort und weitere relevante Informationen zugänglich machen.

3.4. Gesamtarchitektur

Abb. 2 zeigt eine schematische Darstellung der Gesamtarchitektur des geplanten ATS.

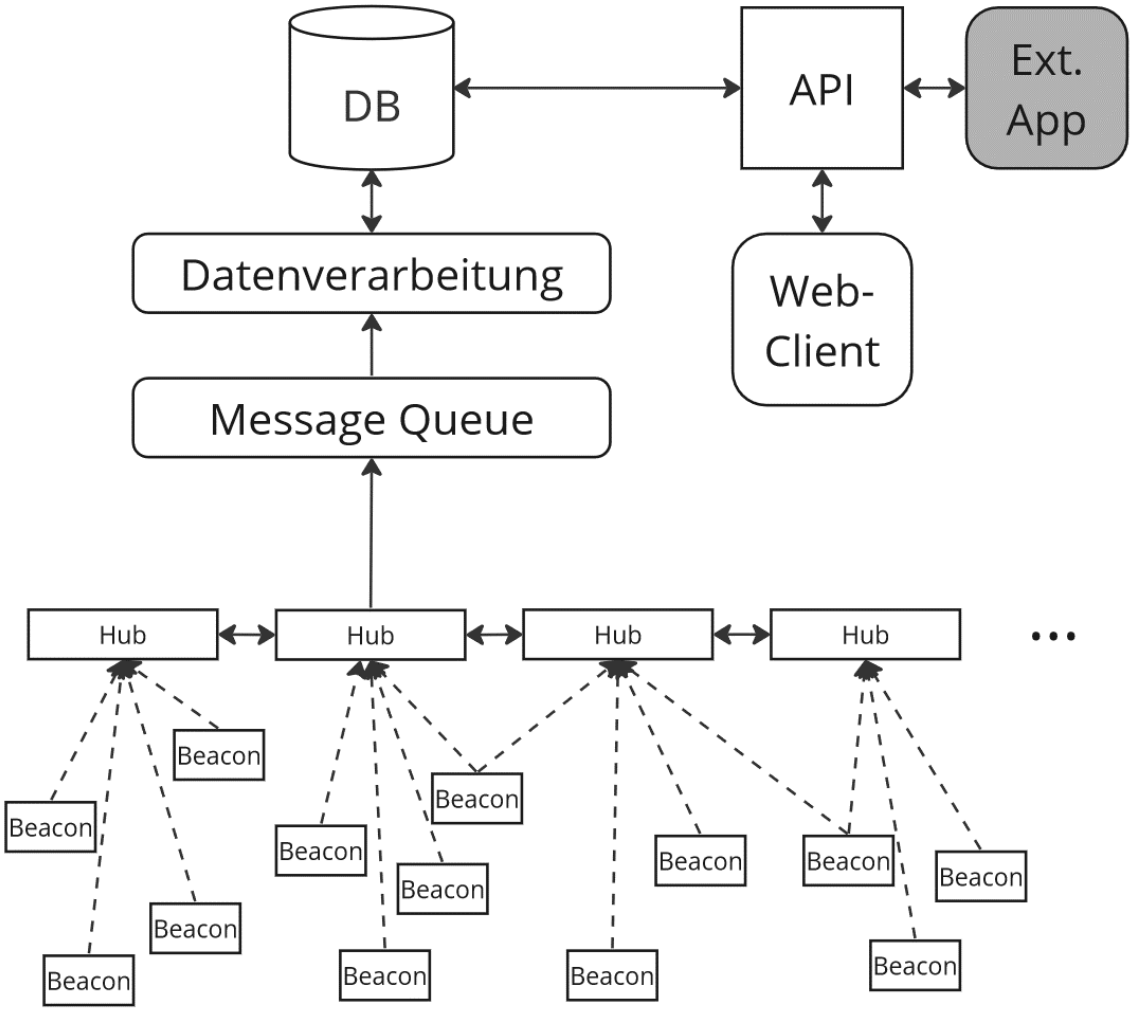


Abb. 2: Gesamtarchitektur resultierend aus Pflichtenheft

Die Abbildung veranschaulicht den Datenfluss sowie die Komponenten des ATS. Signale der Beacons werden von den Hubs empfangen und an die MQ weitergegeben. Die Datenverarbeitung empfängt diese und synchronisiert Änderungen und Zuordnungen in die DB. Die Nutzer des ATS greifen über den Webclient und die API auf die Daten der DB zu. Eine externe Anwendung könnte ebenfalls über die API auf die DB zugreifen.

Aufgrund des angesetzten Zeitrahmens war in Hinblick auf die Anforderungen des Lastenhefts eine Auswahl für die konkrete Umsetzung zu treffen. Mit Ausnahme der Bereiche Import und Export der Daten aus der Datenhaltung des ATS, Nutzergruppen und nutzerspezifische Rechte in den Webanwendungen und mehrsprachige Webansichten, werden alle Anforderungen an die Anwendung vollständig abgedeckt.

4. Entwicklung und Umsetzung

Mit der abschließenden Erstellung des Pflichtenheftes sind alle für die konkrete Umsetzung erforderlichen Informationen formuliert. Im nächsten Schritt werden die einzelnen Komponenten (siehe Kap. 3.4., Abb. 2: Gesamtarchitektur resultierend aus Pflichtenheft) chronologisch entwickelt. Dabei wird auf besondere Herausforderungen und essenzielle Code-Abschnitte eingegangen. Eine vollständige Dokumentation des Quellcodes befindet sich als Repositories auf GitHub, unter „https://github.com/linus-norden“. Alle für die Textabschnitte relevanten Dateien sind dem Anhang dieser Arbeit beigefügt.

4.1 Auswahl der Komponenten

Das Pflichtenheft lässt, trotz eindeutig formulierter Anforderungen, Spielraum bei der Auswahl der Soft- und Hardwarekomponenten. Die weitere Entwicklung erfordert eine entsprechend kriteriengeleitete Entscheidung in Hinblick auf die einzusetzenden spezifischen Komponenten.

Beacons

Als Beacons werden die „Holyiot-21011“ sowie „Holyiot-21080“ Beacons des Herstellers „Shenzhen Holyiot Technology Co.,Ltd“ verwendet, siehe Abb. 3 und 4.

Abb. 3: Holyiot-21011 Beacon Gehäuse [51]

Abb. 4:Holyiot-21080 Beacon Gehäuse [50]

Sie unterstützen den iBeacon-Standard, verwenden BLE 5.0 und verfügen über einen geeigneten Knopf. Mit einer herstellerseitigen Smartphone-App lässt sich das Sendeintervall auf bis zu 10 Sekunden einstellen, ohne, dass eine andere Software aufgespielt werden muss. Der Ruhestromverbrauch liegt bei 6 µA. Während das große Beacon Standard AAA Zellen nutzt und somit für Jahre betriebsfähig ist, wird im kleinen Beacon eine Knopfzelle mit nur 220 mAh Kapazität eingesetzt. Bei einem Sendeintervall von 500 ms liegt der mittlere Stromverbrauch bei 40 µA, bei 1 Sekunde bei 22 µA und bei 10 Sekunden bei wenig über 6 µA. Das entspricht jeweils einer Lebensdauer von 229 Tagen, 416 Tagen und 1145 Tagen. Damit erfüllen die Beacons alle gestellten technischen Anforderungen.

Entscheidend bei der Anschaffung von Beacons für die prototypische Umsetzung war neben der technischen Funktionen auch der Preis. Die Beschaffungskosten von Kleinstmengen liegen (Stand 05/2024)

bei 7 € bzw. 11 € pro Beacon. Für den Einsatz in einer medizinischen Einrichtung ist Anhand des geplanten Einsatzszenarios jeweils eine geeignete Auswahl zu treffen.

Hubs

Um die geforderte Funktionalität der Hubs zu gewährleisten, muss eine Lösung genutzt oder erstellt werden, die über getrennte Antennen zur Bluetooth- und WLAN-Kommunikation verfügt. Dies ist zwingend erforderlich, da nach BLE-Geräten gescannt werden soll, während ein WiFi-Mesh aufgebaut bzw. aufrechterhalten wird. Stromverbrauch und Formfaktor sind bei der Auswahl aufgrund der geplanten Verwendung an gängigen Steckdosen nicht von großer Relevanz. Unter den aktuell verfügbaren Microcontroller (MC)- Entwicklungsplattformern finden sich viele, die sowohl WLAN als auch Bluetooth unterstützen, allerdings nur wenige, die die geforderten zwei Antennen besitzen. In den meisten Anwendungsfällen genügen Boards mit einer Antenne. Boards mit zwei Antennen sind erheblich teurer im Einkauf. Daher wurde sich dafür entschieden, den Hub aus zwei einzelnen Entwicklungsplattformen zu fertigen, die nach Verlöten der Pins direkt miteinander kommunizieren können. Dies senkt nicht nur die Kosten, sondern vereinfacht auch das Programmieren, da beide Komponenten separat voneinander bespielt werden können.

Mit einer hohen Rechenleistung bei geringem Preis und breiter Verfügbarkeit erscheinen ESP32C3-Boards ideal für die Umsetzung der geforderten Funktionalität. Dabei handelt es sich um Entwicklungsplattformen, die den ESP32-C3 von Espressif nutzen. Boards mit diesem MC werden in unterschiedlichsten Branchen für verschiedenste Aufgaben genutzt und sind daher weit verbreitete Bauteile. Entsprechend finden sich zahlreiche Open-Source Bibliotheken und Programmcodes sowie Erklärungen zu umsetzbaren Funktionen, was wiederum die Entwicklung erleichtert. Das Espressif IoT Development Framework (Espressif IDF) trägt mit den verfügbaren Werkzeugen, Bibliotheken und Beispielprojekten ebenfalls zu einer einfacheren Entwicklung bei. Für die Entwicklung wurden ESP32-C3 SuperMini Entwicklungsboards des Herstellers Shenzhen Yuda Technology Co., LTD verwendet. Abb. 5 zeigt ein einzelnes Board, Abb. 6 den Zusammenschluss aus zweien. Die Beschaffungskosten von Kleinstmengen liegen (05/2024) bei 2,50 € pro Board. Hardwarekosten eines Hubs inklusive Gehäuse, USB-Netzteil und Kabel liegen aktuell bei ca. 15 €.

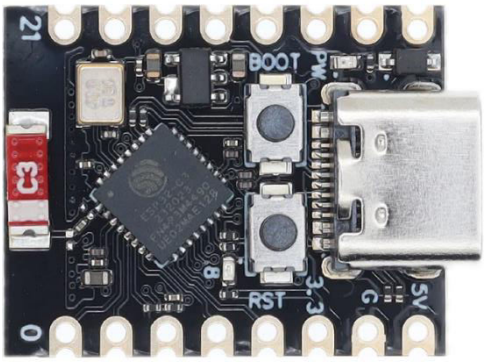


Abb. 5: ESP32-C3 Entwicklungsboard [49]

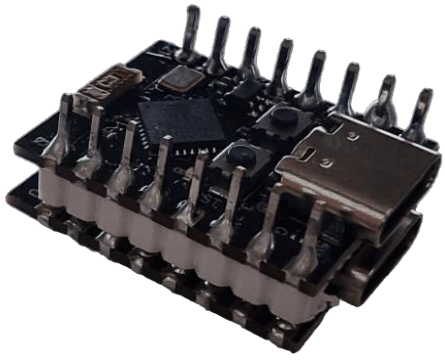


Abb. 6: Zwei ESP32-C3 Entwicklungsboards als Hub

MQTT-Broker

Als MQTT-Broker wird der von der Eclipse Foundation zur Verfügung gestellte, o. g., Broker genutzt. Die Open-Source-Natur und weit verbreitete Nutzung machen ihn zu einem perfekten Werkzeug für die geplante Umsetzung. Empfangenen Nachrichten wird zwar nativ kein Zeitstempel beigefügt, der Open-Source-Code erlaubt allerdings die Nutzung von vorhandenen Plugins oder eine eigenständige Anpassung, um den Eingangszeitpunkt einer Nachricht hinzuzufügen. Damit sind alle Grundlagen gegeben, die für die weitere Entwicklung benötigt werden.

Microservice

Für die Verarbeitung der aus der MQ entnommenen Daten braucht es, wie in Kap. 3.3. „Anforderungen an Softwarekomponenten“ beschrieben, einen Microservice. Dieses Programm muss dazu fähig sein, Daten effizient zu verarbeiten und an eine DB zu übergeben. Aufgrund seiner Einfachheit, der breiten Verfügbarkeit von Bibliotheken und Frameworks sowie einer persönlichen Präferenz wurde sich für eine Entwicklung mit Python entschieden.

Als Teil des Microservices wird zudem ein Memory-Cache benötigt. Für das prototypisierte ATS wird der de-facto-Standard Memcached genutzt [40]. Er steht sowohl unter Unix als auch unter Windows zur Verfügung.

Datenbank

Wie ebenfalls in Kap. 3.3. beschrieben, wird die Datenhaltung des ATS mit einer relationalen SQL-DB realisiert. Dafür wird eine Maria-DB verwendet, da der Programmcode vollständig als Open-Source zur Verfügung gestellt wird und somit transparent ist. Die DB ließe sich auch mit den Produkten vieler weiterer Anbieter umsetzen, solange diese gängige Methoden, wie beispielsweise die später folgenden Stored Procedures, unterstützen.

Webanwendung

Die Interaktion mit den Daten des ATS wird mithilfe von Vue.js umgesetzt. Es handelt sich um ein modernes, leistungsfähiges JavaScript-Framework mit einer umfangreichen Anzahl an Bibliotheken und einer großen aktiven Community. Als Generator für statische und serverseitig gerenderte Apps bietet sich Nuxt.js an, da es speziell für die Verwendung mit Vue.js konzipiert wurde. Eine API als Schnittstelle zur DB wird mit Node.js umgesetzt. Damit basieren Frontend und Backend einheitlich auf JavaScript, was die Entwicklung erleichtert.

4.2. Entwicklungsumgebung

Für die Entwicklung aller Komponenten des ATS wird Virtual Studio Code (VSCode) von Microsoft als primäre Entwicklungsumgebung genutzt. VSCode ist ein leichtgewichtiger Quelltext-Editor, der ein

breites Spektrum an Erweiterungen bietet, die nach Bedarf hinzugefügt werden können. Darunter befinden sich auch Erweiterungen für das Arbeiten mit Python, mit dem Espressif IDF sowie mit SQL-DB. Funktionen wie das integrierte Debugging und IntelliSense, ein Werkzeug für Codevervollständigung und Syntax-Highlighting, erleichtern das Programmieren mit VSCode zusätzlich.

Für die Arbeit mit der Maria-DB wird zusätzlich die MySQL-Workbench verwendet. Diese erleichtert die Erstellung von Entity-Relationship-Modellen (ERM), bietet mit Forward- und Reverse-Engineering ein wichtiges Werkzeug für größere Änderungen und vereinfacht das Anpassen von Tabellen, Indizes und Triggern bei aktivem DB-Betrieb erheblich.

4.3. Laufzeitumgebung

Jegliche Softwarekomponenten werden auf einem dedizierten Ubuntu Server betrieben. So kann das ATS in einer Art und Weise erstellt und getestet werden, die dem Betrieb innerhalb einer medizinischen Einrichtung nahekommt.

4.4. Entwicklung Beacon und Hub

4.4.1. Beacons

Für die Umsetzung werden Holyiot-Beacons verwendet, die den iBeacon-Standard unterstützen. Dieser legt die Datenstruktur des Pakets fest, an dem iBeacons als solche erkannt werden. [35] Wie die überwiegende Mehrzahl, senden auch diese Beacons auf Anfrage allerdings noch ein zweites Paket, das sog. Advertising-Package, in dem weitere Daten enthalten sind. Die Advertising-Packages unterliegen dabei lediglich strukturellen Vorgaben. Der Inhalt kann beliebig vom Hersteller bestimmt werden.

Diese Struktur ist in der Bluetooth Core Specification 5.4, Volume 3 (Host), Part C („Generic Access Profile“) unter Punkt 11 „Advertising and scan response format” definiert und folgt dem in Abb. 7 dargestellten Format [38].

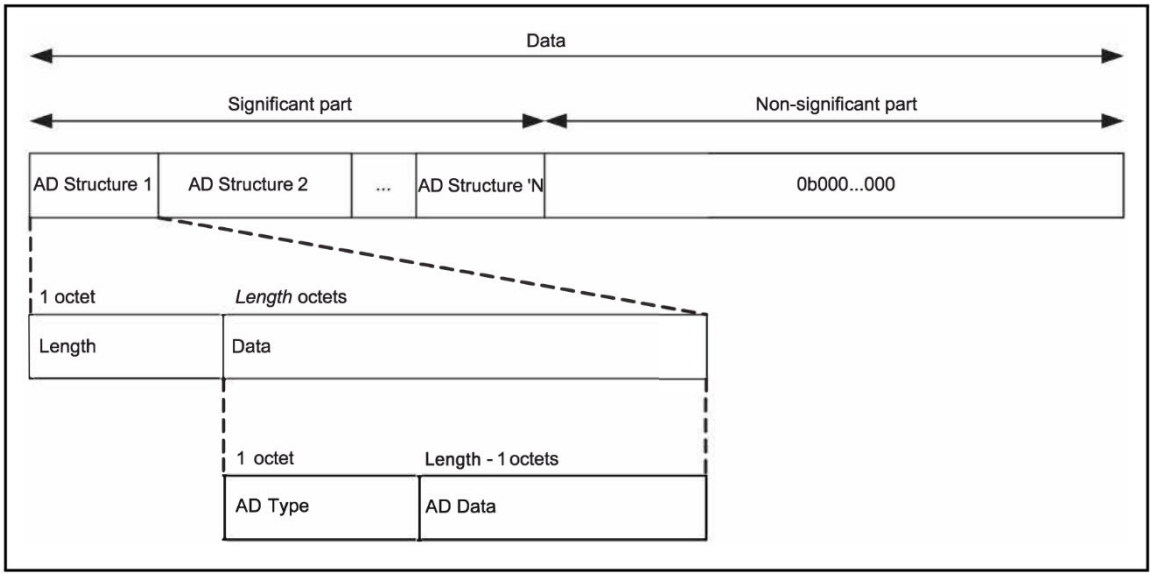


Abb. 7: Format der Advertising- und Scan-Antwort eines Beacons [38]

Die „Core Specification Supplement, Part A, Data Types Specification" definiert die innerhalb dieser Daten zulässigen Advertising Data (AD)-Typen [39]. Umsetzungsrelevant ist vor allem die unter 1.16. „LE Bluetooth Device Address" beschriebene 48 Bit Adresse des Beacons, die in der praktischen Nutzung der MAC-Adresse eines Netzwerk-Interfaces entspricht. In der prototypischen Umsetzung wird sie für die eineindeutige Identifikation der Beacons genutzt.

Hersteller von Beacons verwenden zum Advertising ausschließlich die in der Liste des „Assigned Numbers Document" unter dem Punkt „2.3 Common Data Types" definierten Typen [40].

Relevant für die folgende Umsetzung sind hieraus:

- AD-Type 0x09 – Complete local name

  Der per App/ Beacon Konfiguration vergebene Name des Beacons. Für die Entwicklung dient er zum einfachen Debugging/ der schnellen Identifikation des empfangenen Beacons anhand eines vergebenen Namens.

- AD-Type 0x16 – Service Data

  Als Service Data werden 16 Bit übertragen, die gerätespezifische Daten enthalten. Im Fall der genutzten Holyiot-Beacons befinden sich darin die MAC-Adresse, der Batteriezustand und der Buttonstatus.

Durch den Einsatz von Standard Beacons kann für diese Komponente auf eine Programmierung verzichtet werden. Über eine herstellerseitige App wird lediglich:

- ein Sendeintervall festgelegt (10s, energiesparend),
- ein Name vergeben (Debugging),
- der UUID gesetzt (Möglichkeit des Ausschlusses fremder Beacons).

Ein Teil der Oberfläche der App ist in beispielhaft in den Abb. 8 und 9 dargestellt.

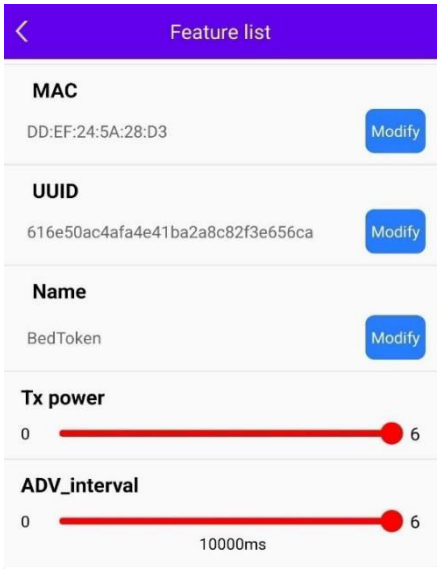


Abb. 8: Holyiot App Beaconeinstellungen

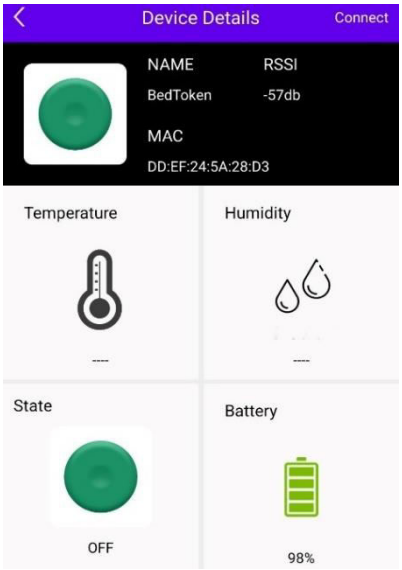


Abb. 9: Holyiot App Beaconübersicht

Die Datenkommunikation eines Beacons erfolgt über zwei Datenpakete. Das erste wird vom Beacon an umliegende Geräte gesendet, das zweite kann von einem Empfänger des Beacons abgefragt werden. Das gesendete Datenpaket folgt dem o. g. iBeacon-Standard und enthält 31 Bytes.

```
0x02, 0x01, 0x06   2 bytes, type = flags, flags = Bit 1: “LE General Discoverable Mode”,
      Bit 2: “BR/EDR Not Supported.”
0x1A, 0xFF, 26 Bytes, type = Manufacturer Specific Data
            0x4C, 0x00, - company_id
            0x02, 0x15, - beacon_type
      0x61,0x6e,0x50,0xac,0x4a,0xfa,0x4e,0x41,0xba,0x2a,0x8c,0x82,0xf3,0xe6,0x56,0xca,  -
proximity_uuid (über die App gepflegt)
            0x00, 0x00, - major (Werte, die als Steuerungsinformationen genutzt werden
      können)
            0x00, 0x00, - minor (Werte, die als Steuerungsinformationen genutzt werden
      können)
            0x80-  measured_power
```

Die Antwort eines Holyiot-Beacons auf einen vom Empfänger automatisch erfolgten Scan, also das Advertising Package, folgt diesem Schema:

```
0x04, 0x09, 'L', '0', 0x00, - 4 bytes, type = local name
0x11, 0x16, 0x42, 0x52, - 17 Bytes, type = service class
      0x41, 0x52, 0xF0, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
```

Der Batteriestatus ist in Byte 3 und der Status des Knopfes in Byte 13 zu finden.

4.4.2. Hubs

Die Umsetzung des Hubs erfolgt über eine Komponente zum Empfang der Beacon Daten und eine Komponente zum Bilden des Mesh-Netzes sowie Versenden der Daten (siehe Kap. „3.2. Anforderungen an Hardwarekomponenten“).

Für beide Komponenten, liefert Espressif als Hersteller der ESP32-C3-Boards Beispielprojekte. Die Entwicklung der beiden Komponenten des Hubs setzt auf diesen Beispielen auf und bereinigt und ergänzt diese.

Die Umsetzung besteht aus:

- dem Entfernen der für ein ATS nicht genutzten Elemente der Beispiele,
- der Ergänzung um die Eingangsverarbeitung von iBeacon-Signalen,
- der Entwicklung einer Kommunikation zwischen der Scan- und der Mesh-Komponente

Board zu Board Kommunikation

Die vom BLE-Board empfangenen und gefilterten Beacon Daten müssen dem Mesh-Board übergeben werden. Hierfür wird eine serielle Schnittstelle verwendet (siehe Kap. 3.3. „Anforderungen an Softwarekomponenten“). Der ESP32-C3 unterstützt hardwareseitig drei serielle Schnittstellen (Universal

Asynchronous Receiver-Transmitter (UART)). Das Nutzen der zweiten Schnittstelle zur Kommunikation zwischen den Komponenten verhindert Probleme mit dem Debugging über die erste Schnittstelle und USB-C.

Das Limit der hardware-Schnittstellen liegt bei 5.000.000 Bit/s. Für die Kommunikation zwischen den Komponenten wird eine Geschwindigkeit von 115.200 Bit/s festgelegt. Bei einer Länge von 9 Byte pro Beacon-Datenpaket könnte das BLE-Board mit dieser Konfiguration hypothetisch bis zu 1600 dieser Pakete an das Mesh-Board weiterleiten.

Scan – Komponente

Als Basis des BLE-Boards dient der Source Code zur iBeacon Demo von Espressif [41]. Dieser wurde lediglich leicht angepasst, um die serielle Datenübertragung umzusetzen. Im Folgenden werden die Konfiguration und die Grundfunktion beschrieben, alle darüber hinausgehenden Informationen können der Dokumentation von Espressif entnommen werden.

Konfiguration:

Die Parameter der seriellen Schnittsteller werden wie folgt definiert:

```
29. #define BOARD2BOARD_TXD 3
30. #define BOARD2BOARD_RXD 4
31. #define BOARD2BOARD_RTS (UART_PIN_NO_CHANGE) // -1, ungenutzt
32. #define BOARD2BOARD_CTS (UART_PIN_NO_CHANGE) // -1, ungenutzt
33.
34. #define BOARD2BOARD_PORT_NUM 1 //esp32c3 bietet Ports 0-2 an
35. #define BOARD2BOARD_BAUD_RATE 115200
36. #define BOARD2BOARD_STACK_SIZE 2048 // UART-Stack Größe, mehr siehe: https://docs.espres-
                                        sif.com/projects/esp-idf/en/latest/esp32/api-refe-
                                        rence/peripherals/uart.html
37. #define BOARD2BOARD_BUF_SIZE 1024 // Zwischenspeicher für Nachrichten
```

Zur Filterung der empfangenen Beacons, wird der ‚eigene' UUID als Empfangskriterium festgelegt.

```
55. static const uint8_t OWNED_UUID[] = {0x61,0x6e,0x50,0xac,0x4a,0xfa,0x4e,0x41,0xba,0x2a,0x8c,
                                        0x82,0xf3,0xe6,0x56,0xca};
```

Zur Bereinigung des Beispiel-Codes werden alle Funktionen zum Senden von iBeacons aus dem Code entfernt. Der für diesen Textabschnitt relevante Quellcode befindet sich im Anhang unter A6.

Kernfunktion:

Wenn ein den BLE-Scan betreffendes Ereignis eintritt, wird die Methode „esp_gap_cb" aufgerufen. Wird ein Bluetooth Gerät erkannt und im Falle eines abfragbaren Gerätes, wie ein iBeacon es darstellt, ein Scan ausgeführt, tritt das Ereignis „ESP_GAP_BLE_SCAN_RESULT_EVT" ein.

Die gesamte Verarbeitung findet innerhalb dieses Ereignis-Handlers statt. Liegen Scanergebnisse vor und ein iBeacon wird erkannt, wird geprüft, ob Servicedaten vorhanden sind. Ist dies der Fall und stimmt auch der konfigurierte UUID zum Filtern überein, werden aus den Servicedaten die 6 Bytes der MAC-Adresse sowie die 2 Bytes für Batterie- und Buttonstatus ausgelesen.

Für den Zugriff auf die strukturierten Informationen der BLE-Kommunikation bietet Espressif Strukturdefinitionen und Zugriffsmethoden an:

- Der UUID wird geholt über ibeacon_data->ibeacon_vendor.proximity_uuid
- Das Service Data Paket wird ausgelesen über esp_ble_resolve_adv_data (param->scan_rst.ble_adv, ESP_BLE_AD_TYPE_SERVICE_DATA, &adv_service_data_len)
- Der Batteriewert steht in adv_service_data[3]
- Der Buttonstatus steht in adv_service_data[13]
- Der RSSI-Wert des empfangenen Pakets kann über scan_result->scan_rst.rssi abgefragt werden

Sind diese Informationen zusammengestellt, werden sie an die konfigurierte Schnittstelle zur Übertragung übergeben.

```
127.                        uart_write_bytes(BOARD2BOARD_PORT_NUM, (const char *)serial_data, 9);
```

Mesh – Komponente

Als Basis für das Mesh-Board dient der Source-Code des „IP internal Network“-Beispiels von Espressif [42]. Das Verfahren zum Aufbau eines Mesh-Netzes ist recht komplex. Da es im Rahmen dieser Arbeit in nicht modifizierter Form verwendet wird, wird an dieser Stelle lediglich auf den ESP-IDF Programming-Guide verwiesen [43, 44]. Der für diesen Textabschnitt relevante Quellcode befindet sich im Anhang unter A7.

Die aus dem Guide entnommene Abb. 10 zeigt die Übersicht zur Netzwerktopologie.

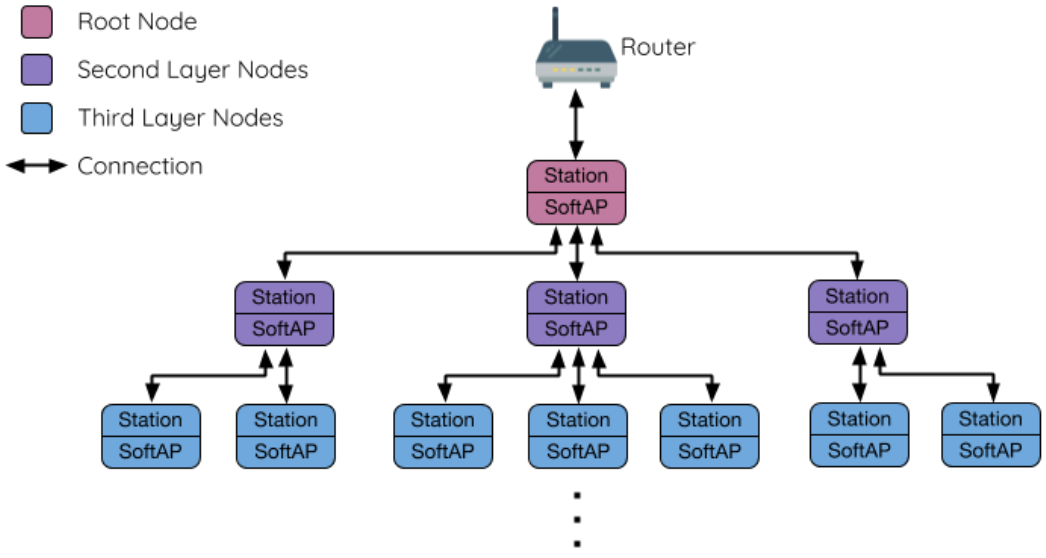


Abb. 10: ESP-WIFI-MESH Netzwerktopologie [44]

Für die Umsetzung des ATS sind folgende Punkte relevant:

- Die komplette Selbstorganisation des Mesh-Netzes
    - mit automatischer Bestimmung der Root-Node,
    - mit automatischer Strukturentwicklung innerhalb des Netzes bei neuen Knoten oder mit sich ändernden Verteilungen,
    - mit dem Durchreichen von Datenpaketen bis an die Root-Node
    - mit der Weiterleitung von Paketen an den externen Router,
    - mit der verschlüsselten Kommunikation innerhalb des autarken Mesh-Netzes,
- Die Konfigurationsmöglichkeit der Breite und Tiefe des Netzes und damit die Limitierung der erforderlichen Verbindungen in das WLAN einer medizintechnischen Einrichtung,
- Das Zwischenspeichern von Kommunikationspaketen im Falle eines temporären Wegfalls einer Verbindung.

Konfiguration:

Für die Konfiguration des Mesh-Netzes sind wenige Vorgaben erforderlich. Für das ATS wird die Verschlüsselung des Mesh auf WPA2-PSK festgelegt und ein Kennwort vergeben. Damit können sich die Mesh-Boards untereinander verbinden. Für die Kommunikation zum Router werden den Boards die SSID des Routers und das Kennwort übergeben.

Das Projekt bietet die Möglichkeit, die maximale Größe des Mesh-Netzes zu bestimmen. Hierfür werden die Verbindungen pro Knoten und die Tiefe festgelegt. Diese bestimmen den Speicher- und Abstimmungsbedarf der Routing-Tabellen innerhalb eines Netzes:

- MESH_AP_CONNECTIONS = 6 – Anzahl der für eine Node zulässigen Eingangsverbindungen
- MESH_MAX_LAYER = 6 – Maximale Anzahl von Ebenen innerhalb des Mesh

Diese Konfiguration entspricht einer theoretischen maximalen Anzahl von 6^5 = 7.776 Hubs/ Räumen pro Verbindung zu einem Router.

Das Espressif Beispiel nutzt bereits die benötigten MQTT-Bibliotheken, so dass sich die Konfiguration der MQTT-Anbindung auf die Zugangsdaten des MQTT-Servers reduziert.

Kernfunktion:

Die Kernfunktionen der Mesh-Komponente sind das Empfangen, Aufbereiten und Weiterleiten der Daten der Scan-Komponente.

Anstelle der kontinuierlichen Kommunikation des Mesh-Status einer Node an den MQTT-Server, wie sie im Espressif Code-Beispiel vorhanden ist, wird für dieses Projekt die Aufbereitung und Weitergabe der Seriellen Informationen des Scan Boards eingefügt.

Der relevante Ausschnitt der Verarbeitungsfunktion sieht wie folgt aus:

```
105. void esp_mesh_mqtt_task(void *arg)
106. {
[…]
131.         uint8_t serial_data[9]; // MAC + Batt + Button + RSSI
132.         int length = 0;
133.         ESP_ERROR_CHECK(uart_get_buffered_data_len(BOARD2BOARD_PORT_NUM, (size_t*)&length));
134.         while (length > 0) {
135.             ESP_LOGI(MESH_TAG, "uart rx length %d", length );
136.             // Prüfung, ob mindestens ein Datenpaket mit 9 Zeichen vorhanden ist
137.             if (length >= 9) {
138.                 // Versuche 9 Bytes zu lesen
139.                 if ( uart_read_bytes(BOARD2BOARD_PORT_NUM, serial_data, 9, 100) ) {
140.                     esp_log_buffer_hex("Read data:", serial_data, 9);
141.                     // MQTT-Nachricht für Beacon und Hub erstellen
142.                     asprintf(&print, "layer=%d, MAC_ROOM=%02X:%02X:%02X:%02X:%02X:%02X,
                                          MAC_SENSOR=%02X:%02X:%02X:%02X:%02X:%02X, BATT=%d,
                                          BUTTON=%d, RSSI=%d", \
143.                         esp_mesh_get_layer(), mac[0], mac[1], mac[2], mac[3], mac[4],
                             mac[5], \
144.                         serial_data[0], serial_data[1], serial_data[2], serial_data[3], se-
                             rial_data[4], serial_data[5], \
145.                         serial_data[6], serial_data[7], serial_data[8] );
146.                     ESP_LOGI(MESH_TAG, "Tried to publish %s", print);
147.                     mqtt_app_publish("/sensors/rooms", print);
148.                     free(print);
149.                 }
150.             } else {
151.                 // Wenn dieser Zustand ein zweites Mal mit mehr als 0 aber weniger als 9
                    Zeichen erreicht wird, muss ein Fehler in der Kommunikation aufgetreten sein
152.                 // Daher vorsichtshalber den Buffer leeren
153.                 if (is_broken_package == true) {
154.                     ESP_ERROR_CHECK(uart_flush(BOARD2BOARD_PORT_NUM));
155.                     is_broken_package = false;
156.                 } else is_broken_package = true;
157.             }
158.         }
159.         vTaskDelay(60000 / portTICK_PERIOD_MS);
160.         // MQTT-Nachricht für Hub erstellen
161.         asprintf(&print, "layer=%d, MAC_ROOM=%02X:%02X:%02X:%02X:%02X:%02X", \
162.             esp_mesh_get_layer(), mac[0], mac[1], mac[2], mac[3], mac[4], mac[5] );
163.         ESP_LOGI(MESH_TAG, "Publish %s", print);
164.         mqtt_app_publish("/sensors/rooms", print);
165.     }
```

Die seriellen Daten im Zwischenspeicher werden in der vom Scan-Modul gelieferten Größe von 9 Bytes gelesen und verarbeitet. Sollte durch einen Fehler eine Datenmenge im UART stehen, deren Länge kein Vielfaches von 9 ist, so wird dies vermerkt. Wenn dieser Fall zweimal hintereinander auftritt, werden die Daten des Zwischenspeichers bereinigt.

Die empfangenen 9 Bytes werden nach Empfang in die MAC-Adresse des Beacons, den Batteriestatus und den Buttonstatus des Beacons separiert. Zusammen mit der MAC-Adresse des Mesh-Boards als Hub-MAC-Adresse und der Information zum Layer im Mesh werden die Daten an den nächsthöheren Knoten des Meshs weitergereicht und im Falle der Root-Node an den Router/ MQTT-Server weitergegeben.

Eine an den MQTT übertragene Nachricht folgt dem Format:

```
„layer=%d, MAC_ROOM=%02X:%02X:%02X:%02X:%02X:%02X,
      MAC_SENSOR=%02X:%02X:%02X:%02X:%02X:%02X,
      BATT=%d, BUTTON=%d, RSSI=%d“
```

Den umgesetzten Ablauf der Datenverarbeitung stellt Abb. 11 in Form eines Flowchart dar.

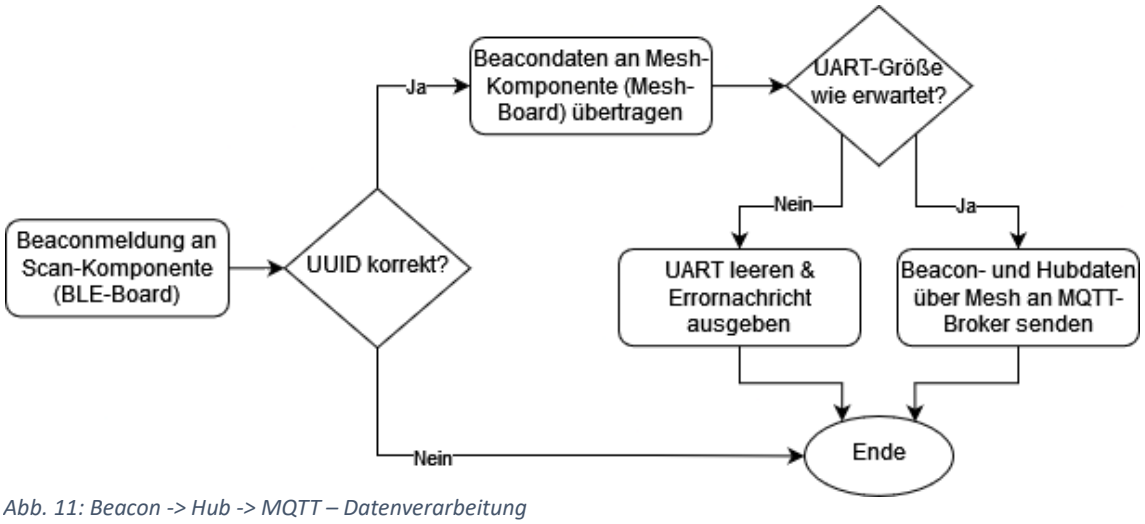


Abb. 11: Beacon -> Hub -> MQTT – Datenverarbeitung

4.5. Modifizierter MQTT-Broker

Verschiedene Entwickler haben Lösungen für die Erweiterung des mosquitto MQTT in ihren Repositories zur Verfügung gestellt. Leider befindet sich bisher keine darunter, die einer eingehenden Nachricht den aktuellen UNIX-Zeitstempel beifügt und das Ergebnis im JSON-Format ausgibt.

Für eine beim MQTT vom Hub eingehende Nachricht, wie beispielsweise

```
“layer=2, MAC_ROOM=64:E8:33:88:38:14, MAC_SENSOR=DD:EF:24:5A:28:D4, BATT=99, BUTTON=1,
      RSSI=240”
```

soll durch das Hinzufügen des Zeitstempels und Umformung in JSON dieses Schema für die Zwischenspeicherung in der MQ genutzt werden:

```
„{"time":"1716645400","data":"layer=2, MAC_ROOM=64:E8:33:88:38:14,
      MAC_SENSOR=DD:EF:24:5A:28:D4, BATT=99, BUTTON=1, RSSI=240"}“
```

Abgesehen von diesem Hinzufügen des Zeitstempels, soll der mosquitto Broker in seiner Funktion unverändert bleiben. Für diese Funktion ist eine Anpassung der „handle_publich.c“ Datei im „src“ Verzeichnis des Brokers erforderlich [45].

Zu Beginn wird die Bibliothek „time.h“ importiert, um die aktuelle Zeit als UNIX-Zeitstempel abrufen zu können.

```
35. #include <time.h> // Um Zeitstempel zur aktuellen Uhrzeit abzufragen
```

Alle weiteren Änderungen finden in der Funktion „handle_publish“ statt. Dort wird zunächst die aktuelle Zeit abgefragt. Anschließend wird zusätzlicher Speicher für das verarbeitete Payload reserviert, um den Zeitstempel und das ursprüngliche Datenfeld im JSON-Format aufzunehmen.

```
236.        time_t current_time;
237.        time(&current_time);
238.
239.        // Erstellen eines neuen Strings, der die Zeit und das ursprüngliche JSON-Payload
             enthält
240.        char *new_json_str = mosquitto__malloc(msg->payloadlen + 64); // Zusätzlicher Spei-
                                                                          cherplatz für den Zeitstempel
                                                                          und Trennzeichen
```

Da die endgültige Nachricht aus zwei Teilen zusammengesetzt ist, muss die eingehende Nachricht in einem temporären String gespeichert werden. Danach wird der Speicher für das endgültige Message Payload reserviert, womit die Vorbereitung für das Hinzufügen des Zeitstempels erfolgt ist.

```
246.        // Erstellen eines neuen Strings, der die ursprüngliche Nachricht beinhaltet
247.        char *tmp_msg = mosquitto__malloc(msg->payloadlen);
[…]
252.
253.        // Speicherzuweisung für die endgülttige Nachricht
254.        msg->payload = mosquitto__malloc(msg->payloadlen+64); // Zusätzlicher Speicherplatz
                                    für den Zeitstempel und Trennzeichen ( war vorher: msg-
                                    >payload = mosquitto__malloc(msg->payloadlen+1); )
```

Zum Abschluss muss der JSON-String in dem erforderlichen Schema gebildet und übergeben werden.

```
274.        // Umwandlung des Payload-void-Pointers in einen char-Pointer zur erleichterten Ma-
             nipulation
275.        char *json_str = (char *)tmp_msg;
276.
277.        // Verketten von Zeitstempel und ursprünglichem Payload
278.        sprintf(new_json_str, "{\"time\":\"%ld\",\"data\":%s}", current_time, json_str);
279.
280.        // Anschließendes übergeben des String 'new_json_str' an nachfolgende Funktionen
281.        memcpy(msg->payload, new_json_str, (size_t)strlen(new_json_str));
282.        msg->payloadlen = (uint32_t)strlen(new_json_str);
283.        mosquitto_free(new_json_str); // Löschen, falls Error
284.        mosquitto_free(tmp_msg);
```

Damit ist die Modifizierung des MQTT-Brokers abgeschlossen. Die vollständig angepasste Datei befindet sich im Anhang unter A8.

4.6. Entwicklung Microservice und Datenbank

Wie zuvor beschrieben, werden die aus der MQ entnehmbaren JSON-Nachrichten von einem Python-Skript abgefragt, verarbeitet und die resultierenden Daten anschließend in einer Maria-DB gespeichert. Beide Komponenten sind in ihrer Funktion eng miteinander verbunden und voneinander abhängig. Die Entwicklung erfolgt parallel. Es muss bestimmt werden, welche Datenverarbeitungsschritte im Microservice und welche über Stored Procedures in der DB stattfinden sollen. Auch muss die Datenhaltung in beiden Softwareteilen ausgearbeitet werden, wofür ein Entity-Relationship-Modell (ERM) für die Maria-DB erstellt wird. Erst nach Erstellung des ERM erfolgt die Programmierung des Microservices.

Aufteilung der Datenverarbeitung

Die wichtigsten Verarbeitungsschritte bezüglich der Beacondaten lassen sich wie folgt unterteilen:

- Ortung der Beacons (Erstzuordnung zu Hub, Hubwechsel)
- Beacon- zu Beacon Zuordnung (Prüfung auf valide Zuordnung)
- Auflösung der Zuordnung bei Hubwechsel (Prüfen, ob Partner nachrückt)
- Historienerstellung bei Datenänderung von Beacons

Den Ablauf dieser Datenverarbeitungsschritte stellt die Abb. 12 als Flowchart dar.

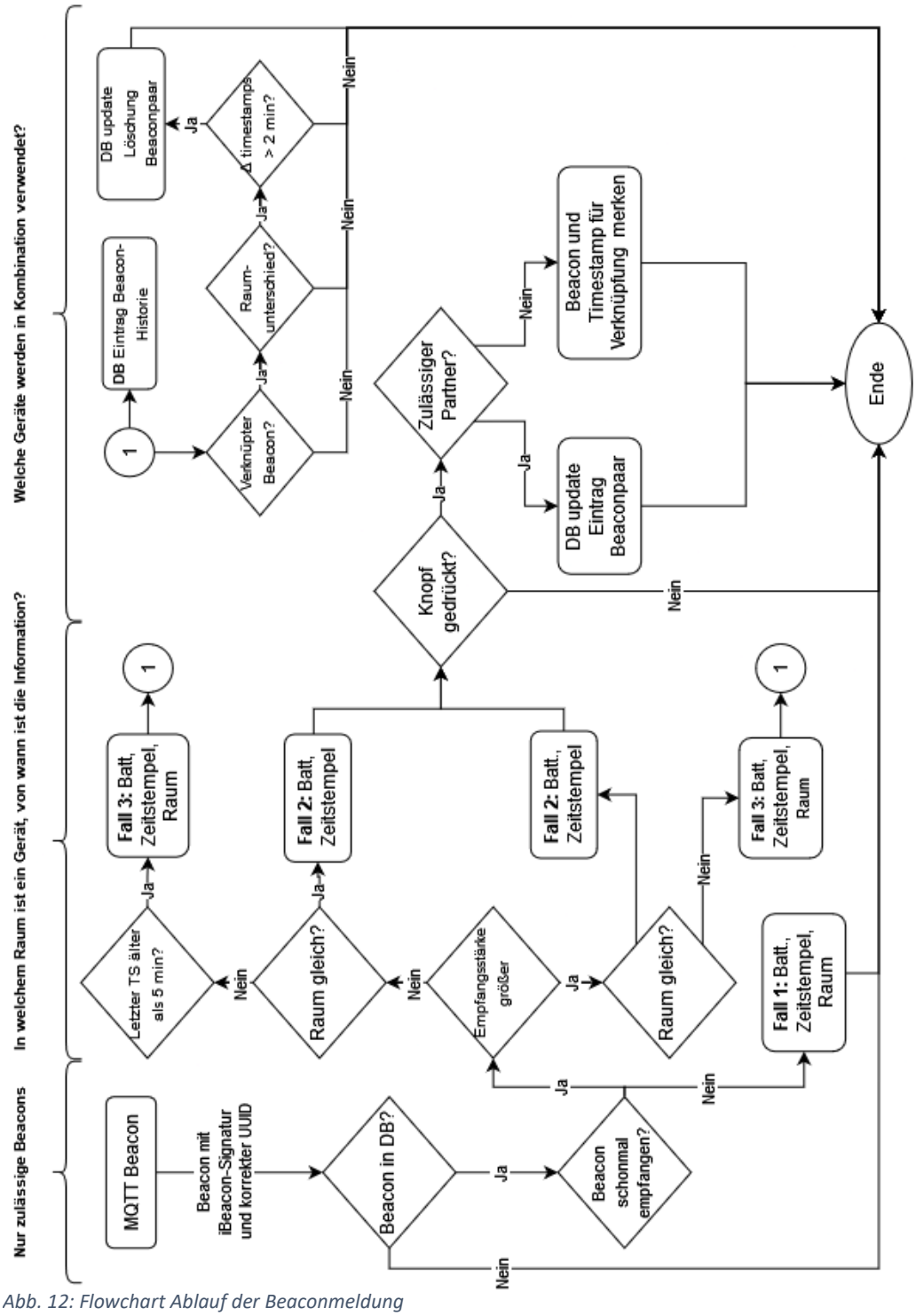


Abb. 12: Flowchart Ablauf der Beaconmeldung

Nicht jeder Verarbeitungsschritt muss in einer DB-Anfrage münden. Wird beispielsweise ein Beacon gemeldet, der sich immer noch im selben Raum befindet, müssen die RSSI und der Batteriestand nicht sofort in der DB aktualisiert werden. Um das Risiko relevanter Datenverluste zu minimieren, genügt es, wenn sie für Python-interne Prozesse im Memcache zur Verfügung stehen und alle 5-10 Minuten an die DB übertragen werden.

Zusätzlich zu den genannten Schritten und den drei Fällen gibt es weitere, die jedoch nicht aus den in der MQ gespeicherten Beacondaten resultieren. Sowohl das Hinzufügen von neuen Räumen, Bereichen, Hubs etc. als auch die Historienerstellung bei einer Änderung an diesen, sind für die Anwendung relevant. Ersteres wird über die später erstellten Webansichten ermöglicht, letzteres durch Stored Procedures und Trigger in der Maria-DB gelöst. Bis auf die Historienpflege der Beacons werden alle im Flowchart visualisierten Prozesse innerhalb des Python-Microservices umgesetzt.

Erstellung der Datenbankstruktur

Mit diesem Wissen und den im Pflichtenheft genannten Anforderungen an die Datenhaltung lassen sich die erforderlichen DB-Tabellen erstellen. Alle Tabellen und deren Abhängigkeiten voneinander werden nachfolgend in einem ERM dargestellt. Als detailliertes Erläuterungsbeispiel dient die „beacon“-Tabelle aus Abb. 13. Auf eine Erläuterung der anderen Tabellen wird im Rahmen dieser Arbeit verzichtet, da diese dem gleichen Schema folgen.

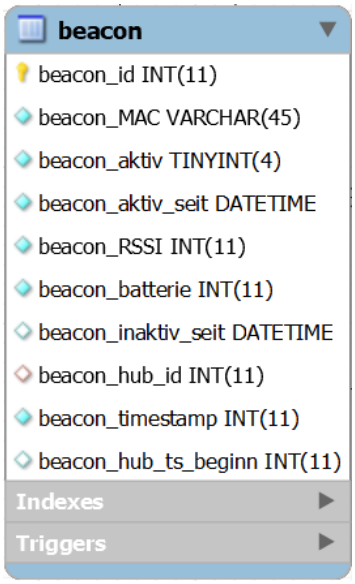


Abb. 13: „beacon“-Tabelle

Die Tabelle beginnt mit einem Eintrag für den Primary-Key (PK), mit dem jeder Tabelleneintrag eindeutig identifiziert werden kann. Dieser ist auf sog. Auto-Increment eingestellt, erhöht sich also mit jedem neuen Eintrag automatisch um 1. Nach dem PK folgen die MAC-Adresse und zwei Einträge, die Rückmeldung darüber geben, ob und seit wann ein Beacon im Betrieb der Einrichtung aktiv ist. Einige Tabellenspalten sind dabei auf „NotNull“ eingestellt, um bereits per Definition der Tabelle eine korrekte Befüllung der Datenfelder sicherzustellen.

Über die MAC-Adresse des Beacons hinaus, liefert die MQ auch die RSSI (Empfangsstärke), den zugehörigen Hub, den Batteriestand und den Zeitstempel des Empfangs. Diese Daten werden nach Verarbeitung in den gleichnamigen Tabellenfeldern gespeichert, die „beacon_hub_id“ verweist als Foreign Key (FK) auf die „hub_id“ des zugehörigen Hubs. So lässt sich zu jedem Zeitpunkt identifizieren, um welchen Beacon es sich handelt, welchem Hub er zugeordnet ist, ob die Batterie ausreichend geladen ist, wann seine Daten zuletzt aktualisiert wurden und wie hoch die zuletzt gemessene Empfangsstärke war. Der Eintrag „beacon_hub_ts_begin“ wird nur bei erstmaliger Zuordnung eines Beacons zu einem Hub aktualisiert, um diesen Datenpunkt für die Historienerstellung festzuhalten.

Alle weiteren Tabellen und Abhängigkeiten sind in Abb. 14 als ERM dokumentiert.

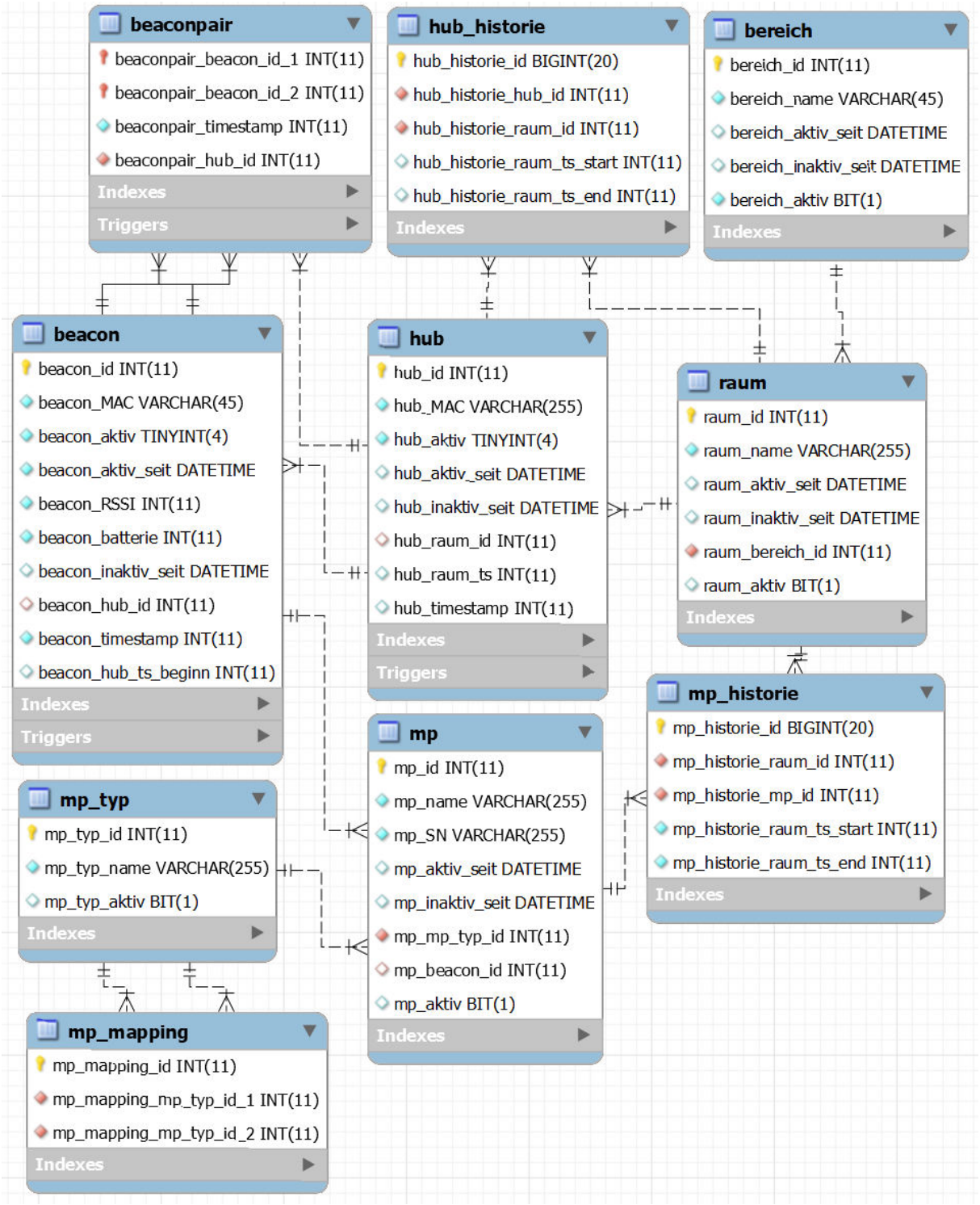


Abb. 14: Entity-Relationship-Modell der Maria-DB

Sowohl der Microservice als auch die Webanwendungen sollen ihre Daten aus der Maria-DB beziehen und in diese schreiben. Diese DB soll alle Informationen enthalten, die benötigt werden, um den Ist-Zustand aller Geräte und Informationen festzustellen. Der vollständige SQL-Code, mit dem die DB initialisiert werden kann, befindet sich im Anhang unter A9.

Wie in der Abb. 14 zu sehen ist, wurde die Historienspeicherung exemplarisch für die Beacons und Hubs umgesetzt. Die Historienpflege anderer Komponenten würde sich nahezu identisch gestalten, hat für die Grundfunktionalität der Anwendung jedoch keine Relevanz.

Entwicklung des Python-Microservices

Nach erfolgter Erstellung der Maria-DB wird anknüpfend das weiterverarbeitende Python-Programm entwickelt. Aus Gründen der Übersichtlichkeit wurde sich dafür entschieden, mit Ausnahme der Überprüfung der bestehenden Beaconpaare, alle Funktionen innerhalb eines einzelnen Python-Skripts umzusetzen. Für eine performantere Datenverarbeitung würde es sich anbieten, einige Funktionen in separate Programme aufzuteilen oder mehrere Instanzen des Microservices zu betreiben.

Das Programm beginnt mit dem initialen Laden von Bibliotheken und dem Einrichten der DB-, MQTT- und Memcache-Verbindungen. Alle Adressen und Passwörter befinden sich in einer umgebungsabhängigen Konfiguration, der „Config.env“ Datei. Codeabschnitte geringerer Relevanz wurden zur besseren Übersicht mit […] gekürzt.

```
import paho.mqtt.client as mqtt  # für die MQTT Verbindung
import mysql.connector  # für die Datenbank-Verbindung
import time  # zur Nutzung von Methoden zur Zeitberechnung
import json  # zum einfachen Separieren der MQTT Nachricht
from pymemcache.client import base  # eine Bibliothek für serverseitigen memcache
import sys  # um Fehlercode bei einem Abbruch zurückzugeben
import os
from dotenv import load_dotenv, find_dotenv

# Laden der ungebungsabhängigen Konfigurationsparameter
load_dotenv(find_dotenv('Config.env'))
# Konfiguration der MySQL-Datenbankverbindung, des Memcached und des MQTT
db_config = {
    'host': os.getenv('DB_HOST'),
    'port': int(os.getenv('DB_PORT')),
    'user': os.getenv('DB_USER'),
    'password': os.getenv('DB_PASSWORD'),
    'database': os.getenv('DB_DATABASE')
}
db_connection = None    # Globale Variable zum Halten des DB connectors
# MQTT Broker Konfiguration
memcache_server = os.getenv('MEMCACHE_SERVER')
memcache_port = int(os.getenv('MEMCACHE_PORT'))
mqtt_server = os.getenv('MQTT_SERVER')
mqtt_port = int(os.getenv('MQTT_PORT'))
mqtt_topic = os.getenv('MQTT_TOPIC')
#client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)
client = mqtt.Client()
client.username_pw_set(os.getenv('MQTT_USER'), os.getenv('MQTT_PW'))
# Raumwechsel bei niedrigerem RSSI nach Zeitlücke
room_timegap = int(os.getenv('ROOM_TIMEGAP'))
pairing_timegap = int(os.getenv('TIMEGAP'))
db_update_cycle_beacon = int(os.getenv('DB_UPDATE_CYCLE_BEACON'))
db_update_cycle_hub = int(os.getenv('DB_UPDATE_CYCLE_HUB'))
```

Nach dem Setzen der Verbindungen wird definiert, dass ein Serialisieren und Deserialisieren bei einem Datenaustausch mit dem Memcache erfolgen soll. Dies erleichtert die Verarbeitung der aus dem Memcache gelesenen Daten. Außerdem werden ein Debugging eingerichtet und die Verbindung zu

Memcache und DB-Server aufgebaut. Das einstellbare Debugging-Level erlaubt es, bestimmte Ausgaben in der Programmkonsole ein- oder auszuschalten.

```
36. # Debug Level und Filter Funktion, Debug Meldungen aus dem Text können bitweise auf Leveln
       definiert werden
37. debug_level = int(os.getenv('DEBUG_LEVEL'))
38. debug_count = 0
39. def do_debug(k):
40.     temp = debug_level >> (k - 1)
41.     return temp & 1 != 0
42.
43. # Serializer zur Memcache Nutzung, erleichten den Zugriff (Strings/Arrays statt bytes).
44. def json_serializer(key, value):
45.     return json.dumps(value), 1
46. def json_deserializer(key, value, flags):
47.     if flags == 1:
48.         return json.loads(value)
49.     raise Exception("Unknown serialization format")
50.
51. # Verbindung zur Datenbank herstellen. Hier als Methode, da
52. def connect_to_db():
53.     try:
54.         return mysql.connector.connect(**db_config)
55.     except mysql.connector.Error as err:
56.         print(f"Error connecting to MySQL: {err}")
57.         return None
58.
59. # Optimierungsbedarf: Globale Datenbankverbindung, einmal geöffnet für die Laufzeit des
       Programms
60. db_connection = connect_to_db()
61.
62. # Aufbau der globalen Verbindung zum memcached
63. memcache = base.Client((memcache_server, memcache_port), serializer=json_serializer,
                             deserializer=json_deserializer)
```

Einige der im Programm definierten Funktionen benötigen Informationen über die Beacons und Beaconpaare, die in der Maria-DB hinterlegt sind. Diese werden zunächst initial geladen und in dem Memcache gespeichert, damit alle nachfolgenden Funktionen ohne DB-Anfrage auf die Informationen zugreifen können. Das Speichern sollte dabei möglichst in einem eindimensionalen Array, mit vorab festgelegten Indizes und unter einem eindeutigen, bekannten Key erfolgen. Nur so kann gewährleistet werden, dass auf die Daten schnell und effizient zugegriffen werden kann. Eine Speichermethode, die beim Auslesen der Daten beispielsweise eine Schleife benötigte, nähme erheblich mehr Zeit in Anspruch.

Der Eintrag eines Beacons in den Memcache erfolgt anhand der MAC-Adresse als Key und folgt diesem Schema:

```
memcache.set(baecon_MAC, beacon_id, beacon_hub_id, beacon_rssi, beacon_timestamp_aktualisierung,
beacon_hub_timestamp_begin, beacon_batterie, beacon_mp_typ, beacon_last_db_sync)
```

Der Eintrag der Paarungen eines Beacons (eine 1:1-n Relation) wird wie folgt vorgenommen:

```
memcache.set(‚beacon_mapping_‘ {beacon_id}, [beaon_id1, beacon_id2, …])
```

Da ein MP mit mehreren anderen MPs gepaart sein kann, verweist ein Mapping-Eintrag zu einer Beacon-ID auf ein Array mit allen Paarungen zu dieser ID.

Nachfolgend sind Ausschnitte aus der Funktion „load_inital_data“ zu sehen.

```
65. # Initalisieren von relevanten Daten bei Programmstart aus der Maria-DB
66. def load_initial_data():
67.     # Initialisieren von Beacondaten mit und ohne MP
68.     try:
69.         cursor = db_connection.cursor(dictionary=True)
70.         query = ''' SELECT * from beacon_left_join_mp '''
71.         cursor.execute(query)
72.         beacon_mp_cache = cursor.fetchall()
73.         for data_values in beacon_mp_cache:
74.             if isinstance(data_values, dict):
75.                 beacon_id = data_values['beacon_id']
76.                 beacon_hub_id = data_values['beacon_hub_id']
77.                 beacon_rssi = data_values['beacon_RSSI']
78.                 beacon_timestamp = data_values['beacon_timestamp']
79.                 beacon_hub_ts_beginn = data_values['beacon_hub_ts_beginn']
80.                 beacon_batterie = data_values['beacon_batterie']
81.                 mp_mp_typ_id = data_values['mp_mp_typ_id']  # Null, falls noch nicht
                                                                  zugeordnet
82.                 beacon_MAC = data_values['beacon_MAC']
83.                 beacon_db_sync_timestamp = data_values['beacon_timestamp']
84.                 beacon_neudaten = (beacon_id, beacon_hub_id, beacon_rssi, beacon_timestamp,
                    beacon_hub_ts_beginn, beacon_batterie, mp_mp_typ_id,
                    beacon_db_sync_timestamp)
85.                 # Falls neu oder aktueller als im Memcache, dort aktualisieren
86.                 if memcache.get(beacon_MAC) is None or memcache.get(beacon_MAC)[7] <=
                       beacon_timestamp:
87.                     beacon_initial_anlegen(beacon_MAC, beacon_neudaten)
88.                 else:
89.                     beacon_aktualisieren(beacon_MAC, beacon_neudaten)
90.             else:
91.                 print(f"Unexpected data format: {data_values}")
92.     except Exception as e:
93.         print(f"An error occurred: {e}")
[…]
```

```
[…]
112.     # Initialisieren von bereits vorhandenen Beaconpaaren
113.     try:
114.         cursor = db_connection.cursor(dictionary=True)
115.         query = ''' SELECT * FROM beaconpair '''
116.         cursor.execute(query)
117.         mp_beaconpair_cache = cursor.fetchall()
118.         # Aktualisieren des Memcache für alle Paarungen
119.         for data_values in mp_beaconpair_cache:
120.             if isinstance(data_values, dict):
121.                 beaconpair_beacon_id_1 = int(data_values['beaconpair_beacon_id_1'])
122.                 beaconpair_beacon_id_2 = int(data_values['beaconpair_beacon_id_2'])
123.                 update_beaconpairs(beaconpair_beacon_id_1, beaconpair_beacon_id_2)
124.             else:
125.                 print(f"Unexpected data format: {data_values}")
126.     except mysql.connector.Error as err:
127.         print(f"Error loading initial data: {err}")
```

Die Funktion lädt, wie gefordert, bei Programmstart alle Beacondaten inklusive der zugeordneten MP-Typen und darüber hinaus die zulässigen MP-Typ-Paarungen sowie die bereits gepaarten Beacons aus der Maria-DB in den Memcache. Da der Memcache im Falle eines gewollten oder ungewollten Abbruchs und Neustarts der Verarbeitung Daten enthalten könnte, die noch nicht in der DB gespeichert sind, ist ein Abgleich der hinterlegten Zeitstempel notwendig. Ist der Eintrag in der DB älter, bleibt dieser im Memcache erhalten. Ist jedoch der Eintrag im Memcache älter oder fehlt, wird er durch die Daten aus der DB ersetzt. Die Verwendung der von Python zur Verfügung gestellten „try“- und

„except“- Blöcke dienen dem allgemeinen Exception Handling. Tritt ein Fehler auf, wird die jeweilige Funktion unterbrochen und der Fehlercode in der Programmkonsole ausgegeben.

Alle initial ladenden Programmteile übergeben ihre Daten an Funktionen zur weiteren Verarbeitung. Im Fall der Beispiele sind dies die Funktionen „beacon_initial_anlegen“ und „update_beaconpairs“. Diese dienen der Datenspeicherung im Memcache und sind wie folgt definiert:

```
129. def beacon_initial_anlegen(beacon_MAC, beacon_neudaten):
130.     memcache.set(beacon_MAC, beacon_neudaten)
131.     return None
```

```
156. # Funktion um ein Beaconpaar einzutragen
157. # Notation: beaconpairs_BeaconID: [MappedBeaconID1, MappendBeacon_ID2, MappedBeaconID...]
158. def update_beaconpairs(beacon_id_1, beacon_id_2):
159.     # Für jedes Beacon die jeweils andere ID in den Pairs hinzufügen
160.     beaconpairs_key = f'beaconpairs_{beacon_id_1}'
161.     # Bestehende Paarungen holen.
162.     beaconpairs_temp = memcache.get(beaconpairs_key)
163.     if beaconpairs_temp is None: beaconpairs_temp = []
164.     # Neue ID anhängen und Speichern
165.     if beacon_id_2 not in beaconpairs_temp: beaconpairs_temp.append(beacon_id_2)
166.     memcache.set(beaconpairs_key, beaconpairs_temp)
167.     # Für zweite ID entsprechend
168.     beaconpairs_key = f'beaconpairs_{beacon_id_2}'
169.     beaconpairs_temp = memcache.get(beaconpairs_key)
170.     if beaconpairs_temp is None: beaconpairs_temp = []
171.     if beacon_id_1 not in beaconpairs_temp: beaconpairs_temp.append(beacon_id_1)
172.     memcache.set(beaconpairs_key, beaconpairs_temp)
173.
174.     # Hinzufügen der Beaconpaare in Array für Crawler
175.     tmp_pair_1 = f"{beacon_id_1},{beacon_id_2}"
176.     tmp_pair_2 = f"{beacon_id_2},{beacon_id_1}"
177.     tmp_beaconpairs = memcache.get('beaconpairs')
178.     # Gibt es noch keine Paare? Dann leer.
179.     if tmp_beaconpairs is None: tmp_beaconpairs = []
180.     # Nur hinzufügen, wenn nicht schon existent
181.     if tmp_pair_1 not in tmp_beaconpairs and tmp_pair_2 not in tmp_beaconpairs:
182.         tmp_beaconpairs.append(f"{beacon_id_1},{beacon_id_2}")
183.         memcache.set('beaconpairs', tmp_beaconpairs)
184.     if do_debug(2): print(f"Beaconpair Data von {beacon_id_2} in Memcache:
                              {memcache.get(beaconpairs_key)}")
```

Mit dem vorherigen Code-Abschnitt sind die Vorbereitungen für die nachfolgenden Funktionen erfolgt. Es kann mit der Umsetzung der Datenverarbeitung begonnen werden. Der Code in den Zeilen Nr. 248 bis 256 dient der Vorbereitung auf das Python-Programm zur Bereinigung von Beaconpaaren und wird später näher erläutert.

Die erforderliche Chronologie der Datenverarbeitung setzt voraus, dass die Nachrichten aus der MQ chronologisch verarbeitet werden. Dafür stellt die Python-Bibliothek ‚paho.mqtt‘ die Funktion „client.on_message“ zur Verfügung. Diese erlaubt es, über den Aufruf in der „main“ Funktion, eine eigens definierte Funktion immer dann aufzurufen, wenn eine MQTT-Nachricht innerhalb des abonnierten Topics[4] empfangen wird. Die dafür vorgesehene Funktion wird „on_mqtt_message“ genannt.

[4] Ein MQTT-Topic ist eine Art Kanal, über den Nachrichten gesendet und empfangen werden. Es dient dazu, Nachrichten gezielt an die richtigen Empfänger zu leiten.

Sie stellt bei Aufruf sicher, dass die DB-Verbindung noch besteht und bekommt den Inhalt der empfangenen Nachricht übergeben. Dieser kann problemlos an eine Folgefunktion übergeben werden, die die Datenverarbeitung beginnt.

```
487. def on_mqtt_message(client, userdata, msg):
488.     global db_connection
489.     # Sicherstellen, dass die Datenbankverbindung noch steht, Neuaufbau falls nicht
490.     loop = 0
491.     while db_connection.is_connected() == False:
492.         try:
493.             db_connection = mysql.connector.connect(**db_config)
494.         except mysql.connector.Error as err:
495.             time.sleep(5)
496.         loop = loop + 1
497.         if loop > 50:
498.             exit(51)    # Beendigen des Programms nach 50 erfolglosen Verbindungsversuchen.
499.     # Verbindung zur Datenbank steht, die Verarbeitung der Nachricht kann starten
500.     message = msg.payload.decode()
501.     process_message(message)
```

Die Funktion „process_message“ erhält die Nachricht und teilt diese zunächst in die einzelnen Name-Value Paare auf. Die JSON-Nachrichten folgen dabei, wie in Kap. 4.5. „Modifizierter MQTT-Broker“ beschrieben, immer diesem Schema:

```
„{"time":"1716645400","data":"layer=2, MAC_ROOM=64:E8:33:88:38:14,
MAC_SENSOR=DD:EF:24:5A:28:D4, BATT=99, BUTTON=1, RSSI=240"}“
```

```
389. # Vorverarbeiten der MQTT Nachricht: Zerlegen in die einzelnen Werte, Weitergabe an die
      Verarbeitung, Laufzeitmessung
390. def process_message(message):
391.     global debug_count
392.     start_time = time.time() # Festhalten des Starts der Verarbeitung zur Laufzeitmessung
393.     # Startparameter und Positionen von Werten in den Memcached Datenstrukturen
394.     hub_id = 0  # 0 = kein Hub gefunden. -> keine weitere Verarbeitung.
395.     beacon_altdaten = None  # None = keine Altdaten/ kein Beacon Datensatz gefunden
[…]
405.     # Verarbeiten der Nachricht
406.     try:
407.         # Konvertierung des JSON-Strings in ein Dictionary
408.         message_dict = json.loads(message)
409.         # Direkter Zugriff auf den Zeitpunkt, zu dem die Nachricht im MQTT angekommen ist
410.         timestamp = int(message_dict["time"])
411.         # Verarbeitung des data-Strings: Auslesen und den Text in Einzelwerte aufteilen
412.         data_parts = message_dict["data"].split(", ")
413.         data_values = {part.split('=')[0]: part.split('=')[1] for part in data_parts}
414.         # Direkter Zugriff auf die Werte der MQTT Nachricht
415.         hub_MAC = data_values.get('MAC_ROOM', None)
416.         beacon_MAC = data_values.get('MAC_SENSOR', None)
417.         beacon_batterie = int(data_values.get('BATT', 0))
418.         beacon_taster = int(data_values.get('BUTTON', 0))
419.         beacon_rssi = int(data_values.get('RSSI', 0))
```

Nach der Aufteilung der Daten soll mit ihnen der im Flowchart aus Abb. 12 dokumentierte Ablauf umgesetzt werden. Je nach empfangenen Daten und Ergebnis einiger If-Abfragen, wird dabei eine von drei verarbeitenden Funktionen aufgerufen. Diese stellen die drei im Flowchart abgebildeten Fälle dar. Sie beinhalten die für die Ortung und Zuordnung benötigten Verarbeitungsschritte und aktualisieren die Beacondaten im Memcache und bei Bedarf in der DB.

```
421.          # Verarbeitet wird nur bei gültigem Hub!
422.          # Hub-aktiv-Timestamp aktualisieren. Wenn es den Hub gibt, ist nach Aufruf die
               hub_id > 0.
423.          if hub_MAC is not None:
424.              hub_id = hub_aktualisieren(hub_MAC, timestamp)
425.          # Wenn es den Hub gibt: Holen der letzten Meldungsdaten des beacons. Wenn es den
               beacon gibt, ist nach Aufruf die beacon_id > 0
426.          if hub_id > 0 and beacon_MAC is not None:
427.              beacon_altdaten = beacon_altdaten_holen(beacon_MAC, timestamp)
428.          else:
429.              if do_debug(4): print("Unberkannte Beacondaten: ", data_parts)
430.          # Wenn es den beacon und Hub in der Datenbank gibt, verarbeite die neuen Daten
431.          if beacon_altdaten is not None:
432.              if do_debug(2): print("beacon Altdaten: ", beacon_altdaten)
433.              # Wenn Timestamp der neuen Daten noch nicht im Cache, akualisiere,
434.              # sonst ignorieren, da Nachrict des Beacons vor Reconnect schon prozessiert
                    wurde
435.              if timestamp != beacon_altdaten[index_beacon_timestamp]:
436.                  # Wenn noch kein Hub zugewiesen, wurde ist noch kein Zeitstempel der
                        Zuweisung in den Altdaten, daher aktuellen TS nutzen
437.                  # Wenn schon ein Hub zugewiesen wurde, aber ein Datensatz mit anderm Hub und
                        höherem RSSI kommt, ebenfalls aktuellen TS nutzen
438.                  # Ansonsten Timestamp der Erstverbindung weiterführen, ebenso der letzten
                        Datenbanksynchronisation
439.                  if beacon_altdaten[index_beacon_hub_id] is None:
440.                      # Case 1: Beacon wurde noch nie in einem Hub gefunden. Daten eintragen.
441.                      beacon_neudaten = [beacon_altdaten[index_beacon_id], hub_id,
                           beacon_rssi, timestamp, timestamp, \
442.                                         beacon_batterie,
                          beacon_altdaten[index_beacon_mp_typ], timestamp]
443.                      beacon_erstspeicherung(beacon_MAC, beacon_neudaten)
444.                      if do_debug(2): print("Ersteintrag Beacon erfolgt.")
445.                      if do_debug(2): print("Beacon Neudaten sind: ", beacon_neudaten)
446.                  else:
447.                      if beacon_altdaten[index_beacon_hub_id] == hub_id:
448.                          # Case 2: Hub gleich? Dann RSSI, Batterie und Zeit des beacons
                              aktualisieren.
449.                          # Timestamp der Erstverbindung halten, ebenso der letzten DB-
                              Synchronisation
450.                          beacon_neudaten = [beacon_altdaten[index_beacon_id], hub_id,
                              beacon_rssi, timestamp, \
451.                          beacon_altdaten[index_beacon_hub_timestamp_beginn],
                              beacon_batterie,\
452.                          beacon_altdaten[index_beacon_mp_typ], \
453.                          beacon_altdaten[index_beacon_db_sync_timestamp]]
454.                          beacon_aktualisieren(beacon_MAC, beacon_neudaten)
455.                          if do_debug(2): print("Beacon aktualisiert")
456.                          if do_debug(2): print("Beacon Neudaten sind: ", beacon_neudaten)
457.                          # Taster gedrückt? Aufforderung zur Verknüpfung.
458.                          if beacon_taster == 1:
459.                              if do_debug(2): print("Taster wurde gedrückt, Beaconpairing
                                                      eingeleitet")
460.                              beaconpairing(hub_id,beacon_altdaten[index_beacon_mp_typ]
                                 ,beacon_altdaten[index_beacon_id] , timestamp)
461.                      else:
462.                          # Case 3: Wenn neuer Hub mit besserem RSSI
463.                          # oder neuer Hub, aber alter Hub-Eintrag älter als 5 Minuten,
                              aktualisiere Hub.
464.                          if (beacon_altdaten[index_beacon_rssi] < beacon_rssi) or \
465.                              (beacon_altdaten[index_beacon_timestamp] + room_timegap <
                                 timestamp):
466.                              beacon_neudaten = [beacon_altdaten[index_beacon_id], hub_id,
                                 beacon_rssi, \
467.                              timestamp, timestamp, beacon_batterie, \
468.                              beacon_altdaten[index_beacon_mp_typ], timestamp]
469.                              beacon_hubwechsel(beacon_MAC, beacon_neudaten)
470.                              # Überprüfung von Beaconpaaren auf vorliegenden Standortwechsel
471.                              beacon_pairing_hubwechsel(beacon_altdaten[index_beacon_id],
                                 hub_id, timestamp)
472.                              if do_debug(4): print("beacon hat den Hub gewechselt")
```

```
473.                             if do_debug(4): print("beacon Neudaten sind: ", beacon_neudaten)
474.         else:
475.             if do_debug(4): print("Unberkannte Beacondaten: ", data_parts)
476.         if do_debug(2): print(timestamp, data_values)
477.         if do_debug(2): # Messen der Laufzeit einer Verarbeitung
478.             debug_count += 1
479.             print("Extraktion und Schicken der Nachricht ", "{0:05d}".format(debug_count), "
                        hat %.5f" % (time.time() - start_time), " Sekunden gedauert")
480.     except Exception as e:
481.         print(f"Fehler bei der Verarbeitung der Nachricht: {e}") # ohne debug-filter, das
                                                                      darf nicht passieren.
```

Da sich die drei Funktionen (Aufruf in Z. 443, 454, 469) in ihrem Inhalt ähneln, wird an dieser Stelle die dritte exemplarisch für alle dargestellt. Diese ordnet einen Beacon aufgrund höherer Empfangsstärke einem neuen Hub zu, im Flowchart aus Abb. 12 unter Fall 3 darstellt. Tritt dieser Fall ein, wird ebenfalls die Funktion aufgerufen, mit der Beaconpaare aufgrund des Raumwechsels auf ihre Validität geprüft werden (Aufruf in Z. 471). Auf diese Zuordnung wird zum Ende der Entwicklung des Microservices noch genauer eingegangen.

```
273. def beacon_hubwechsel(beacon_MAC, beacon_neudaten):
274.     if do_debug(2): print("beacon zu anderem Hub wechseln (DB und Cache)")
275.     try:
276.         my_cursor = db_connection.cursor()
277.         my_query = 'update beacon set beacon_hub_id = "%(hub_id)s", beacon_RSSI =
                        "%(rssi)s", beacon_timestamp = "%(ts)s", beacon_hub_ts_beginn =
                        "%(ts)s", beacon_batterie = "%(bat)s" where beacon_id = "%(id)s"' % {
278.             'hub_id': beacon_neudaten[1], 'rssi': beacon_neudaten[2], 'ts':
                          beacon_neudaten[3], 'id': beacon_neudaten[0], 'bat':
                          beacon_neudaten[5]}
279.         my_cursor.execute(my_query)
280.         db_connection.commit()
281.         memcache.set(beacon_MAC, beacon_neudaten)
282.         if do_debug(2): print("beacon im Cache/in DB aktualisiert.")
283.     except mysql.connector.Error as err:
284.         print(f"Error updating beacon hub data in MySQL: {err}")
```

Die dargestellte Funktion zum Hubwechsel enthält eine DB-Anfrage, in der alle relevanten Daten innerhalb eines „UPDATE“ Befehls an die DB übergeben werden. Abschließend werden die Daten im Memcache aktualisiert.

Die Zuordnung von Beacons zu Beacon-Paaren, und entsprechend die Zuordnung von MP zueinander, wird bei gedrücktem Knopf und dem Eintreten von Fall 2 ausgeführt (Z. 460). Fall 2 bedeutet, dass ein Beacon nach Prüfen der Daten demselben Hub zugeordnet geblieben ist. Damit wird sichergestellt, dass sich der Beacon aktuell mit aller Wahrscheinlichkeit nicht in Bewegung befindet, da dies ansonsten zu falschen Paarungen führen könnte.

Innerhalb der Funktion „beaconpairing“ (Aufruf in Z. 460) erfolgt die Prüfung auf mögliche Partnerbeacons. Diese findet anhand der zugeordneten Hubs und MP-Typen sowie den anfangs initialisierten zulässigen MP-Kombinationen statt.

```
306. # Funktion, die zwei zulässige Beacons miteinander paart und die Paarung im Memcache und der
       DB einträgt
307. # Notation des Caches: beaconpairing_cache_hub_id_mp_typ: {beacon_id , timestamp}
308. def beaconpairing(hub_id, mp_typ_id, beacon_id, timestamp):
```

```
309.     try:
310.         beacon_mapping_mp_typen = memcache.get(f"mp_typ_mapping_{mp_typ_id}")
311.         # Darf dieses Beacon mit anderen Typen gepaart werden?
312.         if beacon_mapping_mp_typen is not None:
313.             array_length = len(beacon_mapping_mp_typen)
314.             i = 0
315.             # Für jeden Typen nach Partnern suchen, die auf ein Paaren warten
316.             while i < array_length:
317.                 beaconpairing_temp_key =
                    f"beaconpairing_cache_{hub_id}_{beacon_mapping_mp_typen[i]}"
318.                 moeglicher_partner = memcache.get(beaconpairing_temp_key)
319.                 # Wenn Partner gefunden und Anfrage noch nicht zu alt
320.                 if moeglicher_partner is not None:
321.                     if timestamp - moeglicher_partner[1] <= pairing_timegap:
322.                         # Eintragen des Paares in Memcached und Datenbank, Löschen der
                            temporären Paarungsanfrage
323.                         update_beaconpairs(beacon_id, moeglicher_partner[0])
324.                         beaconpairing_DB_insert(beacon_id, moeglicher_partner[0], hub_id,
                            timestamp)
325.                         memcache.delete(beaconpairing_temp_key)
326.                         return
327.                     else:
328.                         print("Fehler bei Löschen von Cachedaten älter " +
                                str(pairing_timegap) + " Sekunden")
329.                         return
330.                 i += 1
331.             # Wenn Funktion bis hierhin kein Return ausgelöst hat, wurde kein Beacon zum
                 Mapping gefunden.
332.             # Also folgt ein Eintrag in den Cache, der pairing_timegap Sekunden gespeichert
                 wird
333.             # Der Timestamp ist lediglich eine Sicherheitsmaßnahme
334.             beaconpairing_temp_key = f"beaconpairing_cache_{hub_id}_{mp_typ_id}"
335.             beaconpairing_temp_value = (beacon_id, timestamp)
336.             memcache.set(beaconpairing_temp_key, beaconpairing_temp_value, pairing_timegap)
337.         else:
338.             if do_debug(2):
339.                 print("Zu diesem MP gibt es keinen zugelassenen Partner")
340.             return
341.
342.     except Exception as e:
343.         print(f"An error occurred: {e}")
```

Nach Aufruf der Funktion wird ein Eintrag im Memcache getätigt, mit dem der auslösende Beacon als suchend deklariert ist. Wenn nun ein weiterer Beacon mit gedrücktem Knopf empfangen wird, prüft die Funktion, ob dieser mit einem der als suchend deklarierten Beacons gepaart werden darf. Die „beaconpairing“ Funktion (Definition ab Z. 308) arbeitet mit einer Schleife, die alle relevanten Memcache Einträge durchsucht. Wie oben beschrieben, ist dies keine optimierte Art der Datenspeicherung, weshalb die gehaltene Datenmenge so gering wie möglich sein sollte. Dies wird gewährleistet, indem der Eintrag eines suchenden Beacons im Memcache automatisch gelöscht wird, sobald er älter als 30 Sekunden ist (in der „config.env“ unter „pairing_timegap“). Das Beschränken der Suche nach Cacheeinträgen mit zulässigen MP-Typen am selben Hub grenzt die Datenmenge ebenfalls ein und verhindert darüber hinaus eine Fehlzuordnung.

Sobald ein Beaconpaar eingetragen werden soll, wird automatisch die o. g. Funktion „update_beaconpairs“ aufgerufen. Diese übernimmt das Aktualisieren der Beaconpaare im Memcache und der DB.

Nach der erfolgreichen Umsetzung der Bildung von Beaconpaaren muss ferner die Überprüfung erfolgen, ob diese Paare bei einem Raumwechsel aufgelöst werden müssen. Sobald ein Beacon den Hub

gewechselt hat wird dafür wird die o. g. Funktion „beacon_pairing_hubwechsel“ aufgerufen (Aufruf in Z. 471).

In dieser wird geprüft, ob der Beacon einem Paar zugeordnet ist. Ist dies der Fall, wird im Memcache für jeden mit diesem Beacon gepaarten Beacon ein Eintrag erzeugt, welcher den kritischen Zustand der Paarung markiert. Diese Einträge werden von einem separaten Python-Skript zur Überprüfung von Beaconpaaren analysiert und führen, bei Bedarf, zur Entkopplung.

```
346. # Funktion, die ein Beaconpaar als 'kritisch' markiert, wenn einer der Beacons den Hub/Raum
      gewechselt hat
347. def beacon_pairing_hubwechsel(beacon_id, new_hub_id, timestamp):
348.     try:
349.         beaconpair_key = f'beaconpairs_{beacon_id}'
350.         beaconpairs = memcache.get(beaconpair_key)
351.         # Gibt es Beacon-Paare für das zu prüfende Beacon? Wenn nein: Ende.
352.         if not beaconpairs:
353.             if do_debug(2): print(f"Keine Mappings für Beacon ID {beacon_id} gefunden.")
354.             return
355.         # Für jede ID, die in den destehenden Paarungen angegeben is, eine Prüfung
           durchführen
356.         for mapped_beacon_id in beaconpairs:
357.             # Vorbereitung der Key-Namen, Holen der Paarungen zu den IDs, Prüfen
358.             beaconpair_krit_key_1 = f'beacon_mapping_krit_{mapped_beacon_id}_{beacon_id}'
359.             beaconpair_krit_key_2 = f'beacon_mapping_krit_{beacon_id}_{mapped_beacon_id}'
360.             beaconpair_krit_data_1 = memcache.get(beaconpair_krit_key_1)
361.             beaconpair_krit_data_2 = memcache.get(beaconpair_krit_key_2)
362.             if beaconpair_krit_data_1:
363.                 # Wenn auch das Paarungsziel auf neuem Hub bekannt, kritischen Eintrag
                   löschen
364.                 if beaconpair_krit_data_1[1] == new_hub_id:
365.                     memcache.delete(beaconpair_krit_key_1)
366.                     memcache.delete(beaconpair_krit_key_2)
367.                     if do_debug(2):
368.                         print(f"Gefährdungseintrag {beaconpair_krit_key_1} und
                                {beaconpair_krit_key_2} gelöscht.")
369.                 else:
370.                     # Timestamp des bestehenden Eintrags aktualisieren
371.                     memcache.set(beaconpair_krit_key_2, [timestamp, new_hub_id])
372.                     if do_debug(2): print(f"Gefährdungseintrag {beaconpair_krit_key_2}
                                          aktualisiert.")
373.             elif beaconpair_krit_data_2:
374.                 if beaconpair_krit_data_2[1] == new_hub_id:
375.                     memcache.delete(beaconpair_krit_key_1)
376.                     memcache.delete(beaconpair_krit_key_2)
377.                     if do_debug(2):
378.                         print(f"Gefährdungseintrag {beaconpair_krit_key_1} und
                                {beaconpair_krit_key_2} gelöscht.")
379.                 else:
380.                     memcache.set(beaconpair_krit_key_1, [timestamp, new_hub_id])
381.                     if do_debug(2): print(f"Gefährdungseintrag {beaconpair_krit_key_1}
                                          aktualisiert.")
382.             else:
383.                 # Es gibt noch keinen Gefährdungseintrag. Erster Beacon einer Paarung, der
                    einen anderen Raum meldet.
384.                 memcache.set(beaconpair_krit_key_1, [timestamp, new_hub_id])
385.                 if do_debug(2): print(f"Neuer Gefährdungseintrag {beaconpair_krit_key_1}
                                       erstellt.")
386.     except Exception as e:
387.         print(f"An error occurred: {e}")
```

In der „main“ Funktion werden schließlich alle Funktionen aufgerufen, die einen Erstaufruf benötigen. Darunter befinden sich die Funktionen der „paho.mqtt“ Bibliothek, ein Testen der Verfügbarkeit des Memcaches und die Funktion, die alle DB-Daten im Memcache initialisiert.

```
519. def main():
520.     # Prüfen, ob memcache und Datenbank verbunden sind
521.     try:
522.         memcache.set('some_key', 'some value')
523.     except Exception as e:
524.         sys.exit(50) # erfolgt, wenn obiger Befehl fehlschlägt / memchaed nicht verfügbar
                         ist
525.     # memcache.flush_all() # nur zum Testen mit einem leeren memcached
526.     # Verbindung zur Datenbank herstellen. Die Connection wird in der globalen
        'db_connection' gehalten.
527.     try:
528.         db_connection = mysql.connector.connect(**db_config)
529.     except mysql.connector.Error as err:
530.         exit(1) # Beenden des Programms mit dem Returnwert 1. Ohne Datenbank ist keine
                    Verarbeitung möglich.
531.     # Aufbau der initialen Verbindung zum MQTT. Da dieser durchaus neu Starten kann, wird
          diese Verbindung je nach Nutzung wiederhergestellt.
532.     # memcache.flush_all()
533.     # Vorbefüllung des memcache mit den letzten Daten aus der Datenbank
534.     load_initial_data()
535.     # Zuweisung der Methoden zur MQTT Verarbeitung
536.     client.on_connect = on_mqtt_connect          # Bei Aufbau einer Verbindung
537.     client.on_message = on_mqtt_message          # Bei Eingang einer Nachricht auf dem
                                                      konfigurierten Topic
538.     client.on_disconnect = on_mqtt_disconnect    # Bei (i.B. unerwarteten) Verlust der
                                                      Verbindung
539.
540.     # Starten der Verbindung zum MQTT - und damit Abarbeiten der Nachrichten
541.     while True:
542.         try:
543.             client.connect(mqtt_server, mqtt_port, 60)
544.             client.loop_start()
545.             # Einmal in der Schleife wird bei Wegfall der MQTT Verbindung die für diesen
                  Fall konfigurierten Methode zum Re-Connect aufgerufen
546.             while True:
547.                 time.sleep(1)
548.         except Exception as e:
549.             if do_debug(2): print(f"Fehler bei der Verbindung zum MQTT-Broker: {e}.
                                      Versuche, in 5 Sekunden erneut zu verbinden...")
550.             time.sleep(5)
551.
552. main()
```

Zuletzt folgt das Programm zur Überprüfung der gebildeten Beaconpaare. Dieses ruft ebenfalls zunächst einige Bibliotheken auf und verbindet sich mit der Maria-DB und dem Microservice.

```
 1. import mysql.connector
 2. import time
 3. from pymemcache.client import base
 4. import json
 5. import sys
 6. import os
 7. from dotenv import load_dotenv, find_dotenv
 8.
 9. # Laden der ungebungsabhängigen Konfigurationsparameter
10. load_dotenv(find_dotenv('Config.env'))
11. # Konfiguration der MySQL-Datenbankverbindung und ded Memcached
12. db_config = {
13.     'host': os.getenv('DB_HOST'),
14.     'port': int(os.getenv('DB_PORT')),
15.     'user': os.getenv('DB_USER'),
16.     'password': os.getenv('DB_PASSWORD'),
17.     'database': os.getenv('DB_DATABASE')
18. }
19. memcache_server = os.getenv('MEMCACHE_SERVER')
20. memcache_port = os.getenv('MEMCACHE_PORT')
21. db_connection = None    # Globale Variable zum Halten des DB connectors
22. timegap = os.getenv('TIMEGAP')
```

```
23.
24. # Debug Level und Filter Funktion, Debug Meldungen aus dem Text können bitweise auf Leveln
     definiert werden
25. debug_level = 2
26. def do_debug(k):
27.     temp = debug_level >> (k - 1)
28.     return temp & 1 != 0
29.
30. # Serializer zur Memcache Nutzung, erleichten den Zugriff (Strings/Arrays statt bytes).
31. def json_serializer(key, value):
32.     return json.dumps(value), 1
33. def json_deserializer(key, value, flags):
34.     if flags == 1:
35.         return json.loads(value)
36.     raise Exception("Unknown serialization format")
37.
38. # Aufbau der globalen Verbindung zum memcached
39. memcache = base.Client((memcache_server, memcache_port), serializer=json_serializer,
                deserializer=json_deserializer)
```

Nach erfolgreicher Verbindung mit Memcache und DB wird in der ‚main' Funktion alle 60 Sekunden die Funktion zur Überprüfung der Beaconpaare aufgerufen.

```
116. # Hauptprogramm
117. def main():
118.     # Prüfen, ob memcache und Datenbank verbunden sind
119.     try:
120.         memcache.set('some_key', 'some value')
121.     except Exception as e:
122.         sys.exit(50) # erfolgt, wenn obiger Befehl fehlschlägt / memchaed nicht verfügbar
ist
123.     # memcache.flush_all() # nur zum Testen mit einem leeren memcached
124.     # Verbindung zur Datenbank herstellen
125.     try:
126.         db_connection = mysql.connector.connect(**db_config)
127.     except mysql.connector.Error as err:
```

Die Funktion ‚baeconpair_validity_check' ist dabei folgendermaßen definiert:

```
 85. def beaconpaar_validity_check():
 86.     # Holen aller aktuell eingetragenen Beacon-Pairs
 87.     beaconpairs = memcache.get('beaconpairs')
 88.     if do_debug(4): print(beaconpairs)
 89.     # Wenn die Liste nicht leer ist, wird über einen Positionsindex durch die Liste
       iteriert.
 90.     # Der Positionsindex wird ebenfalls zum Löschen eines Eintrags der laufenden Liste
       verwendet.
 91.     if beaconpairs != None:
 92.         array_length = len(beaconpairs) # Bestimmen, wann der Ablauf beendet werden muss
 93.         pairposition = 0 # Die Position des aktuellen Pairs in der Liste. Kein! Hochzählen
                            bei Löschen des aktuellen.
 94.         while pairposition < array_length:
 95.             beacon_id_1 = beaconpairs[pairposition].split(",")[0]
 96.             beacon_id_2 = beaconpairs[pairposition].split(",")[1]
 97.             # Erzeugung der Zugriffsnamen für die kritischen Mapping-Einträge und holen
                 dieser Werte
 98.             krit_key_1 = f'beacon_mapping_krit_{beacon_id_1}_{beacon_id_2}'
 99.             krit_key_2 = f'beacon_mapping_krit_{beacon_id_2}_{beacon_id_1}'
100.             krit_data_1 = memcache.get(krit_key_1)
101.             krit_data_2 = memcache.get(krit_key_2)
102.             # Nutzung der lokalen Zeit zur Prüfung, ob die Zeitgrenze zum Löschen erreicht
ist
103.             current_time = int(time.time())
104.             # Wenn auch nur eines der möglichen zwei Einträge zu alt ist, dann Löschen des
                 Paares
105.             if krit_data_1 and current_time - krit_data_1[0] > 60 or krit_data_2 and
                    current_time - krit_data_2[0] > 60:
```

```
106.                 delete_beaconpair(beacon_id_1, beacon_id_2, pairposition)
107.                 memcache.delete(krit_key_1)
108.                 if do_debug(2):
109.                     print(f"Critical mapping {krit_key_1} dissolved.")
110.                 memcache.delete(krit_key_2)
111.                 if do_debug(2):
112.                     print(f"Critical mapping {krit_key_2} dissolved.")
113.             else:
114.                 pairposition +=1    # nächste Position, da aktuelle nicht gelöscht wurde
```

Sie hat die Aufgabe alle als kritisch markierten Beaconpaare darauf prüfen, ob diese gelöscht werden sollten. Dies erfolgt durch das Auslesen aller im Memcache vorhandenen „beacon_mapping_krit_[…]“-Einträge.

Dazu wird das eingangs beschriebene „beaconpairs“-Array (Z. 256) benötigt, das alle aktuellen Beaconpaare beinhaltet. Ein kritischer Eintrag kann ausschließlich zu einem Beacon erstellt worden sein, der zuvor gepaart wurde. Das bedeutet, dass die in der Schleife zu durchsuchende Datenmenge auch in diesem Fall stark eingegrenzt werden kann. Hierfür wird das Array aus dem Memcache ausgelesen, womit bekannt ist, welche Beacons als kritisch eingetragen sein könnten.

Anschließend folgt die Überprüfung, ob es zu den Beacons kritische Einträge gibt und, wenn ja, wie alt diese sind. Bei Einträgen, die älter als 60 Sekunden sind, wird das Beaconpaar mithilfe der Funktion „delete_beaconpair“ umgehend aufgelöst. Einträge, die jünger sind, werden aufgrund der möglichen asynchronen Übertragung der Beacons nicht gelöscht. Damit soll gewährleistet werden, dass gemeinsam umpositionierte Beacons bei asynchroner Meldung nicht fälschlicherweise getrennt werden.

Zuletzt wird das die Trennung auslösende kritische Mapping aus dem Memcache gelöscht, womit die Trennung der Beacons endgültig erfolgt ist.

Mit diesen beiden Programmen ist die Erstellung des Python-Microservices abgeschlossen. Innerhalb der Programme werden die empfangenen Beacondaten vollständig verarbeitet und die Ergebnisse bei Bedarf an die Maria-DB übergeben. In dieser muss als letzter Schritt der Datenverarbeitung, die Historienerstellung von MP und Hubs umgesetzt werden.

Historienerstellung in der Datenbank

Die Maria-DB bietet die Nutzung von Stored Procedures und Triggern an, mit deren Hilfe die Historienerfassung durch simple Logik innerhalb der DB-Struktur stattfinden und vom DB-Server ausgeführt werden kann (siehe Kap. 3.3. „Anforderungen an Softwarekomponenten“).

In den Tabellen „beacon“ und „hub“ wird ein „AFTER UPDATE“ Trigger gesetzt. Dieser sorgt dafür, dass ein vorgegebener SQL-Code ausgeführt wird, sobald in der Tabelle ein Wert eines Beacons respektive Hubs verändert wurde. Da sich der Code beider Vorgänge im Aufbau gleicht, wird zur Veranschaulichung nur der „AFTER UPDATE“ Trigger des Beacons dargestellt.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `ATSDB`.`beacon_AFTER_UPDATE` AFTER UPDATE ON `beacon`
FOR EACH ROW
BEGIN
	IF OLD.beacon_hub_id != NEW.beacon_hub_id THEN
		IF (select mp_id from mp where mp_beacon_id = OLD.beacon_id) != Null THEN
			-- Trage Beacon-Änderung in Historientabelle ein.
			insert into mp_historie (mp_historie_raum_id, mp_historie_mp_id,
			mp_historie_raum_ts_start, mp_historie_raum_ts_end)

			values ((select hub_raum_id from hub where hub_id =
			OLD.beacon_hub_id), (select mp_id from mp where mp_beacon_id =
			OLD.beacon_id), OLD.beacon_hub_ts_beginn,
			NEW.beacon_hub_ts_beginn);
		END IF;
	END IF;
END
```

Hat sich der zugeordnete Hub verändert und der Beacon ist einem MP zugeordnet, wird der Tabelle „mp_historie“ ein Eintrag hinzugefügt. Der Eintrag enthält die Information, innerhalb welchem Zeitraum sich ein MP in einem Raum befunden hat. Damit ist dieser Eintrag unabhängig von möglichen Änderungen, die Beacons oder Hubs betreffen.

4.7. Entwicklung Webansicht

Die Entwicklung der Webansicht wird, wie beschrieben, mit Vue.js, Nuxt.js und Node.js durchgeführt. Es müssen sowohl das Frontend, das als Webapplikation zur Pflege und Anzeige dient, als auch das Backend, das als API zur DB fungiert, erstellt werden.

Die Architektur setzt primär auf eine Umsetzung mit JavaScript, TypeScript und HTML. Anforderungen, wie das asynchrone Aktualisieren der angezeigten Daten lassen sich damit ebenso umsetzen, wie die Aufgabentrennung der verschiedenen Komponenten.

4.7.1 API

Die Schnittstelle zur DB wird als sog. Representational State Transfer (REST)-Interface umgesetzt. REST bezeichnet einen Softwarearchitekturstil, mit dem Schnittstellen zwischen verschiedenen Systemen beschrieben werden, die HTTP für ihre Kommunikation verwenden. Das REST-Paradigma legt fest, dass beim Datentransfer zwischen zwei Systemen – hier der DB und der nutzenden Anwendung - keine Sitzungsinformationen von dem Datentransfer-Service selbst gespeichert werden. REST ist dabei als Sammlung von Architekturbeschränkungen zu verstehen und stellt kein Protokoll oder Standard dar.

In der Abb. 15 ist die daraus resultierende Architektur der relevanten Komponenten dargestellt.

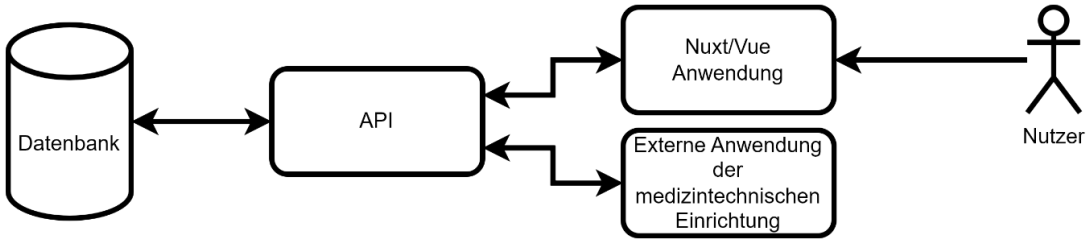


Abb. 15: Anordnung DB, API, Anwendungen

Auf folgende Objekte aus der DB muss zugegriffen werden können:

- Bereiche – Die Bereiche einer medizinischen Einrichtung (Abteilungen, Treppenhäuser, Waren- und Lagerzonen o. ä.)
- Räume – Die Räume eines Bereiches, jedem Raum kann ein Hub zugeordnet werden
- Hubs
- Beacons
- Medizinprodukte
- Medizinprodukt-Typen – Zur Prüfung der Zulässigkeit von Zuordnungen
- Medizinprodukt-Beacon Zuordnungen

Für die einzelnen Aufrufe sind jeweils Methoden zum

- Lesen aller Datensätze einer DB-Tabelle oder eines DB-Views,
- Lesen einzelner Datensätze anhand von Kriterien, wie einer ID,
- Anlegen eines neuen Datensatzes oder Ändern eines bestehenden Datensatzes,
- ggf. Löschen eines Datensatzes

erforderlich.

Die Umsetzung der API erfolgt über Node.js, eine plattformübergreifende Open-Source JavaScript-Laufzeitumgebung. Als äußerst leistungsoptimierte Software ist sie heute die Basis vieler Webserver und insbesondere asynchroner Application-Server. [46]

Für die geplante Anwendung ist, zusätzlich zu einer lokalen Node.js Installation, die Installation ergänzender Module erforderlich. Hier eingesetzt werden „Express“ zum Routen der Anfragen, „MariaDB“ zur Verbindung zur DB, „Body-Parser“ zum strukturierten Zugriff auf die Inhalte einer Anfrage und „dotenv“ für die Verwendung von lokalen „.env“ Dateien zur Systemkonfiguration.

Für eine übersichtliche Struktur der Anwendung werden jeweils alle Aufruf-URLs („Routes“) eines Datentyps bzw. einer Ansicht in jeweils einer eigenen Datei definiert. Damit die Anwendung alle ausführbaren Aufrufe kennt und beantworten kann, werden diese Dateien in die zentrale Anwendung eingebunden. Dies wird innerhalb der „index.js“ (siehe Anhang A11) umgesetzt:

```
require('dotenv').config()
const express = require('express');
const mariadb = require('mariadb');
const bodyParser = require('body-parser');
// Initialisieren und umgebungsspezifische Konfiguration einlesen

const app = express();
const port = process.env.API_PORT;

// Datenbankverbindung herstellen
const pool = mariadb.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
```

```
15.   database: process.env.DB_DATABASE,
16.   connectionLimit: process.env.DB_CONNECTION_LIMIT
17. });
18.
19. // Dantenbank-Pool exportieren, damit die Routes darauf zugreifen können
20. module.exports.pool = pool;
21.
22. // Middleware
23. app.use(bodyParser.json());
24. // Die Routes, die die jeweils aufgerufenen Pfade beantworten
25. app.use(require('./routes/bereich'));
26. app.use(require('./routes/raum'));
27. app.use(require('./routes/hub'));
28. app.use(require('./routes/beacon'));
29. app.use(require('./routes/mp'));
30. app.use(require('./routes/mp_mapping'));
31. app.use(require('./routes/mp_typ'));
32.
33. // Start server
34. app.listen(port, () => {
35.   console.log(`Server is running on port ${port}`);
36. });
```

Node.JS stellt die API mit definiertem Port (API_PORT) 3001 und ohne Angabe eines Hostnames unter localhost:3001 bereit.

Die Logik zur Beantwortung einer Anfrage an die API liegt in den „Routes“ Dateien, hier am Beispiel „Bereich“ aus der bereich.js (siehe Anhang A12):

```
 1. var express = require('express');
 2. var router = express.Router();
 3.
 4. const { pool } = require('../index');  // Datenbank-Pool
 5.
 6. // Bereiche auslesen
 7. router.get('/api/bereich', async (req, res) => {
 8.   let conn;
 9.   try {
10.     conn = await pool.getConnection();
11.     const rows = await conn.query('SELECT * FROM bereich');
12.     res.json(rows);
13.   } catch (err) {
14.     console.error(err);
15.     res.status(500).json({ message: 'Internal server error' });
16.   } finally {
17.     if (conn) conn.release();
18.   }
19. });
[…]
40. // Einen neuen Bereich hinzufügen
41. router.post('/api/bereich', async (req, res) => {
42.   const { bereich_name, bereich_aktiv, bereich_aktiv_seit } = req.body;
43.
44.   if (!bereich_name) {
45.     return res.status(400).json({ message: 'Missing required fields' });
46.   }
47.
48.   let conn;
49.   try {
50.     conn = await pool.getConnection();
51.     await conn.query(
52.       'INSERT INTO bereich (bereich_name, bereich_aktiv, bereich_aktiv_seit) VALUES (?, ?,
?)',
53.       [bereich_name, bereich_aktiv, bereich_aktiv_seit]
54.     );
55.     res.status(201).json({ message: 'Area created successfully' });
56.   } catch (err) {
```

```
57.     console.error(err);
58.     res.status(500).json({ message: 'Internal server error' });
59.   } finally {
60.     if (conn) conn.release();
61.   }
62. });
63.
64. // Einen Bereich aktualisieren
65. router.put('/api/bereich/:id', async (req, res) => {
66.   const { id } = req.params;
67.   const { bereich_name, bereich_aktiv, bereich_aktiv_seit, bereich_inaktiv_seit } =
            req.body;
68.   console.error(req.body);
69.
70.   if (!bereich_name) {
71.       return res.status(400).json({ message: 'Missing required fields' });
72.   }
```

Jede dieser Schnittstellen prüft die Eingangsparameter auf die jeweilig erforderlichen Angaben, führt die Anfrage an die DB aus und liefert die Antwort zurück.

Die gesamte Dateistruktur der API ist in den Abb. 16 und 17 dargestellt.

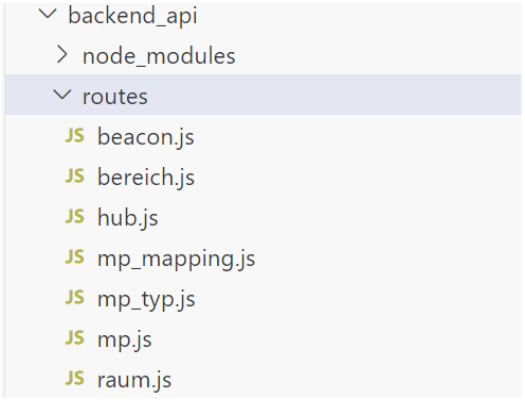


Abb. 16: in der index.js referenzierte "Routes"

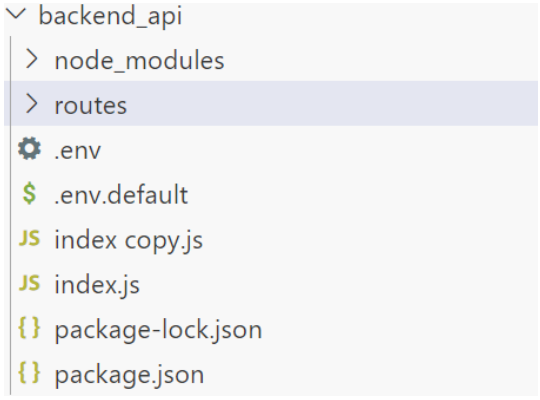


Abb. 17: Struktur des Backends/der API

Die von dem Backend zur Verfügung gestellten Aufrufe werden am Beispiel „Beacon“ erläutert. Sie lauten:

- “router.get('/api/beacon', async (req, res)”
- “router.get('/api/beacon/:id', async (req, res)”
- “router.put('/api/beacon/:id', async (req, res)”
- “router.delete('/api/beacon/:id', async (req, res)”
- “router.post('/api/beacon', async (req, res)”

Je nach Route wird zur Beantwortung einer API-Anfrage eine DB-Anfrage gestellt. Diese greifen entweder direkt auf die entsprechenden Tabellen der DB oder auf Views zu, die die Informationen mehrerer Tabellen bündeln.

Ein Beispiel für die Nutzung eines Views ist der Aufruf von „http://localhost:3001/api/beacon/15“ zum Abruf des Beacons mit der ID 15. Dieser Abruf wird beantwortet mit:


```
{
  "beacon_id": 15,
  "beacon_MAC": "DD:EF:24:5A:28:E4",
  "beacon_aktiv": 1,
  "beacon_aktiv_seit": "2024-06-03T22:00:00.000Z",
  "beacon_inaktiv_seit": null,
  "beacon_timestamp": 1717579369,
  "beacon_hub_ts_beginn": 1717579369,
  "beacon_hub_id": 12,
  "raum_id": 10,
  "raum_name": "R_7_4_12a",
  "raum_aktiv": true,
  "raum_aktiv_seit": "2024-04-14T12:00:00.000Z",
  "raum_inaktiv_seit": "2024-04-25T22:00:00.000Z",
  "bereich_name": "Haus 7, Station 4",
  "bereich_id": 16
}
```


Im Falle einer Verbindung des Beacons mit einem Raum werden hier über den DB-View „beacon_raum_bereich“ auch die Informationen zu Raum und Bereich in der Antwort mitgegeben.

4.7.2 Pflegeinterface

Nuxt

Nuxt ist ein kostenloses Open-Source-Framework, das die Erstellung von produktionsreifen Webanwendungen mit Vue.js vereinfacht. Das Meta-Framework erweitert den Funktionsumfang von Vue.js unter anderem um dateibasiertes Routing und eine Server-Runtime. [47]

Nuxt setzt auf Konventionen und eine klare Verzeichnisstruktur, die die Automatisierung von wiederkehrenden Aufgaben ermöglicht und das Projekt strukturiert.

Vue

Vue.js ist ein clientseitiges JavaScript-Webframework zum Erstellen von Single-Page-Webanwendungen nach dem Model View ViewModel Muster. Es kann, wie in diesem Fall, auch in Multipage-Webseiten für einzelne Abschnitte verwendet werden.

Umsetzung

Abb. 18 enthält eine Übersicht der typischen Komponenten einer Webanwendung, die mit Nuxt.js und Vue.js umgesetzt wurde:

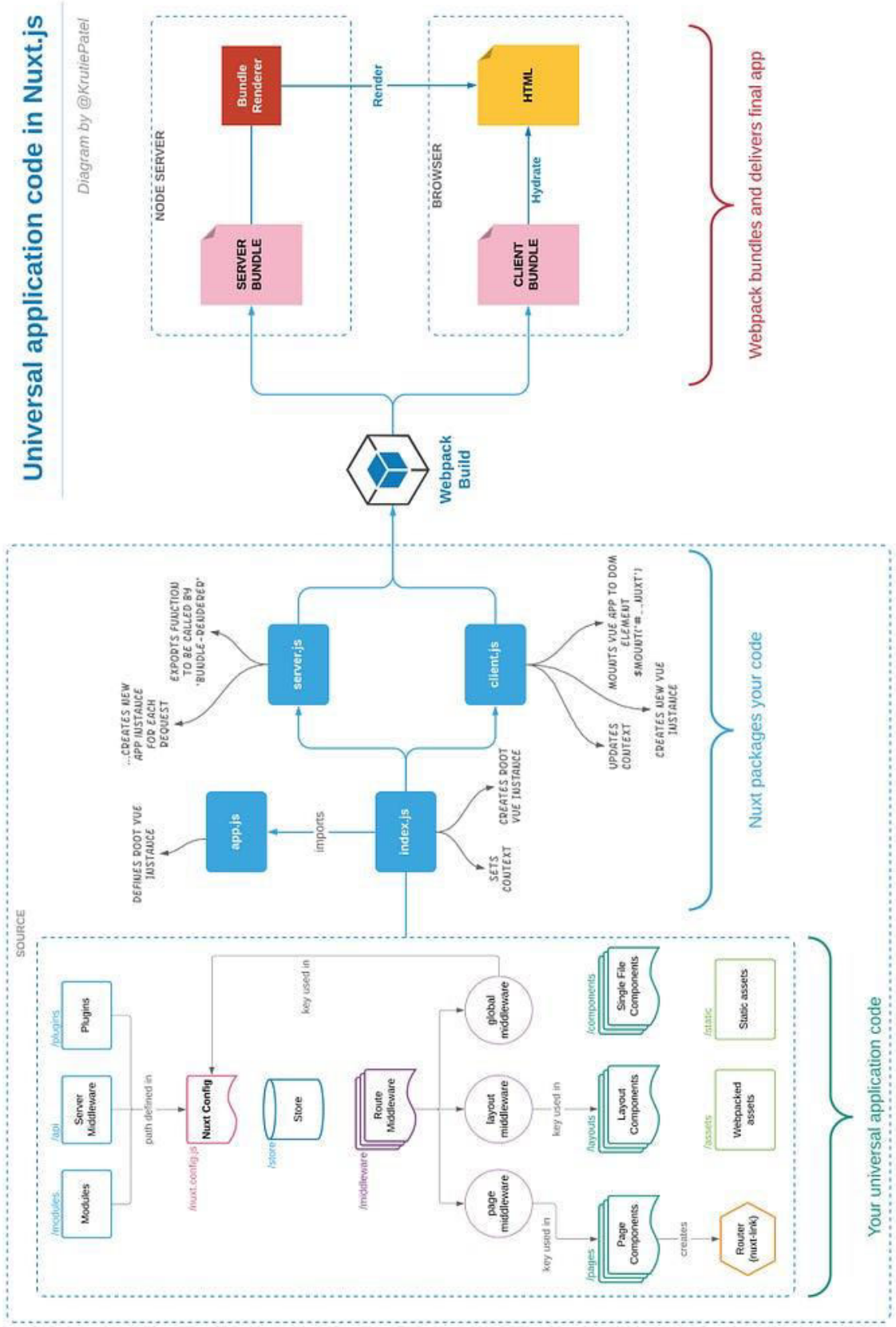


Abb. 18: - Nuxt - Vue Anwendungsarchitektur [53]

Die Ordnerstruktur der Anwendung wurde angelegt mit:

- „assets“
  - „main.css“ – Stylesheets für das Design der Seitenelemente
  - „main.js“ – Zentrale JS-Funktionen, die in mehreren Modulen Anwendung finden
- „pages“
  - „Seitenname.vue“ – alle Seiten jeweils als „.vue“ Dateien. Beispiel: „beacon_add.vue“
- „.env“ – Konfiguration der umgebungsspezifischen Einstellungen, wie der „API_URL“
- „app.vue“ – Startdatei der gesamten Anwendung. definiert hier u.a. die Navigation
- „env.d.ts“ – Konfiguration zum Build der App, importiert die „API_URL“
- „nuxt.config.js“ - Konfiguration der Nuxt spezifischen Regeln. Hier die Konfiguration der Reverse-Proxy Funktion für das Durchreichen von API-Anfragen an das Backend und die Nutzung der CSS aus dem „assets“ Ordner.

Seitenstruktur

Die Start-Datei der Anwendung, „app.vue“ (siehe Anhang A13), enthält:

```
1.  <template>
2.    <div>
3.      <NuxtLayout>
4.        <nav class="navbar">
5.          <ul class="nav-list">
6.            <li class="nav-item"><nuxt-link to="/" exact-active-class="active-
                        link">Home</nuxt-link></li>
7.            <li class="nav-item"><nuxt-link to="/bereich" exact-active-class="active-
                        link">Bereiche</nuxt-link></li>
8.            <li class="nav-item"><nuxt-link to="/raum" exact-active-class="active-
                        link">Räume</nuxt-link></li>
9.            <li class="nav-item"><nuxt-link to="/hub" exact-active-class="active-
                        link">Hubs</nuxt-link></li>
10.           <li class="nav-item"><nuxt-link to="/beacon" exact-active-class="active-
                        link">Beacons</nuxt-link></li>
11.           <li class="nav-item"><nuxt-link to="/mp" exact-active-class="active-
                        link">Medizinprodukte</nuxt-link></li>
12.           <li class="nav-item"><nuxt-link to="/mp_typ" exact-active-class="active-
                        link">Medizinprodukt Typen</nuxt-link></li>
13.           <li class="nav-item"><nuxt-link to="/mp_mapping" exact-active-class="active-
                        link">Medizinprodukt Mapping</nuxt-link></li>
14.         </ul>
15.       </nav>
16.         <NuxtPage />
17.       </NuxtLayout>
18.     </div>
19. </template>
```

Die resultierende Navigationsleiste ist in Abb. 19 dargestellt.

Abb. 19: Navigationsleiste der Webanwendung

Für jede aufrufbare Hauptseite wird in der Navigation ein Eintrag bereitgestellt.

Ein Link wie „to="/bereich"" ruft aus dem Verzeichnis „pages" die Inhalte der „bereich.vue" als Seite auf.

„<NuxtPage />" definiert die Position, an der der Seiteninhalt einer Seite dargestellt werden soll.

Diese Vorgabe erlaubt mit einfachen Mitteln, dass die Hauptnavigation auf jeder Seite sichtbar ist und immer über dem jeweiligen Seiteninhalt abgebildet wird.

Das Hinzufügen, Ändern oder Löschen eines Eintrags wird über Buttons auf den jeweiligen Seiten der Webanwendung ermöglicht.

Die Funktionalität einer vollständig umgesetzten Ansicht wird exemplarisch anhand der Seite „Hubs" („hub.vue", siehe Anhang A14) erläutert. Ergänzend wird auf das Hinzufügen eines neuen Hubs („hub_add.vue", siehe Anhang A15) eingegangen. Die Programmierung der anderen Seiten folgt einem ähnlichen Schema. Das vollständige Verzeichnis kann den o.g. GitHub Repositories entnommen werden.

Der Seitenaufbau der Hub-Übersicht beginnt mit dem Header der Tabelle zur Ansicht der Inhalte und einem Iterator über die Inhalte. Alle erforderlichen Daten werden über die API geladen, die Anzahl der Elemente wird automatisch über diese geladenen Daten bestimmt.

Abb. 20 zeigt die resultierende Webansicht.

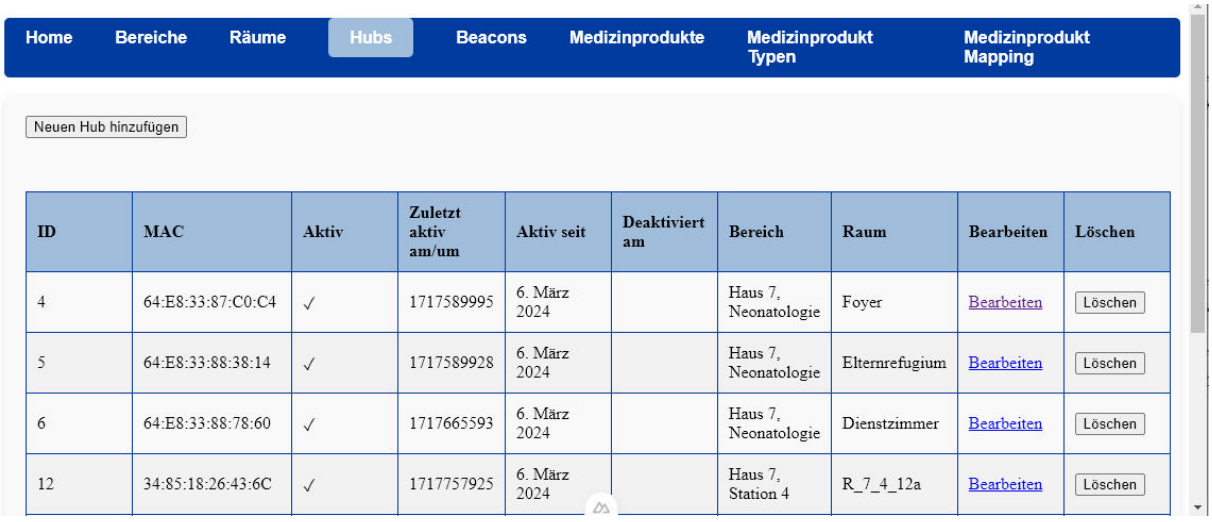


Abb. 20: Webansicht "Hubs"

Nach der globalen Navigationsleiste folgt der Button, mit dem dem ATS ein neuer Hub hinzugefügt werden kann. Bei Betätigen ruft dieser die Vue Komponente „/hub_add" auf.

```
1.  <template>
2.    <div class="container">
3.        <router-link to="/hub_add" custom v-slot="{ navigate }">
4.          <button @click="navigate" role="link">Neuen Hub hinzufügen</button>
5.        </router-link>
```

Unter dem Button wird die tabellarische Liste aller Hubs erzeugt. Über den Tabellenheader besteht die Möglichkeit, den Inhalt der Tabelle zu sortieren.

```
 7.         <table class="table">
 8.           <thead>
 9.             <tr>
10.               <th @click="sort('hub_id')">ID <span v-show="sortColumn === 'hub_id'">{{
                        sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
11.               <th @click="sort('hub_MAC')">MAC <span v-if="sortColumn === 'hub_MAC'">{{
                        sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
12.               <th @click="sort('hub_aktiv')">Aktiv <span v-if="sortColumn === 'hub_aktiv'">{{
                        sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
13.               <th @click="sort('hub_timestamp')">Zuletzt aktiv am/um <span v-if="sortColumn
                        === 'hub_timestamp'">{{ sortOrder === 'asc' ? '↑' : '↓'
                        }}</span></th>
14.               <th @click="sort('hub_aktiv_seit')">Aktiv seit <span v-if="sortColumn ===
                       'hub_aktiv_seit'">{{ sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
15.               <th @click="sort('hub_inaktiv_seit')">Deaktiviert am <span v-if="sortColumn ===
                       'hub_inaktiv_seit'">{{ sortOrder === 'asc' ? '↑' : '↓'
                       }}</span></th>
16.               <th @click="sort('bereich_name')">Bereich <span v-if="sortColumn ===
                       'bereich_name'">{{ sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
17.               <th @click="sort('raum_name')">Raum <span v-if="sortColumn === 'raum_name'">{{
                       sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
18.
19.
20.               <th>Bearbeiten</th>
21.               <th>Löschen</th>
22.             </tr>
23.           </thead>
```

Durch diese Funktion

```
25.             <tr v-for="hub in hubs" :key="hub.hub_id">
```

werden im Body der Tabelle alle vorhandenen Datensätze iterativ verarbeitet und in den korrekten Tabellenfeldern dargestellt.

```
24.           <tbody>
25.             <tr v-for="hub in hubs" :key="hub.hub_id">
26.               <td>{{ hub.hub_id }}</td>
27.               <td>{{ hub.hub_MAC }}</td>
28.               <td><template v-if="hub.hub_aktiv == true">✓</template></td>
29.               <td>{{ hub.hub_timestamp }}</td>
30.               <td>{{ formatDate(hub.hub_aktiv_seit) }}</td>
31.               <td>{{ formatDate(hub.hub_inaktiv_seit) }}</td>
32.               <td>{{ hub.bereich_name }}</td>
33.               <td>{{ hub.raum_name }}</td>
34.               <td><router-link :to="'/hub_edit?id=' + hub.hub_id">Bearbeiten</router-
                                  link></td>
35.               <td><button @click="delete_hub(hub.hub_id)">Löschen</button></td>
36.             </tr>
37.           </tbody>
38.         </table>
39.       </div>
40.     </template>
```

Die dafür nötigen Datensätze werden über JavaScript Funktionen geladen und bereitgestellt. Funktionen zum Sortieren oder Formatieren der Datumsausgabe werden ebenfalls mit JavaScript umgesetzt.

Zur Veranschaulichung wird die Funktion „fetch_hubs“ dargestellt. Sie ruft die in „hub.js“ definierte API-Anfrage „/api/hub“ auf und wartet auf Rückgabe der Werte aus der DB-Anfrage. Sobald die Werte als Antwort angekommen sind, speichert die Funktion diese in dem Datensatz „hubs“, aus dem die zuvor beschriebenen Tabelleneinträge generiert werden können.

```
91.       fetch_hubs() {
92.             fetch('http://localhost:3000/api/hub')
93.             .then(response => {
94.                 console.log('response: ',response)
95.                 if (!response.ok) {
96.                     throw new Error('Network response was not ok');
97.                 }
98.                 return response.json();
99.             })
100.            .then(data => {
101.                this.hubs = data;
102.            })
103.            .catch(error => {
104.                console.error('Error fetching hubs:', error);
105.            });
106.        },
```

Der gleichen Struktur folgt das Hinzufügen eines Hubs über das entsprechende Webformular (siehe Abb. 21)

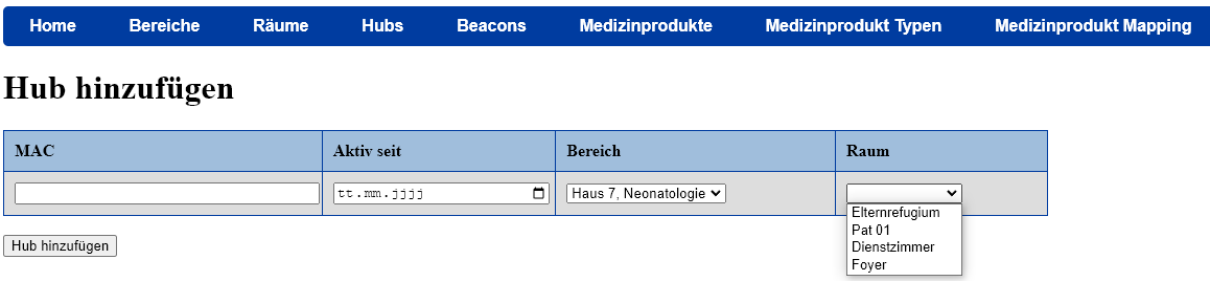


Abb. 21: Webansicht zum Hinzufügen eines Hubs mit Zuweisung von Raum und Bereich

Die Eingabefelder sind die MAC-Adresse des Hubs, das Datum einer eventuellen Aktivsetzung sowie Bereich und Raum, die dem Hub zugeordnet werden können. Da ein Raum in der DB immer an einen Bereich der jeweiligen Einrichtung gekoppelt ist, müssen zunächst alle Bereiche in dem ersten Dropdown-Menü zur Auswahl stehen. Wird in diesem Bereichs-Dropdown einer ausgewählt, wird nachfolgend anhand der „bereich_id“ angefragt, welche Räume sich in dem Bereich befinden. Dies wird über einen Änderungs-Trigger („@change“, Z. 21) realisiert, der die Methode „fetch_raeume (neuer_Hub.bereich_id)“ (Z. 80) aufruft. Diese stellt im Feld „Raum“ die Auswahl-der Räume des gewählten Bereiches bereit.

```
14.           <tbody>
15.             <tr>
16.               <td><input type="text" id="hub_MAC" v-model="neuer_Hub.hub_MAC"
                              required></td>
17.
18.               <td><input type="date" id="hub_aktiv_seit" v-model="formattedActiveSince">
                  </td>
20.               <td>
```

```
21.                 <select id="hub_bereich_id" v-model="neuer_Hub.bereich_id"
                                    @change="fetch_raeume(neuer_Hub.bereich_id)">
22.                   <option v-for="bereich in bereiche" :key="bereich.bereich_id"
                            :value="bereich.bereich_id">{{ bereich.bereich_name }} </option>
23.                 </select>
24.               </td>
25.               <td>
26.                 <select id="hub_raum_id" v-model="neuer_Hub.raum_id" >
27.                   <option v-for="raum in raeume" :key="raum.raum_id"
                          :value="raum.raum_id">{{ raum.raum_name }}</option>
28.                 </select>
29.               </td>
30.             </tr>
31.           </tbody>
[…]
39.   <script setup>
[…]
61. // Holen aller Bereiche aus der Datenbank
62. // 'zuruecksetzen' wird benutzt, um zwischem erstem Laden und Änderung im Formular zu unter-
      scheiden
63. async function fetch_bereiche() {
64.   try {
65.     const response = await fetch('http://localhost:3000/api/bereich')
66.     if (!response.ok) {
67.       throw new Error('Failed to fetch areas')
68.     }
69.     bereiche.value = await response.json()
70.     console.log('Bereiche geholt. Bereiche: ', bereiche.value, "Aktuelle BereichsID ist ",
                 neuer_Hub.value.bereich_id)
71.
72.
73.   } catch (error) {
74.     console.error('Error fetching areas:', error)
75.   }
76. }
77.
78. // Holen aller Räume eines Bereiches aus der Datenbank
79. async function fetch_raeume(bereich) {
80.     try {
81.         // Holen der Räume des selektierten Bereiches! Also nur genau dieser Räume
82.         // `` zur Interpretation der Variablen ...
83.         const response = await fetch(`http://localhost:3000/api/raum_bereich/${bereich}`)
84.         if (!response.ok) {
85.         throw new Error('Fehler beim Lesen der Räume')
86.         }
87.         raeume.value = await response.json()
88.     } catch (error) {
89.         console.error('Error Fehler beim Holen der Räume:', error)
90.     }
91.     console.log('Räume geholt: ', raeume.value)
92.     // Wenn ein neuer Bereich gewählt wurde, muss das Feld im Formular geleert werden
93.     neuer_Hub.raum_id = null // Bereichswechsel
94. }
```

Mit der Entwicklung dieser und den o. g. Ansichten ist die Umsetzung der Webanwendung vollständig und die Entwicklung aller Einzelkomponenten des ATS ist abgeschlossen.

5. Systemtests und Verifikation

In der Abschlussphase des Projekts gilt es sicherzustellen, dass das System alle zuvor definierten Anforderungen erfüllt und als funktionsfähiges Prototyp-System bereits begrenzt einsatzfähig wäre. Dazu sind, zusätzlich zu den Tests während der Entwicklungsphase, weitere umfassende Tests zur Überprüfung der genügenden Funktionalität, Stabilität und technischen Reife des Systems durchzuführen.

Die Tests teilen sich in folgende Bereiche auf:

1. Funktionale Tests: Überprüfung der korrekten Funktion aller Komponenten (Hubs, Beacons, MQTT-Broker, Microservice, DB und Webanwendung)
2. Integrationstests: Sicherstellung der reibungslosen Interaktion aller Systemkomponenten
3. Belastungstests: Simulation der gleichzeitigen Nutzung vieler Beacons und Hubs, um die Stabilität und Performance unter hoher Last zu testen

Es kann davon ausgegangen werden, dass das Sicherheitsrisiko in Hinblick auf Datenschutz minimal ist, da das ATS weder personenbezogenen Daten erfasst noch einen Personenbezug zu Daten herstellt. Deshalb und weil diese Arbeit ihren Fokus auf eine funktionelle, prototypische Entwicklung legt, wird auf eine diesbezügliche Durchführung von Tests verzichtet.

Die nachfolgenden Testbeschreibungen umfassen alle geprüften Aspekte. Um die Detailtiefe der Abläufe wiederzugeben und gleichzeitig Wiederholungen zu vermeiden, werden wesentliche Aspekte exemplarisch vertieft und mit Abbildungen versehen.

5.1. Funktionale Tests

Zu Beginn werden die grundlegenden Funktionen aller einzelnen Komponenten des ATS getestet. Dazu werden die Systemkomponenten auf dem Windows-PC ausgeführt, auf dem auch die Entwicklung stattgefunden hat. Dieses Vorgehen bietet den Vorteil eines vereinfachten Debuggings und ermöglicht bei Bedarf ein unmittelbares Anpassen des Codes.

Beacons und Hubs

Für die korrekte Funktion des ATS müssen die Beacons in regelmäßigen Abständen Signale senden und die Hubs diese korrekt empfangen und weiterleiten. Außerdem dürfen nur solche Daten von Beacons verarbeitet werden, die zum ATS gehören. Der übertragene UUID der Beacons muss dem in den Hubs hinterlegten UUID entsprechen.

Die zu prüfenden Aspekte werden getestet, indem ein Hub so konfiguriert wird, dass er alle erforderlichen Daten über eine serielle Schnittstelle sichtbar macht. Diese können anschließend an dem mit dem Hub verbundenen PC in einem Terminal ausgelesen werden.

Die zwei Entwicklungsboards, aus denen der Hub besteht, werden einzeln ausgelesen. Das BLE-Board ist so eingestellt, dass es die empfangenen und zunächst kodierten Werte der Beacons im Terminal ausgibt. So kann nachvollzogen werden, ob und wann ein zulässiges Beaconsignal empfangen wurde.

Das Mesh-Board soll nun die über die serielle Schnittstelle empfangenen kodierten Daten ebenfalls im Terminal ausgeben, um diese mit den Daten des BLE-Boards zu vergleichen. Anschließend werden sie dekodiert und erneut ausgegeben. Auf diese Weise kann festgestellt werden, ob alle empfangenen Daten über die serielle Schnittstelle weitergeleitet werden und ob die Dekodierung fehlerfrei funktioniert.

Ein Beacon wird auf ein Sendeintervall von einer Sekunde eingestellt und aktiviert. Damit kann getestet werden, ob der Beacon zuverlässig im eingestellten Intervall sendet und ob die Daten zuverlässig vom BLE-Board empfangen werden.

Der Output der Terminals ist exemplarisch in den Abb. 22 und 23 zu sehen. Die rote Markierung im BLE-Terminal kennzeichnet die im Mesh-Terminal während der Verarbeitung ausgegebenen Daten.

```
I (5980) Write data: : dd ef 24 5a 28 d3 62 00 ca
I (6180) Write data: : db b4 ab 2c 37 42 63 00 bb
I (6460) Write data: : dd ef 24 5a 28 d3 62 00 d5
I (7420) Write data: : dd ef 24 5a 28 d3 62 00 e0
I (7900) Write data: : dd ef 24 5a 28 d3 62 00 e0
I (8100) Write data: : db b4 ab 2c 37 42 64 00 d8
I (9050) Write data: : db b4 ab 2c 37 42 64 00 db
I (9340) Write data: : dd ef 24 5a 28 d3 62 00 cf
```

Abb. 22: Terminal Ausgabe BLE-Board

```
I (21134) Read data:: db b4 ab 2c 37 42 64 00 d8
I (21134) ROOM_MONITOR: Tried to publish layer=1, MAC_ROOM=64:E8:33:88:78:60, MAC_SENSOR=DB:B4:AB:2C:37:42, BATT=100, BUTTON=0, RSSI=216
I (21144) mesh_mqtt: sent publish on topic sensors/rooms with value layer=1, MAC_ROOM=64:E8:33:88:78:60, MAC_SENSOR=DB:B4:AB:2C:37:42, BATT=100
, BUTTON=0, RSSI=216 returned msg_id=29671
I (21154) ROOM_MONITOR: uart rx length 90
I (21184) mesh_mqtt: MQTT_EVENT_PUBLISHED, msg_id=29671
I (22084) Read data:: db b4 ab 2c 37 42 64 00 db
I (22084) ROOM_MONITOR: Tried to publish layer=1, MAC_ROOM=64:E8:33:88:78:60, MAC_SENSOR=DB:B4:AB:2C:37:42, BATT=100, BUTTON=0, RSSI=219
I (22094) mesh_mqtt: sent publish on topic sensors/rooms with value layer=1, MAC_ROOM=64:E8:33:88:78:60, MAC_SENSOR=DB:B4:AB:2C:37:42, BATT=100
, BUTTON=0, RSSI=219 returned msg_id=63572
I (22114) ROOM_MONITOR: uart rx length 90
I (22144) mesh_mqtt: MQTT_EVENT_PUBLISHED, msg_id=63572
I (22374) Read data:: dd ef 24 5a 28 d3 62 00 cf
I (22374) ROOM_MONITOR: Tried to publish layer=1, MAC_ROOM=64:E8:33:88:78:60, MAC_SENSOR=DD:EF:24:5A:28:D3, BATT=98, BUTTON=0, RSSI=207
I (22384) mesh_mqtt: sent publish on topic sensors/rooms with value layer=1, MAC_ROOM=64:E8:33:88:78:60, MAC_SENSOR=DD:EF:24:5A:28:D3, BATT=98,
 BUTTON=0, RSSI=207 returned msg_id=5949
I (22394) ROOM_MONITOR: uart rx length 90
I (22424) mesh_mqtt: MQTT_EVENT_PUBLISHED, msg_id=5949
```

Abb. 23: Terminal Ausgabe Mesh-Board

Der Test wurde mit drei verschiedenen Beacons über jeweils eine Minute durchgeführt. Das Sendeintervall war über die gesamte Zeit hinweg konstant. Weder blieb eine Beaconmeldung aus, noch wurde eine nicht empfangen. Die Weiterleitung über die serielle Schnittstelle der ESP32C3-Boards funktionierte lückenlos, genauso wie das Dekodieren der empfangenen Daten.

Die korrekte Funktion der Mesh-Bildung wurde getestet, indem vier Hubs in vier Räumen über zwei Geschosse verteilt und eingeschaltet wurden. Es wurden zwei Access Points als mögliche Verbindungsstellen zum lokalen Netzwerk eingerichtet, um zu testen, ob die Hubs des Meshs sich lediglich mit

einem von beiden verbinden. Einer der Hubs war währenddessen mit dem PC verbunden, um relevante Daten zur Mesh-Bildung im Terminal auszugeben.

Im Ergebnis haben sich die Hubs korrekt miteinander vernetzt. Nach weniger als einer Minute hatten sich alle Hubs zu einem Mesh mit einer Root- und drei Child-Nodes verbunden. Die Daten des das lokale Netzwerk aufbauenden Routers, zeigen, dass exakt ein Hub mit dem Netzwerk verbunden war. Dieser befand sich in demselben Raum, wie der Router. Daraus kann wiederum abgeleitet werden, dass die Verbindung zum lokalen Netzwerk mithilfe des empfangsstärksten Hubs erfolgt ist.

MQTT-Broker

Die korrekte Funktion des MQTT-Brokers soll sichergestellt werden, indem unter einem Topic Nachrichten gesendet und empfangen werden (publish und subscribe). Die empfangene Nachricht soll dem Schema „{"time":"*Unix-Zeitstempel*","data":"*Nachrichteninhalt*"}“ folgen. Der Test wurde mithilfe des MQTT-5.0-Clients „MQTTX“ durchgeführt [48].

Ein Versenden unterschiedlichster Nachrichten hat ergeben, dass der Broker einwandfrei funktioniert. Allen Nachrichten wurde ein UNIX-Zeitstempel mit der aktuellen Uhrzeit beigefügt. Das Format der Nachrichten stimmte mit dem beschriebenen Schema überein (siehe Abb. 24 und 25).

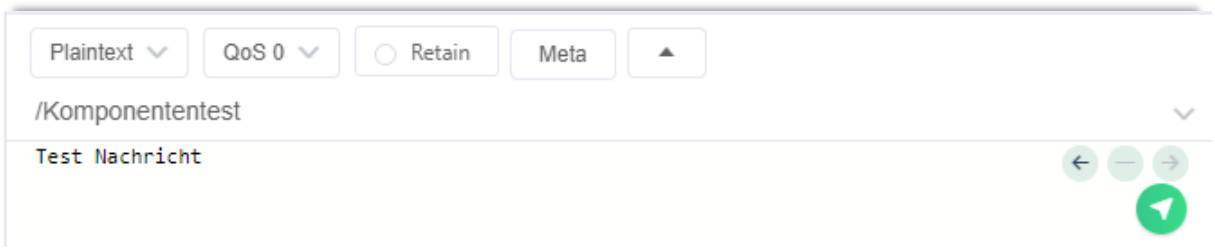


Abb. 24: MQTT-Broker Zeitstempel Testnachricht

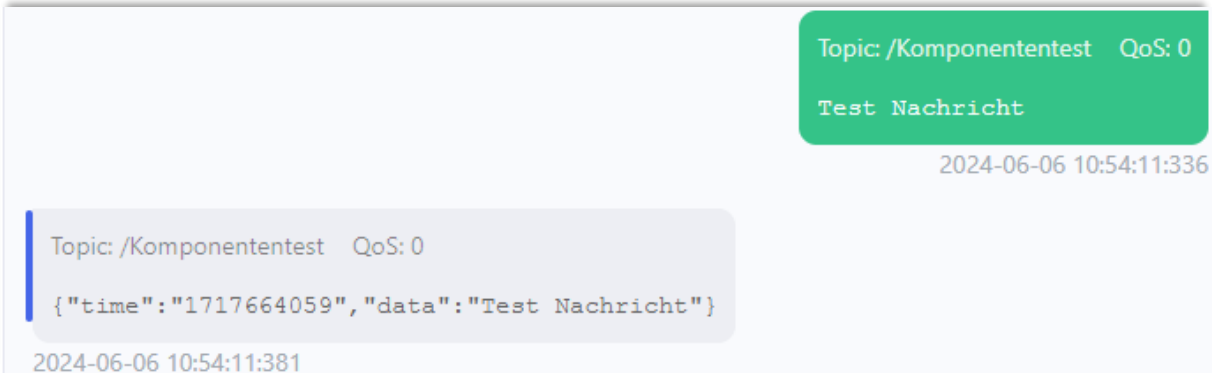


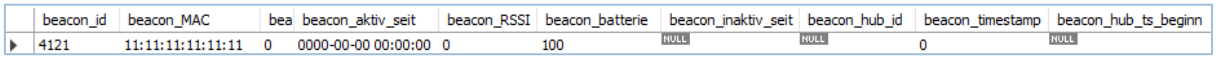
Abb. 25: MQTT-Broker Zeitstempel Testantwort

Datenbank

Das Testen der DB ist ein Kernbestandteil der Verifikation des ATS. Ein Fehler an dieser Stelle würde das ganze System beeinträchtigen. Daher wird jede Spalte in jeder Tabelle mithilfe von „INSERT“- und „UPDATE“-Befehlen auf korrekte Funktion überprüft. Besonders zu beachten sind die erzeugten Abhängigkeiten zwischen den Tabellen, da bei einem beliebigen Befüllen Fremdschlüsselkonflikte auftreten könnten. Auch die Eigenschaften einzelner Tabellenspalten (NotNull, Auto-Increment, Default Values etc.) sollten auf ihre Richtigkeit geprüft werden.

In Abb. 26 und 27 wird gezeigt, wie ein unvollständiger „INSERT“-Befehl in die „beacon“ Tabelle trotz fehlender Werte durchgeführt wird, indem die zur weiteren Verarbeitung erforderlichen Einträge mit Default Values gefüllt werden.

Abb. 26: Unvollständiger "INSERT" Befehl für "beacon" Tabelle



beacon_id	beacon_MAC	bea	beacon_aktiv_seit	beacon_RSSI	beacon_batterie	beacon_inaktiv_seit	beacon_hub_id	beacon_timestamp	beacon_hub_ts_beginn
4121	11:11:11:11:11:11	0	0000-00-00 00:00:00	0	100	NULL	NULL	0	NULL

Abb. 27: Eintrag in "beacon" Tabelle nach unvollständigem "INSERT"

Zusätzlich zum Test aller Tabellen müssen ebenfalls die für die Historienerstellung zuständigen „ON UPDATE“-Trigger der „beacon“- und „hub“- Tabellen geprüft werden. Dazu werden die Beacon- und Hubdaten aktualisiert und die Einträge der jeweiligen Historientabellen überprüft.

Die Ausführung aller „INSERT“ und „UPDATE“ Befehle mithilfe der MySQL-Workbench erfolgte fehlerfrei. Alle gesetzten Primär- und Fremdschlüssel sowie Eigenschaften der Tabellenspalten sind korrekt. Wird einer Tabelle ein Eintrag hinzugefügt, der nicht alle erforderlichen Datenpunkte enthält, werden, wie konzipiert, die Default Values genutzt. Auch die Erstellung von Historieneinträgen bei einer Änderung von Beacon- und Hubdaten erfolgte einwandfrei.

Microservice

Da die korrekte Funktion des MQTT-Brokers und der DB sichergestellt sind, können diese genutzt werden, um den Microservice zu testen. Dafür werden mithilfe von MQTTX gezielt fabrizierte Beacondaten an das Topic „/sensors/rooms“ gesendet, die bestimmte Funktionen provozieren. Durch das Verstellen des Debugging-Levels werden alle nötigen Daten im Terminal von VS-Code ausgegeben. Um den initialen Start des Programms zu simulieren, kann mithilfe der Funktion „memcache.flush()“ bei Bedarf der Memcache geleert werden.

Das Senden unterschiedlicher Beacondaten aus dem MQTTX-Client hat ergeben, dass alle Funktionen des Microservices einwandfrei funktionieren. Alle 3 erläuterten Verarbeitungsfälle (siehe Kap. 4.6. „Entwicklung Microservice und Datenbank“) werden korrekt ausgeführt. Auch die Zuordnung von Beacons zu Beaconpaaren funktioniert. Es wurden nur diejenigen Beacons gepaart, deren zugeordnete MP dafür in dem Mapping der DB eingetragen sind.

Wie beabsichtigt, löst ein Raumwechsel eines gepaarten Beacons den Eintrag eines „kritischen Mappings“ aus, wie er in Kap. 4.6. in den Zeilen 431 ff. des Microservices umgesetzt wurde. Ein unmittelbar nachfolgender Raumwechsel des gepaarten Beacons führt wiederum zur Auflösung dieser kritischen Einträge. Die Abb. 28 und 29 zeigen beispielhaft einige Ausgaben im Terminal von VSCode.

```
beacon-Altdaten im cache:  [17, 5, 200, 1717499751, 1717499751, 99, 14, 1717499751]
beacon Altdaten:  [17, 5, 200, 1717499751, 1717499751, 99, 14, 1717499751]
beacon im Vergleich speichern
beacon im Cache/in DB aktualisiert.
beacon aktualisiert
beacon Neudaten sind:  [17, 5, 200, 1717499885, 1717499751, 99, 14, 1717499751]
Taster wurde gedrückt, beaconpairing eingeleitet
beaconpairing_cache_5_14
(17, 1717499885)
Extraktion und Schicken der Nachricht  00001  hat 0.00108  Sekunden gedauert
Aktualisierung Hub: Bereits im Cache:  [5, 1717499885, 1717499751]
Aktualisierung Hub: Bereits im Cache, aber neuer TS:  [5, 1717499892, 1717499751]
beacon-Altdaten im cache:  [18, 5, 200, 1717499766, 1717499766, 99, 15, 1717499766]
beacon Altdaten:  [18, 5, 200, 1717499766, 1717499766, 99, 15, 1717499766]
beacon im Vergleich speichern
beacon im Cache/in DB aktualisiert.
beacon aktualisiert
beacon Neudaten sind:  [18, 5, 200, 1717499892, 1717499766, 99, 15, 1717499766]
Taster wurde gedrückt, beaconpairing eingeleitet
Beaconpair Data von 17 in Memcache: [20, 18]
Eintrag in die DB-Tabelle 'beaconpair' erfolgt oder aktualisiert
Extraktion und Schicken der Nachricht  00002  hat 0.17354  Sekunden gedauert
```

Abb. 28: Terminal Ausgabe Microservice, Beaconpaarung

```
Unerwarteter Verbindungsverlust zum MQTT-Broker.
Mit MQTT-Broker verbunden.
Hub noch nicht im Cache. Cache geladen mit:  None
beacon-Altdaten im cache:  [18, 5, 200, 1717499892, 1717499766, 99, 15, 1717499766]
beacon Altdaten:  [18, 5, 200, 1717499892, 1717499766, 99, 15, 1717499766]
Neuer Gefährdungseintrag beacon_mapping_krit_17_18 erstellt.
Extraktion und Schicken der Nachricht  00003  hat 0.31617  Sekunden gedauert
Aktualisierung Hub: Bereits im Cache:  [6, 1717499976, 1717499976]
Aktualisierung Hub: Bereits im Cache, aber neuer TS:  [6, 1717499984, 1717499976]
beacon-Altdaten im cache:  [17, 5, 200, 1717499885, 1717499751, 99, 14, 1717499751]
beacon Altdaten:  [17, 5, 200, 1717499885, 1717499751, 99, 14, 1717499751]
Neuer Gefährdungseintrag beacon_mapping_krit_20_17 erstellt.
Gefährdungseintrag beacon_mapping_krit_18_17 und beacon_mapping_krit_17_18 gelöscht.
Extraktion und Schicken der Nachricht  00004  hat 0.17878  Sekunden gedauert
```

Abb. 29: Terminal Ausgabe Microservice, kritisches Mapping

Die ersten zwei Zeilen der Abb. 29 belegen, dass die Ausnahmebehandlung bei einem Verbindungsverlust zum MQTT-Broker funktioniert.

Die in einem weiteren Python-Programm umgesetzte Überprüfung der „kritischen Mappings“ löste Beaconpaare im Memcache und der DB erfolgreich auf, deren Mapping älter als 60 Sekunden war (siehe Abb. 30).

```
['17,18', '20,19']
Beacon pair 20 and 19 dissolved in DB.
Beacon pair 20 and 19 dissolved in Memcache.
Critical mapping beacon_mapping_krit_20_19 dissolved.
Critical mapping beacon_mapping_krit_19_20 dissolved.
['17,18']
```

Abb. 30: : Terminal Ausgabe Microservice, Beaconpaar Überprüfung

Webanwendung und API

Zur Überprüfung der Funktionalität der Benutzeroberfläche werden alle Ansichten aufgerufen und deren Funktionen getestet. Außerdem werden die angezeigten Daten mit den in der DB gespeicherten

Daten verglichen. Bei Nichtübereinstimmung könnten beispielsweise eine DB-Anfrage oder die darauffolgende Datenverarbeitung fehlerhaft sein.

Das Testen aller Ansichten und deren Funktion sowie ein Abgleich mit den Daten aus der DB hat ergeben, dass sowohl Frontend als auch Backend der Webanwendung einwandfrei funktionieren. Die angezeigten Daten stimmten zu jedem Zeitpunkt mit denen aus der DB überein. Alle über die Websites vorgenommene Datenänderungen wurden in der DB übernommen und direkt auf den Seiten aktualisiert. Durch ein Header-Element lässt sich zwischen den unterschiedlichen Ansichten wechseln, die aktuell gewählte ist dabei farblich hinterlegt. Die Tabellen bieten eine übersichtliche Darstellung der jeweiligen Daten.

Ein auf der MP-Website durchgeführtes Testszenario wird beispielhaft in den Abb. 31 bis 33 dargestellt.

Home | Bereiche | Räume | Hubs | Beacons | Medizinprodukte | Medizinprodukt Typen | Medizinprodukt Mapping

Neues MP hinzufügen

Phys

ID	Name	Seriennummer	Typ	Aktiv	Aktiv seit	Deaktiviert am	Bereich	Raum	Beacon ID	Bearbeiten	Löschen
7	Physio Control / Stryker LifePak 15	123456789	Defibrilator	✓	June 1, 2024		Haus 7, Neonatologie	Dienstzimmer	1	Bearbeiten	Löschen
9	4- /6-adriges EKG-Komplettkabel mit Klammern, Physio Control #11111-000019	i1234	Patientenkabel (Komplettkabel)	✓	May 26, 2024		Haus 7, Station 4	R_7_4_12a	13	Bearbeiten	Löschen
11	6-adr. EKG-Brustwandableitung mit Klammern, zu Physio Control für für LP 12/15/20/20e	i1235	Patientenkabel (Komplettkabel)	✓	January 29, 2024		Haus 7, Station 4	R_7_4_12a	14	Bearbeiten	Löschen

Abb. 31: Webansicht "Medizinprodukte" mit Suchfeldtest

In das Suchfeld wurde „Phys“ eingegeben. Es werden korrekterweise nur Einträge gezeigt, die diesen String enthalten. Bei einem Klick auf „Bearbeiten“ wird man zu dem jeweiligen MP weitergeleitet

MP bearbeiten

Name	Seriennummer	Aktiv seit	Typ	Beacon ID
Physio Control / Stryker LifePak 15	123456789	June 3, 2024	Defibrilator	1

Änderungen speichern

Abb. 32: Test des "Bearbeiten" Buttons in der "MP" Webansicht

Nach dem Anpassen der zugeordneten Beacon ID und dem anschließenden Speichern wird der Eintrag in der DB aktualisiert und der Browser springt zur MP-Übersicht zurück. Dort ist das MP mit der neu zugeordneten Beacon ID „3333“ zu sehen. Dem MP sind zudem die aus dem Beacon resultierenden Raum- und Bereichsinformationen zugeordnet worden.

ID	Name	Seriennummer	Typ	Aktiv	Aktiv seit	Deaktiviert am	Bereich	Raum	Beacon ID	Bearbeiten	Löschen
7	Physio Control / Stryker LifePak 15	123456789	Defibrilator	✓	June 2, 2024		Test	Test2	3333	Bearbeiten	Löschen

Abb. 33: Aktualisierter MP Eintrag nach ändern der Beacon ID

Wird eine Beacon ID eingegeben, die nicht existiert oder die bereits einem MP zugeordnet wurde, erzeugt dies eine Fehlermeldung.

Bei der Durchführung der obigen Testschritte wurden keinerlei Fehler festgestellt, damit sind die durchzuführenden Einzeltests der Komponenten abgeschlossen.

5.2. Integrationstests

Nach der erfolgreichen Überprüfung der Funktion aller Einzelkomponenten muss sichergestellt werden, dass diese auch als Gesamtsystem innerhalb der geplanten Laufzeitumgebung funktionieren. Dies wird in Form eines Blackbox-Tests durchgeführt. Es wird geprüft, ob das System bei bestimmten Beacon- und Nutzerinputs (physischer Raumwechsel, Knopfdruck am Beacon etc.) die korrekten Daten erzeugt. Die Erzeugung kann anhand der Webansichten sowie über direkte DB-Anfragen nachvollzogen werden. Um einen möglichst realistischen Einsatzort nachzustellen, wurden die vier Hubs erneut in vier Räumen über zwei Geschosse verteilt platziert.

Solange die Beacons nicht bewegt wurden, waren sie jederzeit dem korrekten Hub und damit dem richtigen Raum zugewiesen. Bei einem Wechsel in einen anderen Raum dauerte die Neuzuordnung in der DB maximal 2 Sekunden.

Bei einem vergrößerten Sendeintervall von 10 Sekunden betrug die größte zeitliche Differenz nach Bewegung bis zum erfassten Ortswechsel, wie erwartet, in etwa 11 Sekunden. Wird ein Beacon kurz nach dem Senden in einen anderen Raum bewegt, sendet er dort erst nach Ablauf des eingestellten Intervalls, hier also nach ca. 10 Sekunden, erneut, wodurch sich die Verzögerung erklärt.

Wurden die Knöpfe zweier Beacons im selben Raum nur sehr kurz betätigt, konnte eine Beaconpaarung nicht immer zuverlässig erzeugt werden. Das Drücken der Knöpfe über eine Sekunde hinweg führte indes wie vorgesehen zuverlässig zu einer Verknüpfung der Beacons. Dies gilt sowohl für das Sendeintervall von 1 als auch von 10 Sekunden.

Wurde ein gepaarter Beacon in einen anderen Raum bewegt, war die Paarung nach etwa 1-2 Minuten nicht mehr in der DB eingetragen. Wurde jedoch der gepaarte Beacon innerhalb einer Minute in denselben Raum bewegt, blieb die Paarung, wie gewünscht, bestehen. Die zeitliche Schwankung beim Löschen der DB-Einträge war zu erwarten, da das Python-Programm so konfiguriert ist, dass es die kritischen Einträge lediglich alle 60 Sekunden prüft. Dies könnte bei Bedarf leicht angepasst werden.

5.3. Belastungstest

Zur Überprüfung der Robustheit des Systems wird ein Belastungstest durchgeführt. Über simulierte Beacon- und Hubmeldungen wird das System stark beansprucht und auf diese Weise seine Leistungsfähigkeit und Stabilität unter hoher Last getestet. Dafür wurden 1000 fiktive Beacons in der DB angelegt, zu denen ein Python-Programm die Meldungen erzeugt. Diese Meldungen enthalten gelegentliche Raumwechsel, Knopfdrücke sowie eine konstant schwankende Empfangsstärke.

Die Funktion, die die Nachrichten generiert und an den MQTT-Broker sendet, ist wie folgt definiert (siehe Anhang A16):

```
39. def publish_messages(client):
40.     i = 1
41.     while True:
42.         for _ in range(messages_per_second):
43.             mac_index = (i - 1) % len(mac_addresses)
44.             mac_address = mac_addresses[mac_index]
45.             randRSSI = random.randint(100,250)
46.             if i % 4 == 0:
47.                 lastNum = random.randint(1, 9)
48.             else:
49.                 lastNum = 2
50.             if i % 44 == 0 or i % 45 == 0:
51.                 randButton = random.randint(0, 1)
52.             else:
53.                 randButton = 0
54.             message_content = f"layer=1, MAC_ROOM=00:00:00:00:00:0{lastNum}, MAC_SEN-
                                   SOR={mac_address}, BATT=100, BUTTON={randButton},
                                   RSSI={randRSSI}"
55.
56.             client.publish(mqtt_topic, message_content)
57.             print(message_content)
58.             i += 1
59.         time.sleep(1)
```

Mithilfe von „lastNum" wird jeder 4ten Nachricht eine Hub-MAC-Adresse zugewiesen, die auf einer zufälligen Zahl von 2 bis 9 endet. Über „randButton“ wird einigen Nachrichten die Information eines Knopfdrucks beigefügt. Die RSSI aller Nachrichten variiert zufällig von 100 – 250. Die so generierten Beacon-Nachrichten sollen eine mögliche Belastung innerhalb einer Einrichtung simulieren. Die MAC-Adressen der Beacons erhält das Programm aus einer „.txt“ Datei im Programmverzeichnis.

Während das beschriebene Python-Programm läuft, wird der Testablauf aus Kap. 5.2. wiederholt. Damit kann festgestellt werden, ob die Daten unter diesen erschwerten Bedingungen genauso schnell und zuverlässig verarbeitet werden.

Nach dem Durchführen verschiedenster Beacon- und Nutzerinputs waren keine Änderungen im Vergleich zu den Ergebnissen aus dem vorherigen Test festzustellen. Die Daten in der DB entsprachen zu jedem Zeitpunkt den zu erwartenden Werten. Es trat keine erkennbare Verzögerung bei der Verarbeitung der Beaconmeldungen auf.

6. Diskussion

Zusammenfassung der Entwicklungs- und Testergebnisse

Die in den Kap. 2. „Anforderungsanalyse und Projektrahmen“ und 3. „Systemkonzept und Design“ identifizierten Systemkomponenten wurden den Anforderungen entsprechend entwickelt und ausgearbeitet. Im Rahmen der Entwicklung wurden bereits kleinere Fehler behoben und fehlende Funktionen ergänzt. Es folgten ausgiebige Tests, um sicherzustellen, dass alle Komponenten für sich genommen, wie gemeinsam funktionieren.

Die Datenerhebungen und Verarbeitungsschritte verliefen fehlerfrei. Die Kommunikation zwischen den Einzelkomponenten funktionierte ebenfalls wie beabsichtigt. Unter simulierter hoher Last wurden keine relevanten Unterschiede in der Verarbeitungsgeschwindigkeit von Beacondaten festgestellt.

Alle geplanten Webansichten wurden umgesetzt und enthalten die im Pflichtenheft geforderten Funktionen. Der Abgleich aller angezeigten Daten mit denen aus der DB ergab, dass diese zu jedem Zeitpunkt übereinstimmten. Diejenigen Elemente, die eine DB-Anfrage mit Datenänderung auslösen, funktionierten einwandfrei.

Interpretation und Bewertung der Entwicklungs- und Testergebnisse

Die Ergebnisse der Entwicklung einschließlich der durchgeführten Tests zeigen, dass das entwickelte System die im Lasten- und Pflichtenheft gestellten funktionalen Anforderungen erfüllt. Die geforderten Funktionen bezüglich Ortung und Zuordnung von MP konnten ausnahmslos umgesetzt werden. Die Möglichkeit der Interaktion mit den gewonnenen Daten ist gegeben. Das entwickelte Gesamtsystem spiegelt den gewählten Titel der Arbeit wider und zeigt auf, welche Möglichkeiten mit kostengünstigen Standardkomponenten und bereits vorhandener Infrastruktur bestehen.

Die Tests bestätigen, dass das ATS auch unter simulierter hoher Last erwartungsgemäß funktioniert. Es besteht die begründete Annahme, dass ein solches System bereits in dem hier entwickelten prototypischen Zustand einen erheblichen Beitrag zur Lösung der anfangs beschriebenen Problemen in Gesundheitseinrichtungen leisten könnte. Bezüglich der Skalierbarkeit wäre es auch für den Einsatz in größeren Einrichtungen geeignet.

Limitationen der Arbeit

Trotz der positiven Ergebnisse sind einige gegebene Limitationen zu berücksichtigen. Eine Einschränkung bestand in der begrenzten Möglichkeit der Testung des Systems. Die simulierte Umgebung konnte die Bedingungen zwar realitätsnah nachahmen, spiegelten jedoch nicht die volle Komplexität und die spezifischen Herausforderungen einer medizinischen Einrichtung wider.

Sicherheitstest gehörten nicht in den Kontext der prototypischen Umsetzung innerhalb dieser Arbeit. Das System verarbeitet zwar keine sensiblen Patientendaten, dennoch sollte in der Praxis sichergestellt werden, dass die Datenkommunikation und -speicherung vor unbefugtem Zugriff geschützt wäre. Die Möglichkeit, dass Hubs einen Angriffspunkt für das Eindringen in die IT-Infrastruktur der Einrichtungen darstellen könnten, müsste zusätzlich beachtet werden.

Die Begrenzung auf die prototypische Entwicklung hatte die Erfüllung der elementaren Grundfunktionen zum Ziel. In Kap. 2.3. „Abgrenzung der Arbeit“ wurde näher darauf eingegangen, was eine Weiterentwicklung des prototypischen ATS für Möglichkeiten der zusätzlichen Informationsgewinnung bieten könnte.

Ausblick der weiteren Entwicklung

Die Ergebnisse dieser Arbeit eröffnen mehrere vielversprechende Perspektiven für die Anwendung und Weiterentwicklung des Systems. Ein sinnvoller nächster Schritt wäre die Erweiterung des frei zur Verfügung gestellten Quellcodes des ATS in Bezug auf die beschriebenen unausgeschöpften Möglichkeiten der Informationsgewinnung sowie ein Ausbau der Webanwendung. Die erhobenen Daten könnten beispielsweise als Grundlage für eine verbesserte Lagerhaltung, Bedarfsplanung und Auslastung sorgen, was evtl. direkte Leistungssteigerungen und sicher Einsparungen für medizinische Einrichtungen erlauben würde. Zur Vermeidung nicht erlaubter Datenzugriffe sollten eine Nutzerverwaltung, Ansichtsrechte sowie die Anforderungen des „Open Web Application Security Project“ umgesetzt werden [49].

Ein weiterer wichtiger Schritt liegt in der Verbesserung der Nutzbarkeit des Systems. Nachfolgend werden beispielhaft einige mögliche Entwicklungen genannt, die diese erhöhen würden:

- Hubs und insbesondere Beacons sollten der DB unkompliziert hinzugefügt werden können, z. B. Beacons über eine hierzu definierte Basisstation mit minimaler Sensitivität
- Beacons sollten bei einer Befestigung an einem MP diesem leicht zugeordnet werden können, z. B. mittels QR-Codes an beiden Objekten
- Nach erfolgreicher Kopplung von Beacons/MP sollte es ein Feedback geben, um Fehler zu vermeiden
- Die DB sollte mit gängigen Geräteverwaltungssoftwares verknüpft werden können, um die Daten beider Systeme einheitlich pflegen und einsehen zu können
- Es sollte eine Live-Karte mit allen MP und einer Suchfunktion geben, die über die Webanwendung aufgerufen werden kann
- Es sollte bei der Suche nach einem MP automatisch ein Pfad auf der Karte gezeigt werden, der den kürzesten Weg dorthin beschreibt
- Ein Hub mit Display könnte die in einem Raum vorhandenen MPs und Paarungen anzeigen
- Ein Stationsmonitor könnte eine rein stationsbezogene Sicht auf die Daten bereitstellen

Zusätzliche Ansätze für die Weiterentwicklung zu einem marktreifen Produkt könnten sich aus der Durchführung von Feldtests in medizinischen Einrichtungen ergeben. Mögliche Themenbereiche wären z. B. Leistung, Zuverlässigkeit und Sicherheit.

Zusätzlich ließe sich der Einsatz des Systems mit wenig Aufwand auch auf andere Anwendungsbereiche erweitern. Neben der Überwachung von MP könnte es beispielsweise für die Patientenüberwachung oder die Ortung weiterer Gerätschaften, wie Tablets, eingesetzt werden. Auch ein Einsatz in anderen Branchen, in denen die Ortung und Verwaltung von Ausrüstung und Zubehör von Bedeutung sind, wäre denkbar.

7. Fazit

Diese Bachelorarbeit im Rahmen des Studiengangs Medizintechnik beschreibt eine prototypische Entwicklung eines innovativen Asset-Tracking-Systems (ATS) für medizinische Einrichtungen. Ziel war es, ein flexibles und anpassbares System zu entwickeln, das sowohl die Ortung von Medizinprodukten (MP) als auch deren jeweilige Zuordnung zueinander ermöglicht.

Die Problemstellung „Entwicklung eines Asset-Tracking-Systems mit automatisierter Zubehörzuordnung für medizinische Einrichtungen“ ergab sich aus eigener Arbeitserfahrung in medizintechnischen Abteilungen Hamburger Krankenhäuser. Sie wurde durch eingehende Recherche, eine eigens erstellte Umfrage gerichtet an medizinische Einrichtungen in Deutschland, und die Durchführung einer Marktanalyse untermauert.

Es stellte sich heraus, dass bestehende Systeme nicht das umfassende Potential an wünschenswerter und leistbarer Funktionalität ausschöpfen, das ein ATS für den Gebrauch in medizinischen Einrichtungen bieten könnte. Daraus entstand das in dieser Arbeit dokumentierte Ziel, ein entsprechendes Open-Source-System zu erstellen, das aus kostengünstigen Standardkomponenten besteht und über eine Ortung hinausgehend zusätzlich eine (teil-) automatisierte Erfassung der kombinatorischen Nutzung von MP ermöglicht.

Wesentlich dafür war die Entwicklung eines umfassenden Lastenhefts. Es dokumentiert mithilfe von funktionalen und nicht-funktionalen Anforderungen, welche Leistungen das ATS erbringen soll und diente als Grundlage für die weiteren Entwicklungsphasen.

Auf der Basis des Lastenhefts wurde ein detailliertes Pflichtenheft mit Informationen zur Planung der Umsetzung erstellt. Es definiert ein Systemkonzept und beschreibt die Hard- und Softwarekomponenten sowie deren Interaktionen miteinander.

Im Anschluss wurden die jeweiligen Systemkomponenten ausgearbeitet und entwickelt. Dazu gehörte die Auswahl geeigneter Hardware für Beacons und Hubs, die Anpassung des mosquitto MQTT-Brokers zur Übertragung eines UNIX-Zeitstempels, die Entwicklung eines Python-Microservices zur Verarbeitung der Beacondaten und die Einrichtung einer relationalen DB zur Speicherung aller relevanten Daten. Zusätzlich wurde eine rein funktionale Webanwendung für eine einfache Verwaltung und Analyse der erfassten Daten entwickelt.

Nach Abschluss der Entwicklung folgten umfangreiche Tests der einzelnen Komponenten. Diese stellten sicher, dass alle Einzelkomponenten ihre jeweils geforderte Funktion fehlerfrei erfüllen. Die durchgeführten Integrationstests gaben Aufschluss darüber, ob alle Systemkomponenten innerhalb der gewählten Laufzeitumgebung einwandfrei zusammenarbeiteten. Mit einem räumlich der Realität

nachgestellten Setting eines Einsatzortes wurde überprüft, ob die Datenverarbeitung korrekt und schnell genug erfolgt. Außerdem wurde die Systemfunktion unter simulierter hoher Last geprüft.

Die geplante prototypische Entwicklung eines ATS für medizinische Einrichtungen auf der Basis von kostengünstigen Standardkomponenten und bereits vorhandener Infrastruktur war erfolgreich. Die Tests zeigen, dass das entwickelte System den Anforderungen entsprechend entwickelt wurde und fehlerfrei funktioniert.

Die entwickelten Systemkomponenten bilden ein ATS, das dazu in der Lage ist, die Ortung und Zuordnung von MP und deren Zubehör effizient zu handhaben. Es hat das Potenzial, beschriebene Problemstellungen in Gesundheitseinrichtungen signifikant zu verringern, indem es die Effizienz in der Verwaltung von MP erhöht. Bei der Implementierung eines solchen Systems wäre von Kosteneinsparungen und einer Erleichterung in Hinblick auf rechtsnormenkonformes Arbeiten auszugehen. Die vollständig als Open-Source-Code zugänglichen Komponenten bieten eine solide und kostengünstige Grundlage mit nahezu unbegrenzten Möglichkeiten der Anpassung und Weiterentwicklung. Ein Aspekt, der aufgrund der allgemein hohen Kosten im medizintechnischen Sektor Vorteile verspricht.

8. Literaturverzeichnis

1. Mutshatshi TE, Mothiba TM, Mamogobo PM, Mbombi MO. Record-keeping: Challenges experienced by nurses in selected public hospitals. Curations. 2018 Jun; 1(41).
2. Banat D. Conditions of Cost-Effectiveness for the Use of Real-Time Location Systems (RTLS) in Medical Facilities and the Benefits of Effective Implementation Process. EDUCATION OF ECONOMISTS AND MANAGERS. 2020 Nov; 58(4).
3. Gholamhosseini L SFSA. Hospital Real-Time Location System (A Practical Approach in Healthcare): A Narrative Review Article. Iranian Journal of Public Health. 2019 Apr: p. 783.
4. Barraclough D. The Best Hospital Asset Tracking Software. [Online].; 2023 [cited 2024 03 04. Available from: https://www.expertmarket.com/asset-tracking/best-healthcare-asset-tracking-system.
5. RF Code. Hospital Asset Tracking: What You Need to Know. [Online].; 2021 [cited 2024 03 04. Available from: http://www.rfcode.com/blog/healthcare-asset-tracking.
6. PcsInfinity Private Limited. Hospital and Healthcare Asset Management and Tracking. [Online].; 2023 [cited 2024 03 04. Available from: https://www.assetinfinity.com/solutions/hospital.
7. BeaconTrax. Asset Tracking for Healthcare. [Online].; 2024 [cited 2024 03 04. Available from: https://www.beacontrax.com/asset-tracking-for-healthcare/.
8. midmark. Midmark RTLS launches cloud-based BLE asset tracking. [Online].; 2021 [cited 2024 03 04. Available from: https://www.midmark.com/newsroom/news-releases/2021/06/02/midmark-rtls-launches-cloud-based-ble-asset-tracking.
9. kontakt.io. Achieving Room-Level Medical Device Tracking Using Bluetooth Low Energy Solutions. [Online].; 2024 [cited 2024 03 04. Available from: https://kontakt.io/blog/achieving-room-level-medical-device-tracking-using-bluetooth-low-energy-solutions/.
10. Pycube. An all-in-one platform for all your asset management. [Online].; 2024 [cited 2024 03 04. Available from: https://www.pycube.com/.
11. Paragon Group. Improve asset visibility and utilisation, save capital expenditure and cut down time spent locating equipment. [Online].; 2024 [cited 2024 03 04. Available from: https://www.rfiddiscovery.com/en-us/solutions/asset-tracking-healthcare.
12. CYBRA. RFID Asset Tracking in Hospitals. [Online].; 2024 [cited 2024 03 04. Available from: https://cybra.com/asset-tracking-hospitals-healthcare/.

13. Securitas Healthcare. Asset Management. [Online].; 2024 [cited 2024 03 04. Available from: https://www.securitashealthcare.com/solutions/asset-management.

14. RFiD Discovery. Bluetooth Low Energy (BLE). [Online].; 2024 [cited 2024 03 04. Available from: https://www.rfiddiscovery.com/en-us/content/bluetooth-low-energy-ble.

15. Sokolova B. Everything you need to know about BLE asset tracking. [Online].; 2023 [cited 2024 03 04. Available from: https://www.mapon.com/en/blog/2023/05/everything-to-know-about-ble-asset-tracking.

16. HPE Aruba Networking. BLE Asset Tracking Helps Hospitals Find Medical Devices Stat. [Online].; 2017 [cited 2024 03 04. Available from: https://blogs.arubanetworks.com/industries/ble-asset-tracking-helps-hospitals-find-medical-devices-stat/.

17. HPE Aruba Networking. Aruba Beacons Data Sheet. [Online].; 2022 [cited 2024 03 04. Available from: https://www.arubanetworks.com/resource/aruba-beacons-data-sheet/.

18. Shiklo B. 8 Vorgehensmodelle der Softwareentwicklung: mit Grafiken erklärt. [Online].; 2019 [cited 2024 03 20. Available from: https://www.scnsoft.de/blog/vorgehensmodelle-der-softwareentwicklung.

19. Narayan DR. Study of Various Software Development Methodologies. EPRA International Journal of Multidisciplinary Research (IJMR). 2021 Apr; 7(4).

20. Mark Smith RSLSaJMM. Best Care at Lower Cost. 1st ed. Sciences NAo, editor. Washington, DC: National Academy of Sciences; 2013.

21. Julia Adler-Milstein CMDPKGFCWDCTSAKJ. Electronic Health Record Adoption In US Hospitals: Progress Continues, But Challenges Persist. Health Affairs. 2015 Dec; 34(12).

22. Europäische Kommission. Gestaltung der digitalen Zukunft Europas; European Hospital Survey - Benchmarking Deployment of eHealth services (2012-2013). [Online].; 2014 [cited 2024 04 13. Available from: https://digital-strategy.ec.europa.eu/de/node/8744.

23. E. Mossialos ADRODS. The Commonwealth Fund: International Profiles of Health Care Systems. [Online]. New York; 2017 [cited 2024 04 13. Available from: https://doi.org/10.26099/xp8v-kz31.

24. Sooyoung Yoo SKEKEJKHLHH. Real-time location system-based asset tracking in the healthcare field: lessons learned from a feasibility study. BMC Medical Informatics and Decision Making. 2018 Sep; 18.

25. Țăran AM, Mustea L, Vătavu S, Lobonț OR, Luca MM. Challenges and Drawbacks of the EU Medical System Generated by the COVID-19 Pandemic in the Field of Health Systems' Digitalization.

International Journal of Environmental Research and Public Health. 2022 Apr; Big Data, Decision Models, and Public Health(2).

26. Kamel Boulos MN,&BG. Real-time locating systems (RTLS) in healthcare. International Journal of Health Geographics. 2012 Jun; 11(25).

27. Kathy O. Roper ASBA. A cost-benefit case for RFID implementation in hospitals: adapting to industry reform. Emerald Group. 2015 Apr; 33(5/6): 367-388.

28. kontakt.io. 10 Steps to Consider When Implementing RTLS in Healthcare. [Online].; 2023 [cited 2024 03 17. Available from: https://kontakt.io/blog/rtls-real-time-location-system-in-healthcare/#increasing-bed-availability.

29. Kuan F. So wählen Sie aus 8 RTLS-Technologien. [Online].; 2023 [cited 2024 03 17. Available from: https://www.mokosmart.com/de/rtls-technologies/.

30. Rodrigo Bazo CAdCLASLGdSJ,RSARdRRVFR. A Survey About Real-Time Location Systems in Healthcare Environments. Journal of Medical Systems. 2021 Feb; 45: Article Nr. 35.

31. Kontakt.io. RTLS vs. RFID: The Pros and Cons of Both for Asset Tracking and Management. [Online].; 2024 [cited 2024 06 06. Available from: https://kontakt.io/blog/rtls-vs-rfid-the-pros-and-cons-of-both-for-asset-tracking-and-management/.

32. Cox Communications, Inc.. Which Type of RTLS Tracking Technology Is Best for Hospitals? Comparing RFID vs BLE and Other RTLS Technologies. [Online].; 2022 [cited 2024 06 06. Available from: https://www.coxprosight.com/post/rfid-vs-ble-rtls.

33. Blackstone A. Understanding the different types of BLE Beacons. [Online].; 2015 [cited 2024 05 09. Available from: https://os.mbed.com/blog/entry/BLE-Beacons-URIBeacon-AltBeacons-iBeacon/.

34. Pleško J. A Guide to Beacon Technology in 2021. [Online].; 2021 [cited 2024 05 09. Available from: https://infinum.com/blog/bluetooth-beacons/.

35. Apple Inc. iBeacon. [Online].; 2024 [cited 2024 05 13. Available from: https://developer.apple.com/ibeacon/.

36. Eclipse Foundation. Eclipse Mosquitto™ An open source MQTT broker. [Online].; 2024 [cited 2024 05 07. Available from: https://mosquitto.org/.

37. Hive MQ. MQTT Tutorial: An Easy Guide to Getting Started with MQTT. [Online].; 2024 [cited 2024 03 18. Available from: https://www.hivemq.com/blog/how-to-get-started-with-mqtt/.

38. Bluetooth SIG, Inc. Specifications and Documents. [Online].; 2024 [cited 2024 03 28. Available from: https://www.bluetooth.com/specifications/specs/core-specification-5-4/.

39. Bluetooth SIG, Inc. Part A. Data Types Specification. [Online].; 2024 [cited 2024 03 28. Available from: https://www.bluetooth.com/specifications/css-11/.

40. Bluetooth SIG, Inc. Assigned Numbers. [Online].; 2024 [cited 2024 03 29. Available from: https://www.bluetooth.com/specifications/assigned-numbers/.

41. Espressif Systems. ESP-IDF iBeacon demo. [Online].; 2024 [cited 2024 03 29. Available from: https://github.com/espressif/esp-idf/tree/master/examples/bluetooth/bluedroid/ble/ble_ibeacon.

42. Espressif Systems. Mesh IP Internal Networking example. [Online].; 2024 [cited 2024 03 30. Available from: https://github.com/espressif/esp-idf/tree/master/examples/mesh/ip_internal_network.

43. Espressif Systems. ESP-WIFI-MESH Programming Guide. [Online].; 2024 [cited 2024 03 30. Available from: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp-wifi-mesh.html.

44. Espressif Systems. ESP-WIFI-MESH. [Online].; 2024 [cited 2024 03 30. Available from: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/esp-wifi-mesh.html.

45. Eclipse Foundation. mosquitto. [Online].; 2024 [cited 2024 03 31. Available from: https://github.com/eclipse/mosquitto.

46. OpenJS Foundation. About Node.js®. [Online].; 2024 [cited 2024 04 12. Available from: https://nodejs.org/en/about.

47. Nuxt. Introduction. [Online].; 2024 [cited 2024 04 25. Available from: https://nuxt.com/docs/getting-started/introduction.

48. EMQ Technologies Inc. Your All-in-One MQTT Client Toolbox. [Online].; 2024 [cited 2024 05 30. Available from: https://mqttx.app/.

49. OWASP® Foundation. OWASP Top Ten. [Online].; 2021 [cited 2024 06 15. Available from: https://owasp.org/www-project-top-ten/.

50. Shenzhen Yuda Technology Co., LTD. ESP32-C3 entwicklungs board esp32 c3 supermini wifi bluetooth esp32c3. [Online].; 2024 [cited 2024 05 15. Available from:

https://ae01.alicdn.com/kf/S926764ab5703404b8df005c03f0ad18ep/ESP32-C3-entwicklungs-board-esp32-c3-supermini-wifi-bluetooth-esp32c3.jpg_.webp.

51. Holyiot. Holyiot nrf52810 Beacon Tag Bluetooth 5,0 Modul mit geringem Strom verbrauch. [Online].; 2024 [cited 2024 05 15. Available from: https://ae01.alicdn.com/kf/S409e158e9afc4c1bbc311d41f7b3336aO/Holyiot-nrf52810-Beacon-Tag-Bluetooth-5-0-Modul-mit-geringem-Strom-verbrauch.jpg_.webp.

52. Holyiot. Holyiot nrf52810 beacon ble 5.0 Bluetooth-Modul Innen position ierung programmier bare Langstrecken-Tracke für ibeacon-Automatisierung module. [Online].; 2024 [cited 2024 05 15. Available from: https://ae01.alicdn.com/kf/Sab205dbd6fac4ca3b3256d173c6d3049o/Holyiot-nrf52810-beacon-ble-5-0-Bluetooth-Modul-Innen-position-ierung-programmier-bare-Langstrecken-Tracke-f.jpg_.webp.

53. Dormando. Memcached. [Online].; 2024 [cited 2024 05 16. Available from: http://memcached.org/.

54. Espressif. ESP-WIFI-MESH Programming Guide. [Online].; 2024 [cited 2024 05 27. Available from: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp-wifi-mesh.html.

55. Patel K. Universal application code structure in Nuxt.js. [Online].; 2018 [cited 2024 04 25. Available from: https://medium.com/free-code-camp/universal-application-code-structure-in-nuxt-js-4cd014cc0baa.

9. Anhang

A1 - Rechtsnormen

Folgend wird erläutert, was in dieser Arbeit unter der „vereinfachten Erfüllung geltenden Rechts“ verstanden wird. Dafür werden verschiedene Rechtsnormen und deren Interpretation bezogen auf die Pflichten der Betreiber von MP herangezogen. Dies geschieht beispielhaft.

- Führung eines Bestandsverzeichnisses (§ 13 MPBetreibV): Ein ATS kann bei der Führung eines genauen Bestandsverzeichnisses aller aktiven Medizinprodukte helfen, indem es kontinuierlich Standort und Status jedes Gerätes aktualisiert.

- Kontrolle und Meldung von Vorkommnissen (§ 6 MPBetreibV): Ein ATS kann die genaue Dokumentation und schnelle Meldung von Vorkommnissen erleichtern, indem es detaillierte Informationen über den Einsatz von Medizinprodukten bereitstellt.

- Wartung und Instandhaltung (§ 7 MPBetreibV): Ein ATS kann helfen Wartungs- und Instandhaltungspläne zu überwachen und sicherzustellen, dass alle Medizinprodukte gemäß den Herstellervorgaben und gesetzlichen Bestimmungen instandgehalten werden.

- Sicherstellung des ordnungsgemäßen Betriebs von Medizinprodukten (MPBetreibV § 4 Abs. 1): Durch Überwachung der Produktkombinationen und deren korrekte Anwendung kann ein ATS dazu beitragen, dass die Sicherheitsanforderungen eingehalten werden.

- Schulung und Weiterbildung des Personals (§ 4 Abs. 2 MPBetreibV): Ein ATS kann durch Bereitstellung von Informationen über die korrekte Handhabung und Kombination von Medizinprodukten die Schulung des Personals unterstützen.

- Behandlungsvertrag (§§ 630a ff. BGB): Betreiber von Krankenhäusern und Kliniken sind verpflichtet, die Behandlung nach den allgemein anerkannten fachlichen Standards zu erbringen. Eine präzise Ortung von Medizinprodukten kann dazu beitragen, den Überblick über die verwendeten Geräte zu behalten und sicherzustellen, dass stets die geeignetsten und sichersten Produkte zur Anwendung kommen. Die Sicherstellung der Wartungszustände unterstützt dabei, zum jeweiligen Behandlungszeitpunkt auf funktionsbereite Geräte zugreifen zu können.

- Schadensersatzpflicht (§ 823 BGB): Behandler müssen Maßnahmen ergreifen, um Schäden bzw. Haftungsfälle zu vermeiden. Durch die genaue Verfolgung und Lokalisierung von Medizinprodukten kann die Gefahr von Fehlbedienungen oder dem Verlust von Geräten minimiert werden, was wiederum das Risiko von Schadensersatzforderungen senkt.

- § 4 ArbSchG: Der Arbeitgeber hat die Pflicht, die Arbeit so zu gestalten, dass die Gefährdung der psychischen und physischen Gesundheit von Arbeitnehmern vermieden wird (§ 4 Nr. 1). Hierzu gehört z.B. auch, dass Behandlungsfehler, die die psychische Gesundheit der Mitarbeiter beeinträchtigen können, durch ordnungsgemäße Bereitstellung und Kontrolle von Arbeitsmitteln vermieten werden. Der Arbeitgeber hat den Stand der Technik zu beachten (§ 4 Nr. 3 Alt. 1) und beim Sicherstellen des Arbeitsschutzes auch die Arbeitsorganisation zu beachten. Ein ATS kann hierbei maßgeblich zur erleichterten Organisation beitragen.

A2

Bedarfs- und Bestands-Umfrage zur Nutzung von Systemen zur Echtzeit-Ortsbestimmung medizintechnischer Geräte

Diese Umfrage wurde im Rahmen einer Bachelorarbeit des Medizintechnikstudiums an der HAW-Hamburg durchgeführt und dient der Bedarfs- und Bestands-Analyse zur Nutzung von Systemen zur Echtzeit-Ortsbestimmung medizintechnischer Geräte (RTLS bzw. Asset Tracking Systems).
Ziel der Arbeit ist die Entwicklung eines flexiblen und kostengünstigen Systems, das sowohl die Ortung medizintechnischer Geräte als auch deren jeweilige Zubehörzuordnung ermöglicht.
Die Ergebnisse der Umfrage fließen in die Gestaltung und Abgrenzung der entstehenden Lösung ein.

An dieser Stelle möchte ich mich nochmals ausdrücklich bei allen Teilnehmenden bedanken, die sich die Zeit genommen haben, diese Umfrage zu beantworten!

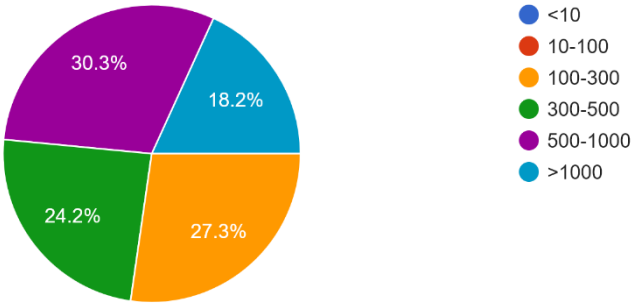
Ihr Linus Norden

Umfrageergebnisse

Basisinformation

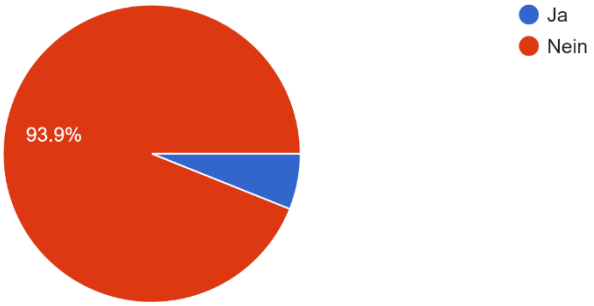
Wie viele Bettplätze sind in Ihrer Einrichtung vorhanden?

33 responses



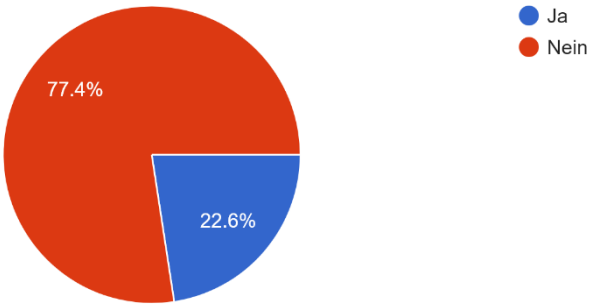
Nutzt Ihre Einrichtung Asset Tracking oder RTLS für Medizinprodukte?

33 responses



Planen Sie aktuell oder in naher Zukunft die Einführung eines Asset Tracking / RTLS?

31 responses

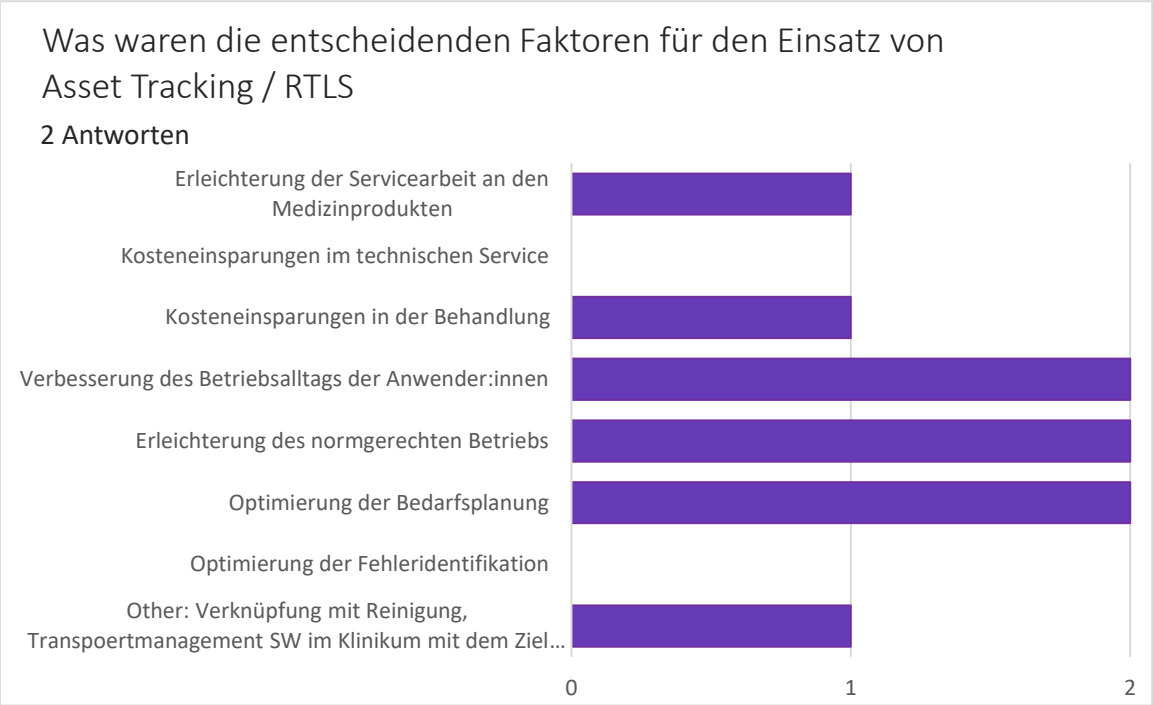


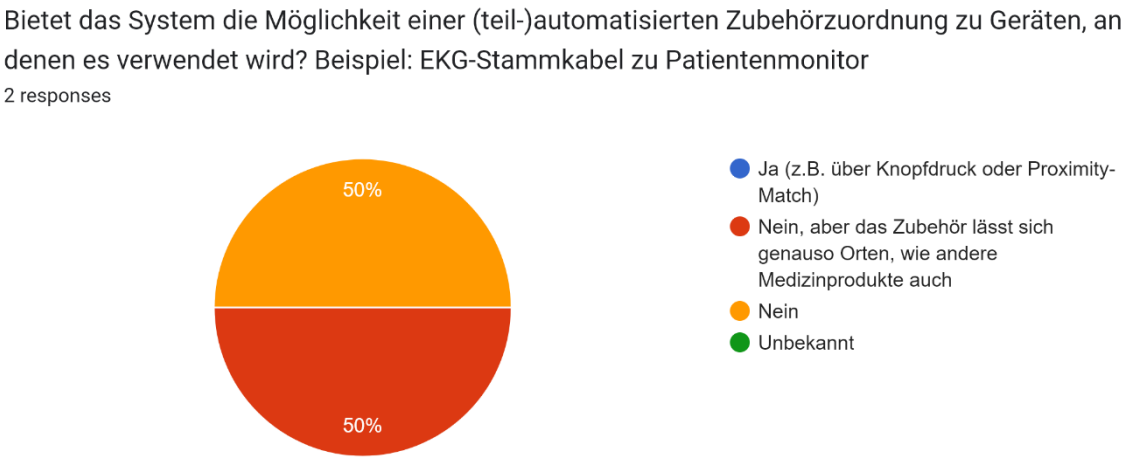
Fragegruppe: Einsatz von Asset Tracking / RTLS

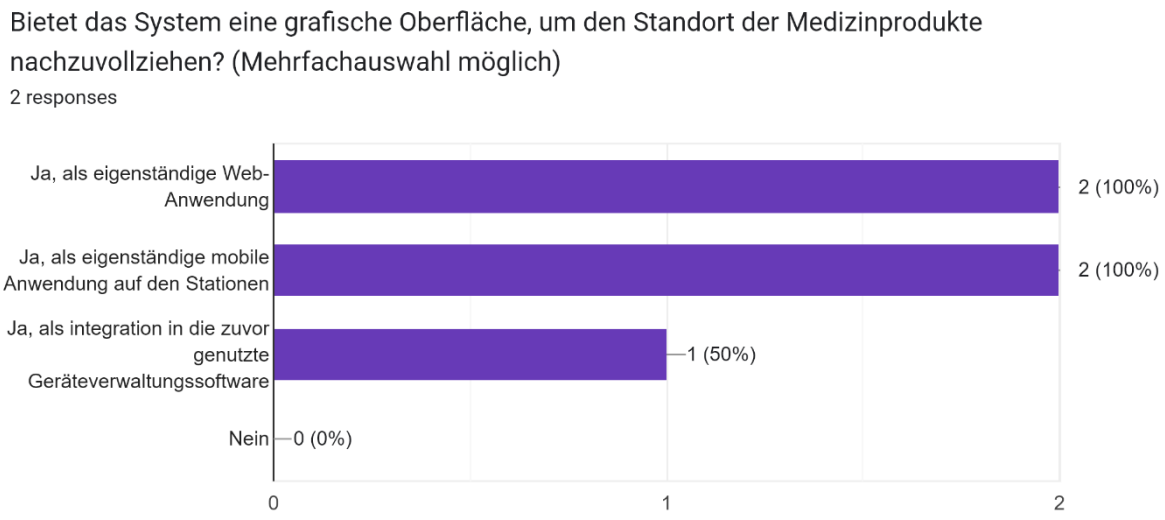
Welches System wird genutzt? (Optional, Hersteller und Produktbezeichnung)

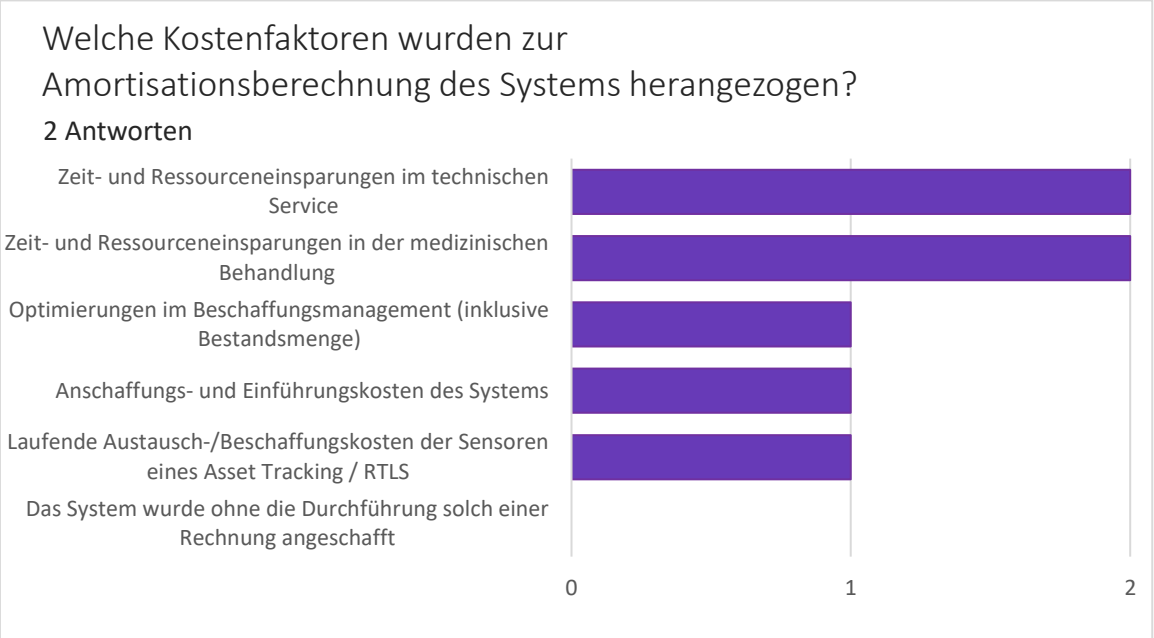
1 Antwort

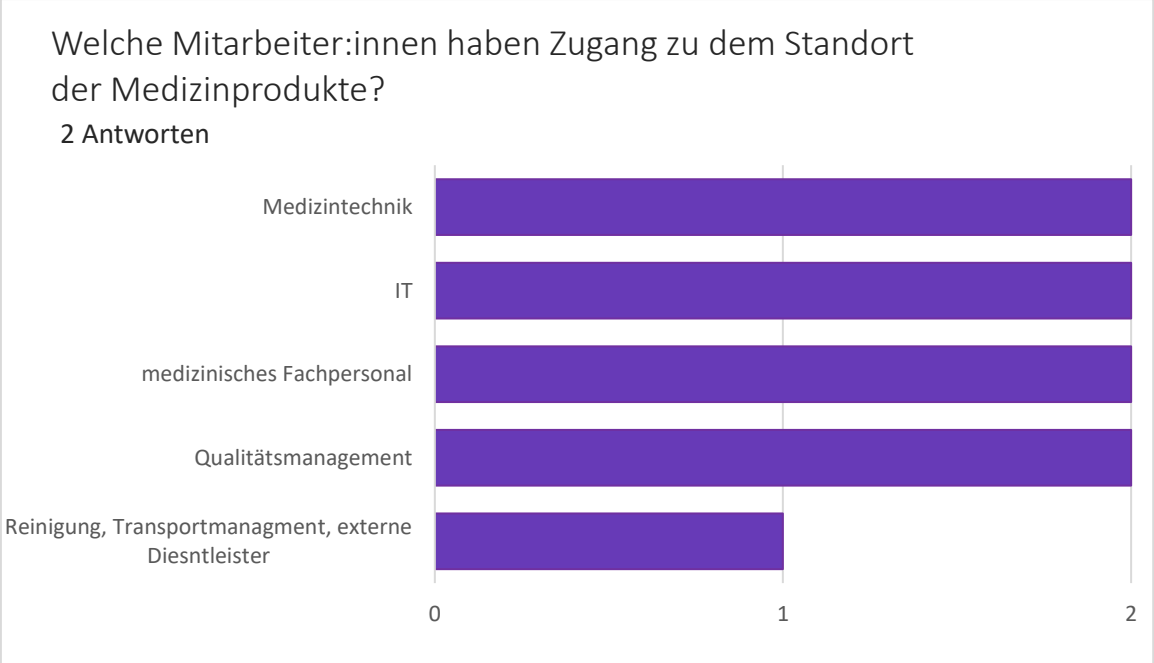
- Dräger, aber nicht mehr aktiv







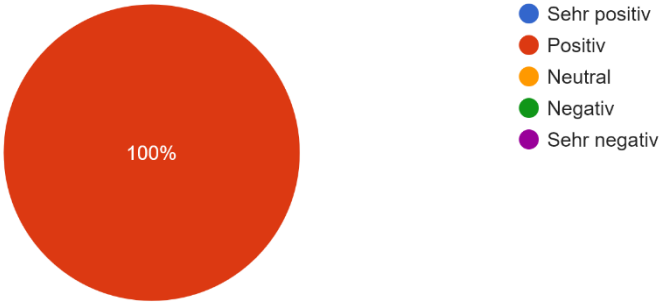




A2 – Umfrage ATS/RTLS

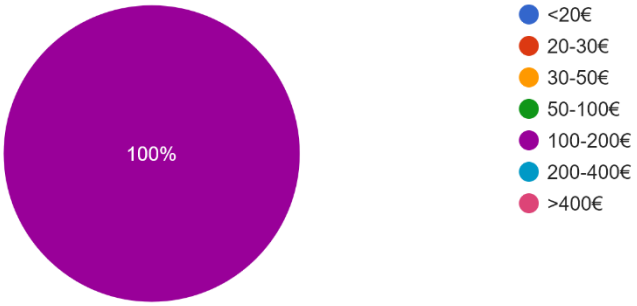
Wie war das Feedback des medizinischen Personals nach Einführung des Systems? (Optional)

2 responses



Wie viel hat die Anschaffung des Systems in etwa pro Bettplatz gekostet (Optional, Grobe Schätzung genügt)

1 response



Welche Informationen oder Funktionen vermissen Sie in Ihrem aktuell genutzten System? (Optional)

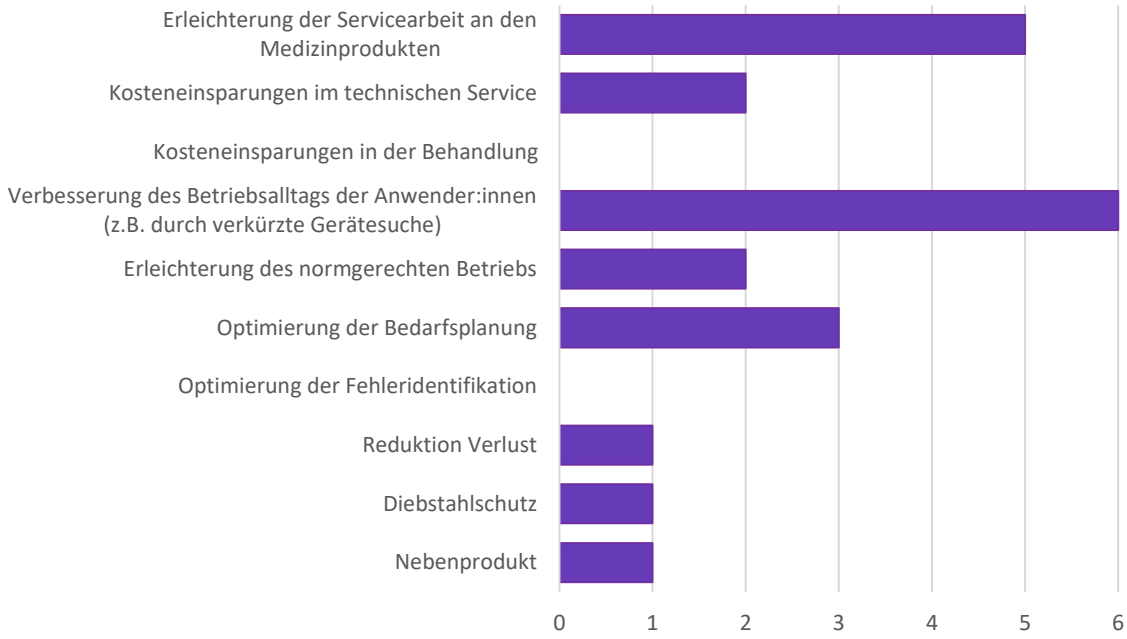
2 Antworten

- Besseres UI zur Verwaltung von Tags und Geräten.
- Keine Ortung wenn Batterie leer, Kein Knopf um ggf Zustand rück zu melden wie Bett Defekt, Bett Gereinigt, etc. Ortung nicht 100% Zimmergenau

Fragegruppe: Geplante Anschaffung eines Asset Tracking / RTLS

Was sind entscheidende Faktoren für den geplanten Einsatz von Asset Tracking / RTLS

7 Antworten

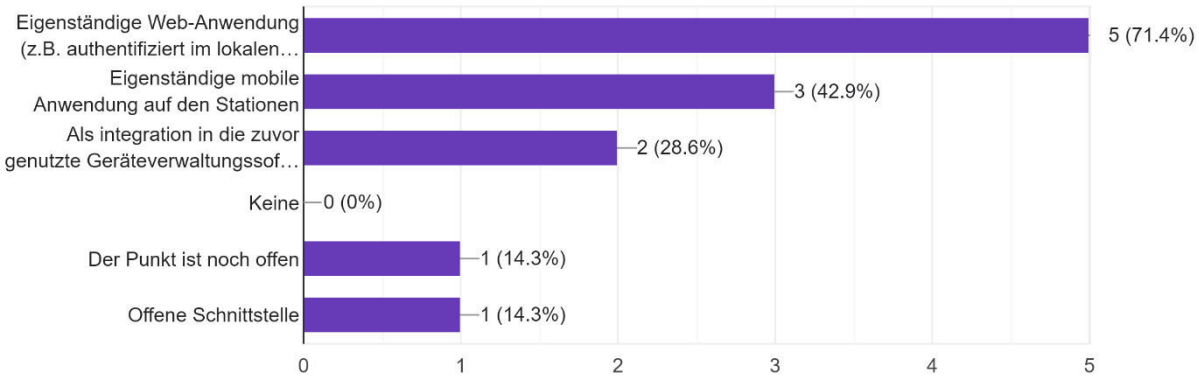


Erwarten Sie vom System die Möglichkeit einer (teil-)automatisierten Zubehörzuordnung zu Geräten, an denen es verwendet wird?

7 responses

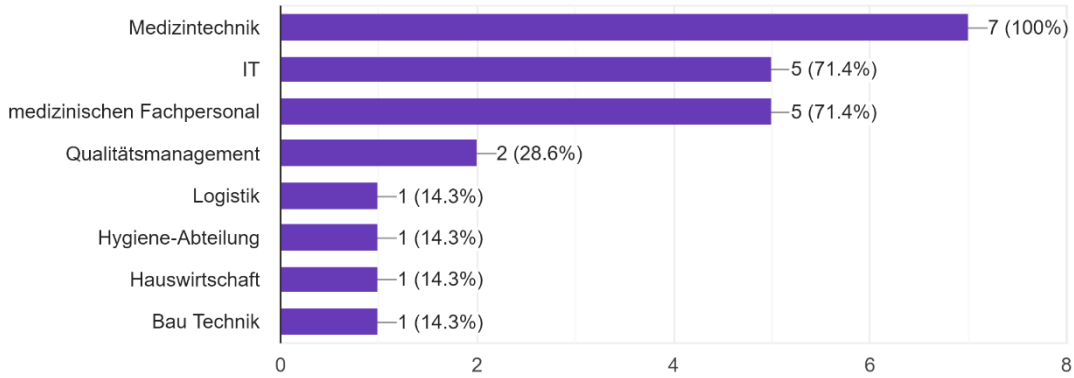
Welche grafische Oberfläche soll angeboten werden, um den Standort der Medizinprodukte nachzuvollziehen? (Mehrfachauswahl möglich)

7 responses



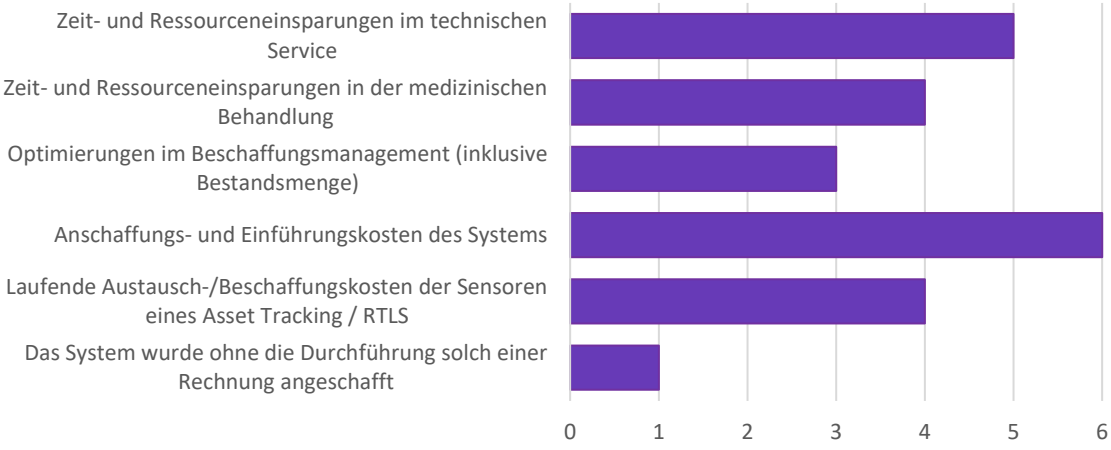
Welche Mitarbeiter:innen sollen Zugang zu dem Standort der Medizinprodukte haben? (Mehrfachauswahl möglich)

7 responses



Welche Kostenfaktoren würden Sie zur Amortisationsberechnung des Systems heranziehen?

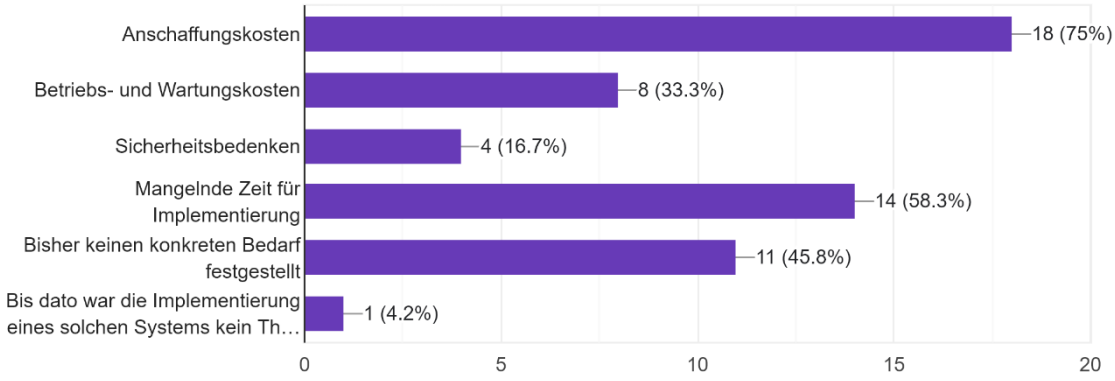
7 Antworten



Fragegruppe: Keine Nutzung oder Planung der Nutzung von Asset Tracking oder RTLS

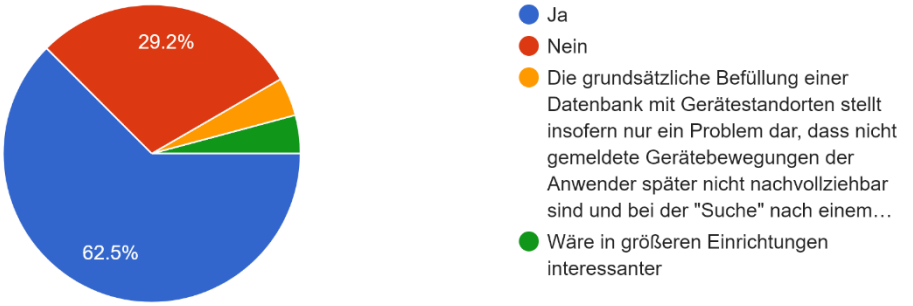
Wenn nein, aus welchen Gründen nicht? (Mehrfachauswahl möglich)

24 responses



Nehmen Sie die manuelle Pflege von Gerätestandorten und Zubehörzuordnungen als ein Problem wahr?

24 responses



Würden Sie, als technisches Personal, sich solch ein System wünschen? (Optional)

24 responses

A3

Lastenheft
Asset Tracking in Krankenhäusern und Kliniken

Projekt im Rahmen der Bachelorarbeit.

Inhaltsverzeichnis

1. Einleitung

1.1. Zweckbestimmung und Begriffsdefinition

Die hier beschriebene Kombination aus Soft- und Hardwarekomponenten soll das Asset Tracking von Medizinprodukten in einem klinischen Umfeld, wie Krankenhäusern, Kliniken und sonstigen medizinischen Einrichtungen, ermöglichen. Der Begriff Asset Tracking System (ATS) umfasst sämtliche im Endprodukt verwendeten Soft- und Hardwarekomponenten, wie Microcomputer, DB-Server, Webserver, Nachrichtenprotokolle etc. Es handelt sich also um den Zusammenschluss dieser Komponenten und die dadurch gemeinsam ermöglichten Funktionen. Die folgenden Anforderungen sind überwiegend aus der Nutzerperspektive formuliert. In einigen Fällen werden jedoch bereits spezifische technische Anforderungen gestellt, welche aus dem Einsatzumfeld in med. Einrichtungen hervorgehen.

2. Funktionale Anforderungen

2.1. Grundlegende Funktionen

1. Das ATS soll die Ortung von Medizinprodukten (MP) ermöglichen. (siehe auch 2.2.)
2. Das ATS soll die Speicherung des detektierten Ortes von MP ermöglichen.
3. Das ATS soll die Erkennung einer kombinatorischen Nutzung von MP ermöglichen. (siehe auch 2.3.)
4. Das ATS soll die Speicherung der Zuordnung mehrerer detektierter MP zueinander ermöglichen.
5. Das ATS soll eine Historie relevanter Datenänderungen zu Verfügung stellen. (siehe auch 2.4.1.)
6. Das ATS soll die Datenweiterleitung innerhalb bestehender lokaler Netzwerke ermöglichen. (siehe auch 2.4.2.)
7. Das ATS soll über eine Web-Schnittstelle Erfassungsdaten zur Integration in bestehende Geräteverwaltungssoftwares zur Verfügung stellen. (siehe auch 2.4.3.)
8. Es soll eine Web-Anwendung geben, welche die Interaktion mit den Daten des ATS ermöglicht. (Siehe auch 2.5.)
9. Das ATS soll skalierbar sein, um mit einer variablen Anzahl von MP, Bereichen und Stationen umgehen zu können. (siehe auch 2.6.)

2.2. Ortung von MP

1. Das ATS soll dazu in der Lage sein, alle registrierten MP innerhalb einer Einrichtung zu Orten.
2. Ein MP soll dabei zuverlässig einem Raum oder Flurabschnitt zugeordnet werden können.
3. Dafür soll das ATS die Möglichkeit besitzen alle Räume, Bereiche etc. einer Einrichtung als Datensätze zu speichern.
4. Die Zuordnung des aktuellen Standorts eines MP soll mithilfe dieser Datensätze erfolgen.
5. Ein MP soll mindestens minutenaktuell geortet werden, also Intervall von $t \leq 60$ Sekunden.

2.3. Zuordnung von MP zu MP-Kombination

1. Das ATS soll über einen einfachen Input, wie z.B. Knopfdruck, die Kopplung von MPs ermöglichen.
2. Es sollen nur MP miteinander gekoppelt werden können, welche für den Gebrauch miteinander bestimmt sind.
3. Dafür soll eine Prüfung auf die ‚Zulässige Verwendung miteinander' geschehen, wenn zwei oder mehr MP eine Kopplung initiieren.
4. Das Koppeln soll nur dann ausgeführt werden, wenn sich die MP an demselben Ort befinden.
5. Die Kopplung von MP soll aufgelöst werden, wenn sie sich nicht mehr an demselben Ort befinden.

2.4. Datenmanagement

1. Alle für das ATS nötigen Datensätze sollen in einer gängigen Datenbankinfrastruktur gespeichert werden. (MySQL, Maria-DB o.Ä.)
2. Alle Softwarekomponenten des ATS sollen, je nach Bedarf, sowohl auf externen als auch auf internen Servern gehostet werden können.

2.4.1. Historie

1. Die folgenden Datenänderungen sollen für mindestens 24 Monate gespeichert werden und nachträglich einsehbar sein:
    - Standortänderung MP
    - Änderung MP-Kombination
    - MP-Inaktivierung
    - Raum-Inaktivierung
    - Bereichs-Inaktivierung
2. Die Historie dieser Datensätze soll den Anfangs- und Endzeitpunkt des Vorgangs enthalten. (z.B. MP vom 01.01.2001 bis 12.12.2012 aktiv gewesen)

2.4.2. Datenweiterleitung innerhalb vorhandener Netzwerke

1. Bei der notwendigen Weiterleitung von Daten innerhalb lokal verfügbarer Netzwerke der Einrichtung, darf der Traffic die Funktion jeglicher mit dem Netzwerk verbundener Gerät nicht beeinflussen. (So wenig wie möglich, so viel wie nötig)
2. Die Kommunikation jeglicher Hardwarekomponenten untereinander soll außerhalb der lokal verfügbaren Netzwerke der Einrichtungen erfolgen. (z.B. WiFi-Mesh)
3. Die Anzahl der mit den lokal verfügbaren WLAN verbundenen Hardwarekomponenten soll in jedem Bereich (z.B. Krankenhausstation) auf max. 2 begrenzt werden.

2.4.3. Datenaustausch mit vorhandenen Verwaltungssoftwares

1. Es soll die Möglichkeit bestehen, relevante Datensätze, wie die von MP (SN, Name, Hersteller etc.), über eine kommaseparierte Liste aus einer bereits vorhandenen Geräteverwaltungssoftware in die Datenhaltung des ATS zu importieren.
2. Es soll die Möglichkeit bestehen, relevante Datensätze, wie den Standort von MP, als kommaseparierte Liste aus der Datenhaltung des ATS zu exportieren.

2.5. Web-Anwendung

1. Alle Ansichten der Webanwendungen sollen sich an modernen UI-Designs orientieren, um die Nutzung zu erleichtern.
2. Die Webanwendung soll mehrere Ansichten bereitstellen, welche folgende Inhalte darstellen sollen:
    - „MP-Suchansicht“: Suche nach MP, anschließendes Anzeigen des aktuellen Standorts.
    - „Raum- / Bereich-Suchansicht“: Suche nach Raum/Bereich, Ansicht aller MP an diesem Ort als Tabelle.
    - „MP-Serviceansicht“: Tabelle mit allen MP, mit Möglichkeit zur Anpassung von Datensätzen.
    - „Raum-Serviceansicht“: Tabelle mit allen Räumen, mit Möglichkeit zur Anpassung von Datensätzen.
    - „Bereich-Serviceansicht“: Tabelle mit allen Bereichen, mit Möglichkeit zur Anpassung von Datensätzen.
    - „MP-Kombination-Serviceansicht“: Tabelle mit allen zulässigen MP-Kombinationen, mit Möglichkeit zur Anpassung von Datensätzen.
    - „Hardware-Serviceansicht“: Ansicht zur Pflege und Einrichtung aller für die Ortung benötigten Hardwarekomponenten (z.B. Sender, Empfänger o.Ä.)
3. Alle Nutzeroberflächen sollen die Sprachen Deutsch und Englisch unterstützen.

2.5.1. Bedienbarkeit durch medizinisches Personal

1. Das medizinische Personal soll die Möglichkeit haben MP durch Knopfdruck miteinander zu koppeln.
2. Das medizinische Personal soll Zugriff auf die folgenden Ansichten der Webanwendung haben:
    - MP-Suchansicht
    - Raum- / Bereich-Suchansicht

2.5.2. Bedienbarkeit durch technische Mitarbeitende (MT/IT) sowie Herstellerpersonal

1. Das MT-/ IT-Personal soll Zugriff auf alle Ansichten der Webanwendung haben.
2. Damit soll dem MT-/IT-Personal die Modifikation aller vorhandenen Datensätze ermöglicht werden.

2.6. Skalierbarkeit

1. Das ATS soll so umgesetzt werden, dass die Soft- und Hardwarearchitektur eine nahezu unbegrenzte Skalierung zulassen. (Alles von unter 10 bis über 10.000 Bettplätze)

3. Nichtfunktionale Anforderungen

3.1. Anforderungen an Eigenschaften der Verwendeten Materialien und den Formfaktor der zur Umsetzung benötigten Komponenten

1. Die in den medizinischen Einsatzbereichen verwendeten Hardwarekomponenten sollen gegen Umwelteinflüsse geschützt sein, wie sie in dem Einsatzbereich auftreten könnten:
    - Staub
    - Feuchtigkeit
    - UV-Strahlung
    - Schlag- und Stoßbeanspruchung
    - Desinfektion mit gängigen Mitteln
2. Die Betriebsdauer aktiver Komponenten zur Positionserfassung soll ohne Wartung oder Batterietausch minimal 2 Jahre betragen.
3. Ein Batterietausche oder ein Aufladen von Komponenten zur Positionserfassung soll schnell, einfach und mit Standardkomponenten (z.B. standardisierten Batteriezellen) möglich sein.
4. Im Falle einer Wartung von Komponenten zur Positionserfassung sollen die Schutzklasse (z.B. IP67) sowie alle o.g. Eigenschaften weiterhin erhalten bleiben.
5. Jegliche Hardwarekomponenten, welche an MP befestigt werden, sollen in ihren Dimensionen und Befestigungsmöglichkeiten an die jeweiligen MP-Typen angepasst sein. Z. B.:
    - Kabelbinder für Verwendung an Sensorkabeln
    - Längliche, schmale Form für Verwendung an Sensorkabeln
    - Klebepads für Verwendung an flachen Gehäusen
    - Magneten für Verwendung an Gehäusen mit ferromagnetischen Eigenschaften
    - Usw.

3.2. Anforderungen an die Sicherheit der verwendeten Komponenten für den Gebrauch im medizinischen Umfeld

1. Die in den medizinischen Einsatzbereichen verwendeten Hardwarekomponenten des ATS sollen keine Auswirkungen auf die Funktionsfähigkeit umliegender MP haben. (z.B. EMV beachten)
2. Die in den medizinischen Einsatzbereichen verwendeten Hardwarekomponenten sollen keine Gefahr für umliegende Geräte oder Personen darstellen (z.B. durch Entflammbarkeit)

A4

Pflichtenheft
Asset Tracking in Krankenhäusern und Kliniken

Projekt im Rahmen der Bachelorarbeit.

Inhaltsverzeichnis

1. Funktionale Anforderungen

1.1. Architektur, Datenerhebung und Datenweiteleitung

1.1.1. Architekturübersicht

In Abb. 1 ist eine Übersicht der gesamten Systemarchitektur zu sehen. Diese wird durch nachfolgende Anforderungen definiert und in späteren Anforderungen spezifiziert.

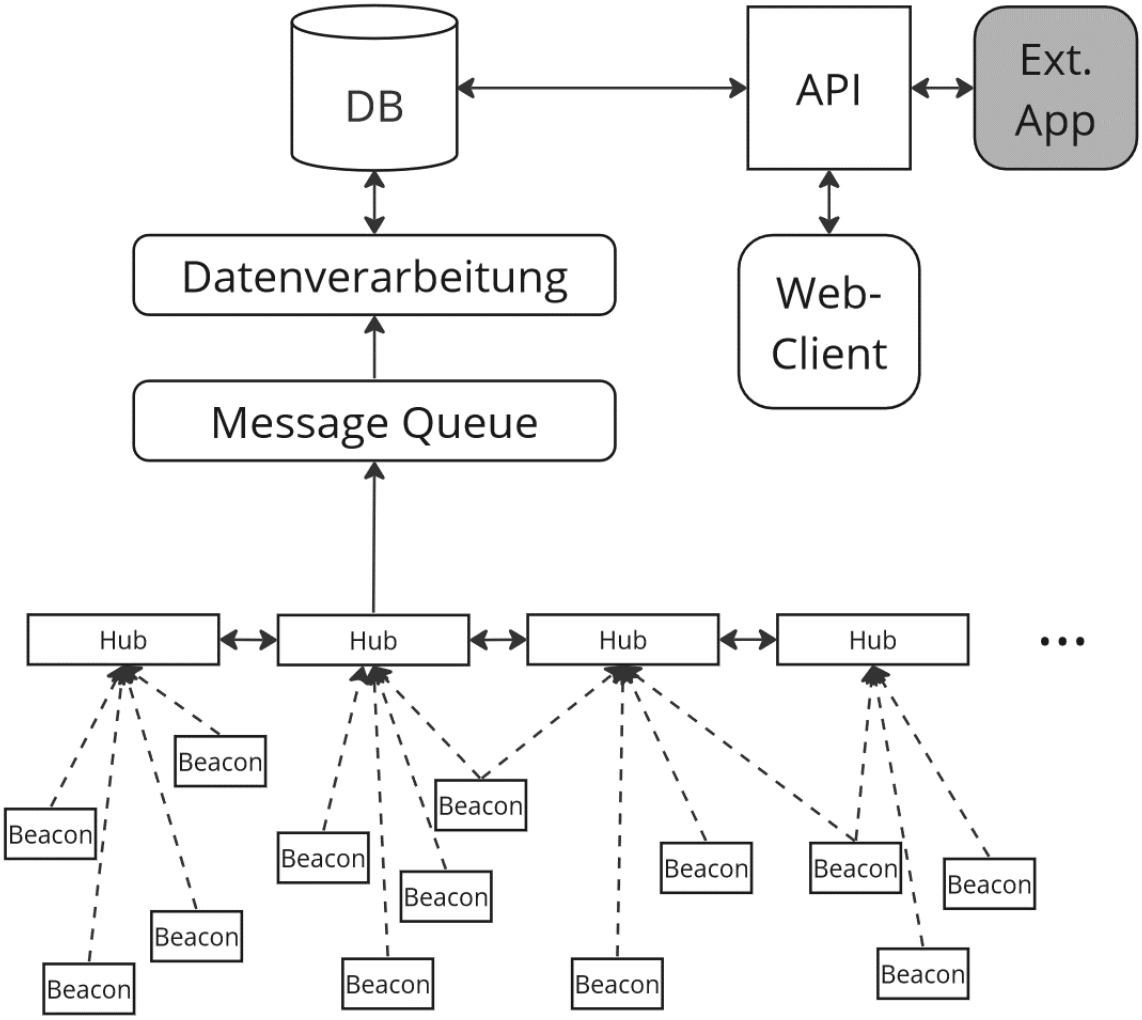


Abbildung 1: Architekturübersicht

1. Die Komponenten des ATS, die die Ortung ermöglichen, sollen aus einem BLE fähigen Sender (folgend Beacon) und einem BLE fähigen Empfänger (folgend Hub) bestehen.
2. Die Beacons sollen in einem Intervall von t ≤ 60 Sekunden an alle umliegenden BT/BLE fähigen Geräte senden

3. Die Hubs sollen permanent nach umliegenden Beacons suchen und deren Daten empfangen.
4. Die Hubs sollen ein Wifi-Mesh untereinander aufbauen.
5. Auf einem dedizierten Server soll permanent ein MQTT-Broker laufen.
6. Die Hubs sollen alle relevanten Beacon-Daten an den MQTT-Broker senden.
7. Auf einem dedizierten Server soll ein verarbeitender Microservice laufen.
8. Der Microservice soll die Daten aus der MQ abrufen und entsprechend ihrer Zweckbestimmung verarbeiten.
9. Es soll eine zentrale Datenbank (DB) geben, auf die von mehreren Clients aus gleichzeitig zugegriffen werden kann.
10. Die DB soll für jede betrachtete Soft- und Hardwarekomponente die relevanten Daten speichern.
11. Der Microservice soll die verarbeiteten Daten in der DB speichern oder an dafür vorgesehene Stored Procedures übergeben.
12. Die Datenhaltung und Interaktion mit den Daten des ATS soll mit einem Client-Server System umgesetzt werden.
13. Alle notwenigen DB-Anfragen, welche für die Interaktion mit den Webansichten benötigt werden, sollen mithilfe einer API umgesetzt werden (backend).

1.1.1.1. Beacons

1. Die Beacons sollen über einen Knopf verfügen, der sie unabhängig von dem festgelegten Sendeintervall sofort senden lässt.
2. Die Beacons sollen den iBeacon Standard unterstützen (impliziert einen UUID).
3. Die Beacons sollen eine eindeutige Mac-Adresse haben
4. Die Beacons sollen austauschbare Standardkomponenten sein, sodass sie leicht durch einen anderen Typ ersetzt werden könnten.

1.1.1.2. Hubs

1. Die Hubs sollen dazu in der Lage sein lückenlos nach Bluetoothgeräten zu suchen, während das WiFi-Mesh aufrecht erhalten wird.
2. Die Hubs sollen sich mit einem am Einsatzort verfügbaren Access-Point verbinden können.
3. Die für die Hubs verwendete Hardware soll aus austauschbaren Standardkomponenten bestehen, sodass sie leicht durch einen anderen Typ ersetzt werden könnten.
4. Die Hubs sollen eine eindeutige MAC-Adresse haben.
5. Das WiFi-Mesh der Hubs soll eine Tiefe von mindestens 4 Knoten bei mindestens 5 Child-Nodes aufbauen können.

1.1.2. Datenerhebung

1. Die Beacons sollen beim Senden folgende Daten an umliegende Hubs übermitteln:
   - UUID
   - MAC-Adresse
   - Buttonstatus
   - Batteriestatus
2. Die Hubs sollen permanent nach BLE-Geräten suchen.
3. Die Hubs sollen bei Detektion eines BLE-Geräts mit iBeacon Signatur die von dem Gerät gesendeten Daten empfangen.
4. Die Hubs sollen die Empfangsstärke (RSSI) zu den jeweiligen Beacons erfassen.

1.1.3. Datenweiterleitung

1. Bei der notwendigen Weiterleitung von Daten innerhalb lokal verfügbarer Netzwerke der Einrichtung, darf der Traffic die Funktion jeglicher mit dem Netzwerk verbundener Gerät nicht beeinflussen. (So wenig wie möglich, so viel wie nötig)
2. Die Anzahl der mit den lokal verfügbaren WLAN verbundenen Hardwarekomponenten soll in jedem Bereich (z.B. Krankenhausstation) auf max. 2 begrenzt werden
3. Die Hubs sollen eine Verbindung zum MQTT-Broker aufbauen können.
4. Die Daten, welche die Hubs empfangen, sollen chronologisch unmittelbar nach dem Verarbeiten gesendet werden.
5. In den Daten, welche die Hubs senden, soll die jeweilig festgestellte RSSI der Beacons enthalten sein.
6. Die Daten, welche die Hubs senden, sollen als kommaseparierte Liste mit Name-Value Paaren an den MQTT-Broker gesendet werden.
7. Der MQTT-Broker soll jeder empfangenen Nachricht die Empfangszeit als Unix-Zeitstempel hinzufügen.
8. Der MQTT-Broker soll die um den Timestamp erweiterten Nachrichten im JSON-Dateiformat bereitstellen.
9. Der MQTT-Broker soll die Nachrichten nur einmal an ein bekanntes Gerät senden. (für Exception Handling)

1.2. Datenhaltung

1. Auf einem dedizierten Server soll ein DB -Server laufen.
2. In der DB soll eine relationale Datenstruktur zur Abbildung folgender Daten umgesetzt werden:

- Beacons
- Hubs
- MP
- Räume
- Bereiche
- MP zu MP Zuordnungen
- Zulässigen MP-Kombinationen
- MP-Ortungs-Historien
- Raum-Historien
- Bereich-Historien
- MP-Zuordnungs-Historien

3. Die folgenden Datenänderungen sollen für mindestens 24 Monate gespeichert werden und nachträglich einsehbar sein:
   - Standortänderung MP
   - Änderung MP-Kombination
   - MP-Inaktivierung
   - Raum-Inaktivierung
   - Bereichs-Inaktivierung
4. Die Historie dieser Datensätze soll den Anfangs- und Endzeitpunkt des Vorgangs enthalten. (z.B. MP vom 01.01.2001 bis 12.12.2012 aktiv gewesen)

1.3. Datenverarbeitung

1. Die aus der MQ ausgelesenen Daten sollen so verarbeitet werden, dass die Verarbeitungsergebnisse zunächst in einem lokalen Zwischenspeicher (folgend MemCache) gespeichert werden.
2. Der MemCache soll zur unterbrechungsfreien Verarbeitung zwischengespeicherte Daten auch im Falle eines Neustarts des Microservice halten.
3. Bei der Verarbeitung neu empfangener Daten sollen diese zunächst mit den im MemCache befindlichen Daten verglichen werden.
4. Eine DB-Anfrage soll erst dann gestellt/ abgesetzt werden, wenn eine relevante Datenänderung, wie ein Hubwechsel oder eine Erstzuweisung, festgestellt wurde.
5. Die Zuweisung eines Beacons zu einem Hub/Raum soll anhand des Vergleichs der Empfangsfeldstärke an unterschiedlichen Hubs erfolgen (Höchste RSSI gewinnt).

6. Wenn der Knopf eines Beacons betätigt wurde, soll mithilfe des Microservices und Stored Procedures geprüft werden, ob eine Zuordnung zu einem (weiteren) Beacon nötig ist (z.B. Zubehör zu Gerät).
7. Die Beacon zu Beacon Zuordnung soll nur dann stattfinden, wenn sich die Beacons eindeutig an demselben Ort befinden.
8. Die Beacon zu Beacon Zuordnung soll nur dann stattfinden, wenn beide Knöpfe innerhalb eines Zeitintervalls von t ≤ 60 Sekunden betätigt wurden.
9. Die Beacon zu Beacon Zuordnung soll nur dann stattfinden, wenn die zugeordneten MP in der entsprechenden DB-Tabelle für eine zulässige kombinatorische Nutzung eingetragen sind.
10. Die Beacon zu Beacon Zuordnung soll aufgelöst werden, wenn die Beacons 2-mal hintereinander (2 Scanintervalle) in anderen Räumen detektiert wurden.
11. Es sollen möglichst alle Ausnahmefälle erörtert und in der Positionsbestimmung bedacht werden (Raumwechsel für kurze Zeit, Öffnen und Schließen von Türen etc.)

In den Abb. 2 und 3 ist die geplante Beacondaten Verarbeitung zur besseren Verständlichkeit als Flowcharts dargestellt.

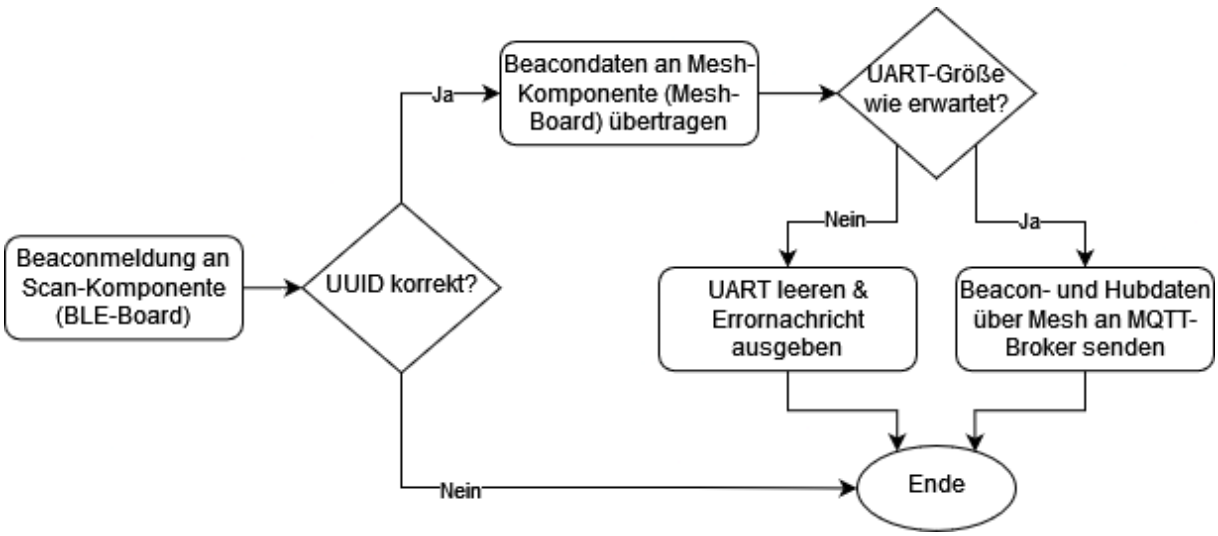


Abbildung 2: Beacon -> Hub -> MQTT – Nachrichtenverlauf

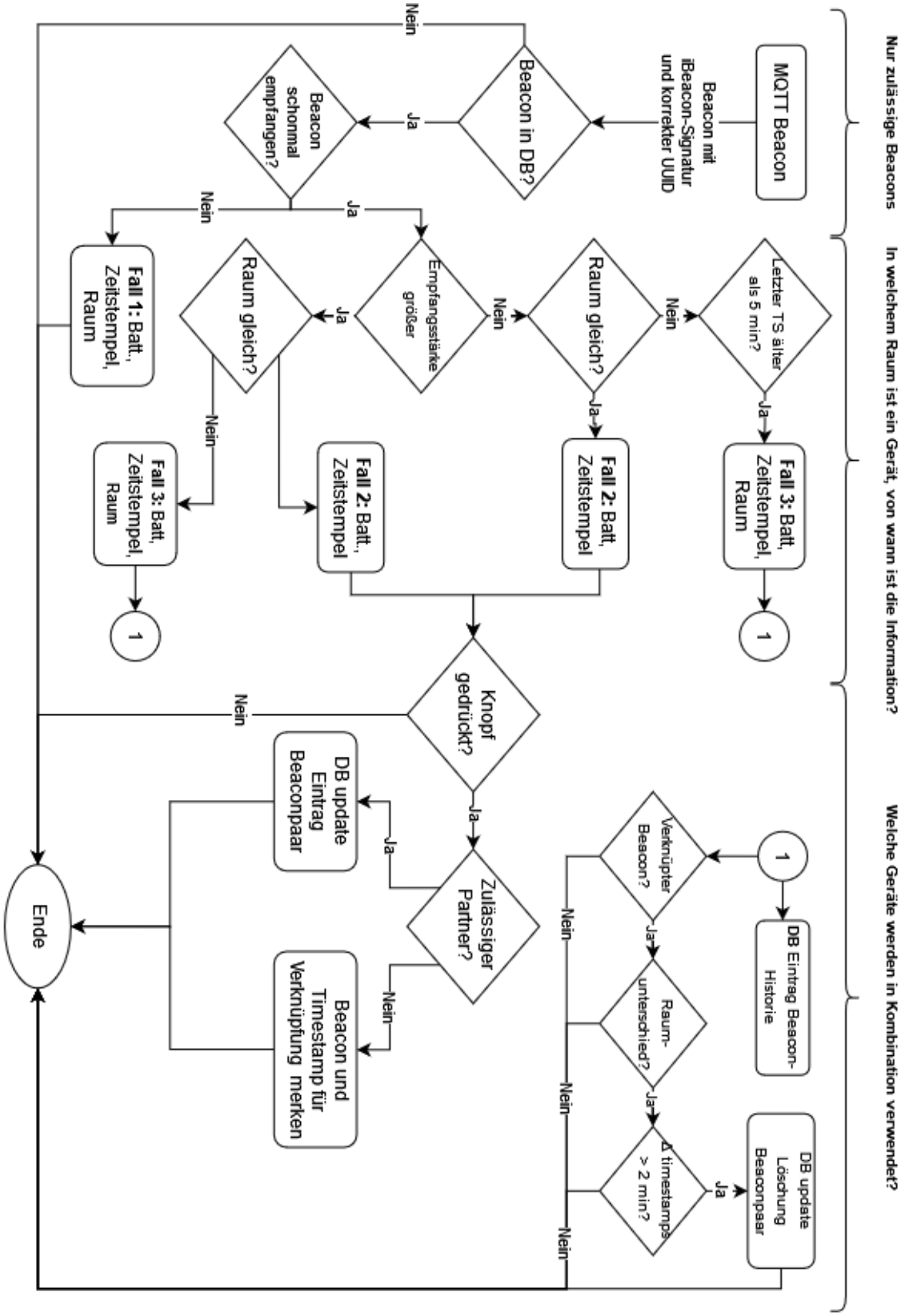


Abbildung 3: Ablauf Beacondatenverarbeitung

1.4. Webanwendung und API

1. Die Webanwendung soll mehrere Ansichten bereitstellen, welche folgende Inhalte darstellen sollen (Frontend):
   - „MP-Suchansicht“: Suche nach MP, anschließendes Anzeigen des aktuellen Standorts.
   - „Raum- / Bereich-Suchansicht“: Suche nach Raum/Bereich, Ansicht aller MP an diesem Ort als Tabelle.
   - „MP-Serviceansicht“: Tabelle mit allen MP, mit Möglichkeit zur Anpassung von Datensätzen.
   - „Raum-Serviceansicht“: Tabelle mit allen Räumen, mit Möglichkeit zur Anpassung von Datensätzen.
   - „Bereich-Serviceansicht“: Tabelle mit allen Bereichen, mit Möglichkeit zur Anpassung von Datensätzen.
   - „MP-Kombination-Serviceansicht“: Tabelle mit allen zulässigen MP-Kombinationen, mit Möglichkeit zur Anpassung von Datensätzen.
   - „Hardware-Serviceansicht“: Ansicht zur Pflege und Einrichtung aller für die Ortung benötigten Hardwarekomponenten (z.B. Sender, Empfänger o.Ä.)
2. Alle in den Webanwendungen gezeigten Daten/Ansichten sollen mithilfe von asynchronen Elementen umgesetzt werden (z.B. über node.js/vue.js)
3. Alle Ansichten der Webanwendungen sollen sich an modernen UI-Designs orientieren, um die Nutzung zu erleichtern.
4. Die API soll alle notwendigen Funktionen zur Dateninteraktion definieren (mit: GET, POST, PUT, DELETE), welche von dem Client aufgerufen werden können sollen.

2. Nichtfunktionale Anforderungen

2.1. Anforderungen an Eigenschaften der Verwendeten Materialien und den Formfaktor der zur Umsetzung benötigten Komponenten

1. Die in den medizinischen Einsatzbereichen Verwendeten Hardwarekomponenten sollen gegen Umwelteinflüsse geschützt sein, wie sie in dem Einsatzbereich auftreten könnten:
   - Staub

- Feuchtigkeit
- UV-Strahlung
- Schlag- und Stoßbeanspruchung
- Desinfektion mit gängigen Mitteln

2. Die Hubs sollen über ein USB-C Netzteil mit Strom versorgt werden.
3. Die Beacons sollen über eine Batterie mit Strom versorgt werden.
4. Die Betriebsdauer aktiver Komponenten zur Positionserfassung soll ohne Wartung oder Batterietausch minimal 2 Jahre betragen.
5. Ein Batterietausche oder ein Laden von Komponenten zur Positionserfassung soll schnell, einfach und mit Standardkomponenten (z.B. standardisierten Batteriezellen) möglich sein.
6. Im Falles einer Wartung von Komponenten zur Positionserfassung sollen die Schutzklasse (z.B. IP67) sowie alle o.g. Eigenschaften weiterhin erhalten bleiben.
7. Jegliche Hardwarekomponenten, welche an MP befestigt werden, sollen in ihren Dimensionen und Befestigungsmöglichkeiten an die jeweiligen MP-Typen angepasst sein. Z. B.:
   - Kabelbinder für Verwendung an Sensorkabeln
   - Längliche, schmale Form für Verwendung an Sensorkabeln
   - Klebepads für Verwendung an flachen Gehäusen
   - Magneten für Verwendung an Gehäusen mit ferromagnetischen Eigenschaften
   - Usw.

2.2. Anforderungen an die Sicherheit der verwendeten Komponenten für den Gebrauch im medizinischen Umfeld

1. Die in den medizinischen Einsatzbereichen verwendeten Hardwarekomponenten des ATS sollen keine Auswirkungen auf die Funktionsfähigkeit umliegender MP haben. (z.B. EMV beachten)
2. Die in den medizinischen Einsatzbereichen verwendeten Hardwarekomponenten sollen keine Gefahr für umliegende Geräte oder Personen darstellen (z.B. durch Entflammbarkeit)

A5 – MDR-Auszug

(Geltungsbereich der Verordnung (EU) 2017/745 über Medizinprodukte (MDR) vom 5. April 2017 inklusive Corrigendum vom 13.03.2019, 25.11.2019, 23.04.2020 und 08.07.2021)

[1]

Artikel 1: Gegenstand und Geltungsbereich (Auszug)

(1) Mit dieser Verordnung werden Regeln für das Inverkehrbringen, die Bereitstellung auf dem Markt und die Inbetriebnahme von für den menschlichen Gebrauch bestimmten Medizinprodukten und deren Zubehör in der Union festgelegt. Diese Verordnung gilt ferner für in der Union durchgeführte klinische Prüfungen, die diese Medizinprodukte und dieses Zubehör betreffen.

(2) Diese Verordnung gilt des Weiteren ab dem Geltungsbeginn der GS gemäß Artikel 9 für die in Anhang XVI aufgeführten Produktgruppen ohne medizinische Zweckbestimmung, wobei der Stand der Technik und insbesondere bereits geltende harmonisierte Normen für analoge Produkte mit medizinischer Zweckbestimmung, die auf ähnlicher Technologie beruhen, zu berücksichtigen sind. Gegenstand der GS für jede in Anhang XVI aufgeführte Produktgruppe sind mindestens die Anwendung des Risikomanagements gemäß Anhang I für die betreffende Produktgruppe und erforderlichenfalls die klinische Bewertung der Sicherheit.

….

(3) Produkte mit medizinischer und nicht-medizinischer Zweckbestimmung müssen sowohl die Anforderungen an Produkte mit medizinischer Zweckbestimmung als auch die Anforderungen an Produkte ohne medizinische Zweckbestimmung erfüllen.

(4) Für die Zwecke dieser Verordnung werden Medizinprodukte und ihr Zubehör sowie die in Anhang XVI aufgeführten Produkte, auf die diese Verordnung gemäß Absatz 2 Anwendung findet, im Folgenden als „Produkte“ bezeichnet.

(5) In Fällen, in denen dies aufgrund der Ähnlichkeit eines in Verkehr gebrachten Produkts mit medizinischer Zweckbestimmung und eines Produkts ohne medizinische Zweckbestimmung in Bezug auf ihre Merkmale und die damit verbundenen Risiken gerechtfertigt ist, wird der Kommission die Befugnis übertragen, gemäß Artikel 115 delegierte Rechtsakte zu erlassen, um die in Anhang XVI enthaltene Liste von Produkten durch Hinzufügung neuer Produktgruppen anzupassen, um den Schutz der Gesundheit und Sicherheit der Anwender oder anderer Personen oder anderer Aspekte der öffentlichen Gesundheit zu gewährleisten.

Artikel 2: Begriffsbestimmungen (Auszug)

1. „Medizinprodukt“ bezeichnet ein Instrument, einen Apparat, ein Gerät, eine Software, ein Implantat, ein Reagenz, ein Material oder einen anderen Gegenstand, das dem Hersteller zufolge für Menschen bestimmt ist und allein oder in Kombination einen oder mehrere der folgenden spezifischen medizinischen Zwecke erfüllen soll:

- Diagnose, Verhütung, Überwachung, Vorhersage, Prognose, Behandlung oder Linderung von Krankheiten,
- Diagnose, Überwachung, Behandlung, Linderung von oder Kompensierung von Verletzungen oder Behinderungen,
- Untersuchung, Ersatz oder Veränderung der Anatomie oder eines physiologischen oder pathologischen Vorgangs oder Zustands,
- Gewinnung von Informationen durch die In-vitro-Untersuchung von aus dem menschlichen Körper — auch aus Organ-, Blut- und Gewebespenden — stammenden Proben

und dessen bestimmungsgemäße Hauptwirkung im oder am menschlichen Körper weder durch pharmakologische oder immunologische Mittel noch metabolisch erreicht wird, dessen Wirkungsweise aber durch solche Mittel unterstützt werden kann.

Die folgenden Produkte gelten ebenfalls als Medizinprodukte:

- Produkte zur Empfängnisverhütung oder -förderung,
- Produkte, die speziell für die Reinigung, Desinfektion oder Sterilisation der in Artikel 1 Absatz 4 genannten Produkte und der in Absatz 1 dieses Spiegelstrichs genannten Produkte bestimmt sind.

2. „Zubehör eines Medizinprodukts“ bezeichnet einen Gegenstand, der zwar an sich kein Medizinprodukt ist, aber vom Hersteller dazu bestimmt ist, zusammen mit einem oder mehreren bestimmten Medizinprodukten verwendet zu werden, und der speziell dessen/deren Verwendung gemäß seiner/ihrer Zweckbestimmung(en) ermöglicht oder mit dem die medizinische Funktion des Medizinprodukts bzw. der Medizinprodukte im Hinblick auf dessen/deren Zweckbestimmung(en) gezielt und unmittelbar unterstützt werden soll;

10. „Behandlungseinheit“ bezeichnet eine Kombination von zusammen verpackten und in Verkehr gebrachten Produkten, die zur Verwendung für einen spezifischen medizinischen Zweck bestimmt sind;

11. „System“ bezeichnet eine Kombination von Produkten, die zusammen verpackt sind oder auch nicht und die dazu bestimmt sind, verbunden oder kombiniert zu werden, um einen spezifischen medizinischen Zweck zu erfüllen;

A6 – scan.c

```c
/*
    Projektarbeit Linus Norden 2024
    Adaption der Espressif iBeacon Demo
*/

#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include <stdio.h>
#include "nvs_flash.h"

#include "esp_bt.h"
#include "esp_gap_ble_api.h"
#include "esp_gattc_api.h"
#include "esp_gatt_defs.h"
#include "esp_bt_main.h"
#include "esp_bt_defs.h"
#include "esp_ibeacon_api.h"
#include "esp_log.h"
#include "freertos/FreeRTOS.h"

// Lower Level access für UART
#include "freertos/task.h"
#include "driver/uart.h"
#include "driver/gpio.h"
#include "sdkconfig.h"

// Serielle Schnittstellen definieren
#define BOARD2BOARD_TXD 3
#define BOARD2BOARD_RXD 4
#define BOARD2BOARD_RTS (UART_PIN_NO_CHANGE) // -1, ungenutzt
#define BOARD2BOARD_CTS (UART_PIN_NO_CHANGE) // -1, ungenutzt

#define BOARD2BOARD_PORT_NUM 1 //esp32c3 bietet Ports 0-2 an
#define BOARD2BOARD_BAUD_RATE 115200
#define BOARD2BOARD_STACK_SIZE 2048 // UART-Stack Größe, mehr siehe:
                                    https://docs.espressif.com/projects/esp-
                                    idf/en/latest/esp32/api-reference/peripherals/uart.html
#define BOARD2BOARD_BUF_SIZE 1024 // Zwischenspeicher für Nachrichten

uart_config_t uart_config = {
    .baud_rate = BOARD2BOARD_BAUD_RATE,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_DISABLE,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    .rx_flow_ctrl_thresh = 122,
    .source_clk = UART_SCLK_DEFAULT
};
int intr_alloc_flags = 0;
#if CONFIG_UART_ISR_IN_IRAM
    intr_alloc_flags = ESP_INTR_FLAG_IRAM;
#endif

static const char* APP_TAG = "SENSOR_SCANNER";
// Das hier konfigurierte Scan-Board kann anhand der eingestellten UUID explizit
 ausschließlich nach Geräten suchen, die dem eigenen Pool entstammen
static const uint8_t OWNED_UUID[] =
                                    {0x61,0x6e,0x50,0xac,0x4a,0xfa,0x4e,0x41,0xba,0x2a,0x8c,
                                    0x82,0xf3,0xe6,0x56,0xca};
extern esp_ble_ibeacon_vendor_t vendor_config;

// Statische Funktionen deklarieren
static void esp_gap_cb(esp_gap_ble_cb_event_t event, esp_ble_gap_cb_param_t *param);
// BLE Scan Einstellungen
static esp_ble_scan_params_t ble_scan_params = {
    .scan_type              = BLE_SCAN_TYPE_ACTIVE,
    .own_addr_type          = BLE_ADDR_TYPE_PUBLIC,
```

```
64.     .scan_filter_policy     = BLE_SCAN_FILTER_ALLOW_ALL,
65.     .scan_interval          = 0x50,
66.     .scan_window            = 0x30,
67.     .scan_duplicate         = BLE_SCAN_DUPLICATE_DISABLE
68. };
69.
70. // BT event handler
71. static void esp_gap_cb(esp_gap_ble_cb_event_t event, esp_ble_gap_cb_param_t *param)
72. {
73.     esp_err_t err;
74.     switch (event) {
75.     case ESP_GAP_BLE_SCAN_PARAM_SET_COMPLETE_EVT: {
76.         // Einheit sind Sekunden, 0 -> permanent scannen
77.         uint32_t duration = 0;
78.         esp_ble_gap_start_scanning(duration);
79.         break;
80.     }
81.     case ESP_GAP_BLE_SCAN_START_COMPLETE_EVT:
82.         // Indikator für Scan-Start erfolgreich oder fehlerhaft
83.         if ((err = param->scan_start_cmpl.status) != ESP_BT_STATUS_SUCCESS) {
84.             ESP_LOGE(APP_TAG, "Scan start failed: %s", esp_err_to_name(err));
85.         }
86.         break;
87.     case ESP_GAP_BLE_SCAN_RESULT_EVT: {
88.         /* Wenn es BLE-Scanergebnisse gibt, müssen einige Werte aus dem advertising data
             Paket geholt werden.
89.            - die Mac-Adresse als eindeutige Kennung
90.            - Buttonstatus, falls vorhanden
91.            - Batteriestand in Prozent, falls angegeben
92.            - RSSI als Entfernungsindikator
93.         */
94.         esp_ble_gap_cb_param_t *scan_result = (esp_ble_gap_cb_param_t *)param;
95.         switch (scan_result->scan_rst.search_evt) {
96.             case ESP_GAP_SEARCH_INQ_RES_EVT:
97.                 // dieses Ereignis wird ausgelöst, wenn z.B. zusätzliche Werbedaten gefunden
                      wurden
98.                 // Suche nach BLE iBeacon-Paket
99.                 if (esp_ble_is_ibeacon_packet(scan_result->scan_rst.ble_adv, scan_result-
                                                  >scan_rst.adv_data_len)){
100.                    esp_ble_ibeacon_t *ibeacon_data = (esp_ble_ibeacon_t*)(scan_result-
                                                          >scan_rst.ble_adv);
101.                    // Für Debugging: Namen lesen
102. //                 uint8_t *adv_name = NULL;
103. //                 uint8_t adv_name_len = 0;
104.                    uint8_t *adv_service_data = NULL;
105.                    uint8_t adv_service_data_len = 0;
106. //                   adv_name = esp_ble_resolve_adv_data(param->scan_rst.ble_adv,
                                    ESP_BLE_AD_TYPE_NAME_CMPL, &adv_name_len);
107. //                 if (adv_name ) ESP_LOGI(APP_TAG,"adv_name: %s", adv_name );
108.                    /* Wenn die UUID übereinstimmt, wurde ein zugehöriger Sensor
                        identifiziert. Gegebene Parameter für MAC-Adresse, Tastenzustand,
109.                    RSSI und Batteriestand müssen über SoftwareSerial an das WLAN-Board
                        übertragen werden.
110.                    */
111.                    if ( memcmp(OWNED_UUID, ibeacon_data->ibeacon_vendor.proximity_uuid,
ESP_UUID_LEN_128) == 0){
112.                        uint8_t serial_data[9]; // MAC + Batt + Button + RSSI
113.                        // Hinzufügen der MAC-Adresse zu den seriellen Daten - Schritt 1:
                            Kopieren der uint_8-Daten in den char- Buffer, Schritt 2: Verkettung
114.                        adv_service_data = esp_ble_resolve_adv_data(param->scan_rst.ble_adv,
ESP_BLE_AD_TYPE_SERVICE_DATA, &adv_service_data_len);
115.                        memcpy(serial_data, scan_result->scan_rst.bda, 6);
116.                        if (adv_service_data ) {
117. //                         ESP_LOGI(APP_TAG, "Battery: %d %%", adv_service_data[3]);
118.                            serial_data[6] = adv_service_data[3];
119. //                         ESP_LOGI(APP_TAG, "Button pressed: %d", adv_service_data[13]);
120.                            serial_data[7] = adv_service_data[13];
121. //                         ESP_LOGI(APP_TAG, "RSSI of packet:%d dbm", scan_result-
                                          >scan_rst.rssi);
122.                            serial_data[8] = scan_result->scan_rst.rssi;
123.                            esp_log_buffer_hex("Write data: ", serial_data, 9 );
```

A6 - scan.c

```
124.
125.                             // Daten schreiben und auf das Senden des Pakets warten
126.                             // Wegen möglichen Problemen mit dem Timing könnte dies
                                eventuell besser mit einer asynchronen Methode übertragen werden.
127.                             uart_write_bytes(BOARD2BOARD_PORT_NUM, (const char
                                                 *)serial_data, 9);
128.                             ESP_ERROR_CHECK(uart_wait_tx_done(BOARD2BOARD_PORT_NUM, 100));
                                 // Timeout sind 100 RTOS ticks (TickType_t)
129.                         }
130.                         // Im 'else'-Fall fehlen advertising-Daten
131.                         // Es braucht einen temporären Buffer für eingehende Daten
132.                         // uint8_t *data_in = (uint8_t *) malloc(BOARD2BOARD_BUF_SIZE);
133.                         // Daten aus dem UART lesen
134.                         // int len = uart_read_bytes(BOARD2BOARD_PORT_NUM, data_in,
                            (BOARD2BOARD_BUF_SIZE - 1), 20 / portTICK_PERIOD_MS);
135.                         // if (len) {
136.                         //    esp_log_buffer_hex("IBEACON_DEMO: read data:", data_in, len);
137.                         // }
138.                     }
139.                 }
140.                 break;
141.             default:
142.                 ESP_LOGI(APP_TAG, "Search event type: %d", scan_result->scan_rst.search_evt);
143.                 break;
144.         }
145.         break;
146.     }
147.     case ESP_GAP_BLE_SCAN_STOP_COMPLETE_EVT:
148.         if ((err = param->scan_stop_cmpl.status) != ESP_BT_STATUS_SUCCESS){
149.             ESP_LOGE(APP_TAG, "Scan stop failed: %s", esp_err_to_name(err));
150.         }
151.         else {
152.             ESP_LOGI(APP_TAG, "Stop scan successfully");
153.         }
154.         break;
155.
156.     default:
157.         break;
158.     }
159. }
160.
161. void ble_ibeacon_appRegister(void)
162. {
163.     esp_err_t status;
164.     ESP_LOGI(APP_TAG, "register callback");
165.     // Registriere callback-Funktion zum gap-Modul
166.     if ((status = esp_ble_gap_register_callback(esp_gap_cb)) != ESP_OK) {
167.         ESP_LOGE(APP_TAG, "gap register error: %s", esp_err_to_name(status));
168.         return;
169.     }
170. }
171.
172. void ble_ibeacon_init(void)
173. {
174.     esp_bluedroid_init();
175.     esp_bluedroid_enable();
176.     ble_ibeacon_appRegister();
177. }
178.
179. void app_main(void)
180. {
181.     ESP_ERROR_CHECK(nvs_flash_init());
182.     ESP_ERROR_CHECK(esp_bt_controller_mem_release(ESP_BT_MODE_CLASSIC_BT));
183.     esp_bt_controller_config_t bt_cfg = BT_CONTROLLER_INIT_CONFIG_DEFAULT();
184.     esp_bt_controller_init(&bt_cfg);
185.     esp_bt_controller_enable(ESP_BT_MODE_BLE);
186.
187.     // UART Treiber installieren
188.     ESP_ERROR_CHECK( uart_driver_install(BOARD2BOARD_PORT_NUM, BOARD2BOARD_BUF_SIZE *2,0, 0,
                                             NULL, intr_alloc_flags) ); // no event queue
                                             defined (NULL)
```

A6 - scan.c

```
189.     ESP_ERROR_CHECK(uart_param_config(BOARD2BOARD_PORT_NUM, &uart_config));
190.     ESP_ERROR_CHECK(uart_set_pin(BOARD2BOARD_PORT_NUM, BOARD2BOARD_TXD, BOARD2BOARD_RXD,
                                     BOARD2BOARD_RTS, BOARD2BOARD_CTS));
191.
192.     ble_ibeacon_init();
193.     /* Scan Parameter setzen */
194.     esp_ble_gap_set_scan_params(&ble_scan_params);
195.
196. }
197.
198.
```

A7 – mesh_main.c

```c
/*
    Projektarbeit Linus Norden 2024
    Adaption des Espressif Mesh Internal Communication Example
*/
#include <string.h>
#include "esp_wifi.h"
#include "esp_mac.h"
#include "esp_event.h"
#include "esp_log.h"
#include "esp_mesh.h"
#include "nvs_flash.h"
#include "mesh_netif.h"
#include "driver/gpio.h"
#include "freertos/semphr.h"

// Lower Level access für UART
#include "freertos/task.h"
#include "driver/uart.h"
#include "driver/gpio.h"

// Serielle Schnittstellen definieren
#define BOARD2BOARD_TXD 4
#define BOARD2BOARD_RXD 3
#define BOARD2BOARD_RTS (UART_PIN_NO_CHANGE) // -1, ungenutzt
#define BOARD2BOARD_CTS (UART_PIN_NO_CHANGE) // -1, ungenutzt

#define BOARD2BOARD_PORT_NUM 1 //esp32c3 bietet Ports 0-2 an
#define BOARD2BOARD_BAUD_RATE 115200
#define BOARD2BOARD_STACK_SIZE 2048 // UART-Stack Größe, mehr siehe:
                                    https://docs.espressif.com/projects/esp-
                                    idf/en/latest/esp32/api-reference/peripherals/uart.html
#define BOARD2BOARD_BUF_SIZE (1024*2) // Zwischenspeicher für Nachrichten

uart_config_t uart_config = {
    .baud_rate = BOARD2BOARD_BAUD_RATE,
    .data_bits = UART_DATA_8_BITS,
    .parity = UART_PARITY_DISABLE,
    .stop_bits = UART_STOP_BITS_1,
    .flow_ctrl = UART_HW_FLOWCTRL_CTS_RTS,
    .rx_flow_ctrl_thresh = 122,
    .source_clk = UART_SCLK_DEFAULT
};

/*******************************************************
 *                Macros
 *******************************************************/
// commands for internal mesh communication:
// <CMD> <PAYLOAD>, where CMD is one character, payload is variable dep. on command
#define CMD_ROUTE_TABLE 0x56

/*******************************************************
 *                Constants
 *******************************************************/
// Fürs Logging
static const char *MESH_TAG = "ROOM_MONITOR";
// ID des WLAN-Meshnetzes, das die Boards bilden
static const uint8_t MESH_ID[6] = { 0x77, 0x77, 0x77, 0x77, 0x77, 0x76};

/*******************************************************
 *                Variable Definitions
 *                Mesh Parameter und serieller-Payload Check
 *******************************************************/
static bool is_running = true;
static mesh_addr_t mesh_parent_addr;
static int mesh_layer = -1;
static esp_ip4_addr_t s_current_ip;
static mesh_addr_t s_route_table[CONFIG_MESH_ROUTE_TABLE_SIZE];
static int s_route_table_size = 0;
```

```c
static SemaphoreHandle_t s_route_table_lock = NULL;
static uint8_t s_mesh_tx_payload[CONFIG_MESH_ROUTE_TABLE_SIZE*6+1];
static int is_broken_package = false; // wird auf true gesetzt, wenn UART weniger Zeichen
                                      als benötigt empfangen hat

/*******************************************************
 *                Function Declarations
 *******************************************************/
// interaction with public mqtt broker
void mqtt_app_start(void);
void mqtt_app_publish(char* topic, char *publish_string);

/*******************************************************
 *                Function Definitions
 *******************************************************/

// WLAN-Empfänger Daten-Handling
void static recv_cb(mesh_addr_t *from, mesh_data_t *data)
{
    if (data->data[0] == CMD_ROUTE_TABLE) {
        int size =  data->size - 1;
        if (s_route_table_lock == NULL || size%6 != 0) {
            ESP_LOGE(MESH_TAG, "Error in receiving raw mesh data: Unexpected size");
            return;
        }
        xSemaphoreTake(s_route_table_lock, portMAX_DELAY);
        s_route_table_size = size / 6;
        for (int i=0; i < s_route_table_size; ++i) {
            ESP_LOGI(MESH_TAG, "Received Routing table [%d] "
                    MACSTR, i, MAC2STR(data->data + 6*i + 1));
        }
        memcpy(&s_route_table, data->data + 1, size);
        xSemaphoreGive(s_route_table_lock);
    } else {
        ESP_LOGE(MESH_TAG, "Error in receiving raw mesh data: Unknown command");
    }
}

// mqtt-Hauptfunktion, Empfang serieller Daten vom BLE-Board, Verarbeitung der Daten
void esp_mesh_mqtt_task(void *arg)
{
    is_running = true; // lock process
    char *print;
    mesh_data_t data;
    esp_err_t err;
    uint8_t mac[6] = {0};
    err = esp_efuse_mac_get_default(mac);
    mqtt_app_start();
    while (is_running) {
        if (esp_mesh_is_root()) {
            esp_mesh_get_routing_table((mesh_addr_t *) &s_route_table,
                                       CONFIG_MESH_ROUTE_TABLE_SIZE * 6,
                                       &s_route_table_size);
            data.size = s_route_table_size * 6 + 1;
            data.proto = MESH_PROTO_BIN;
            data.tos = MESH_TOS_P2P;
            s_mesh_tx_payload[0] = CMD_ROUTE_TABLE;
            memcpy(s_mesh_tx_payload + 1, s_route_table, s_route_table_size*6);
            data.data = s_mesh_tx_payload;
            for (int i = 0; i < s_route_table_size; i++) {
                err = esp_mesh_send(&s_route_table[i], &data, MESH_DATA_P2P, NULL, 0);
                ESP_LOGI(MESH_TAG, "Sending routing table to [%d] "
                        MACSTR ": sent with err code: %d", i,
                        MAC2STR(s_route_table[i].addr), err);
            }
        }

        uint8_t serial_data[9]; // MAC + Batt + Button + RSSI
        int length = 0;
        ESP_ERROR_CHECK(uart_get_buffered_data_len(BOARD2BOARD_PORT_NUM, (size_t*)&length));
        while (length > 0) {
```

A7 – mesh_main.c

```
135.             ESP_LOGI(MESH_TAG, "uart rx length %d", length );
136.             // Prüfung, ob mindestens ein Datenpaket mit 9 Zeichen vorhanden ist
137.             if (length >= 9) {
138.                 // Versuche 9 Bytes zu lesen
139.                 if ( uart_read_bytes(BOARD2BOARD_PORT_NUM, serial_data, 9, 100) ) {
140.                     esp_log_buffer_hex("Read data:", serial_data, 9);
141.                     // MQTT-Nachricht für Beacon und Hub erstellen
142.                     asprintf(&print, "layer=%d, MAC_ROOM=%02X:%02X:%02X:%02X:%02X:%02X,
                                    MAC_SENSOR=%02X:%02X:%02X:%02X:%02X:%02X, BATT=%d,
                                    BUTTON=%d, RSSI=%d", \
143.                         esp_mesh_get_layer(), mac[0], mac[1], mac[2], mac[3], mac[4],
                            mac[5], \
144.                         serial_data[0], serial_data[1], serial_data[2], serial_data[3],
                            serial_data[4], serial_data[5], \
145.                         serial_data[6], serial_data[7], serial_data[8] );
146.                     ESP_LOGI(MESH_TAG, "Tried to publish %s", print);
147.                     mqtt_app_publish("/sensors/rooms", print);
148.                     free(print);
149.                 }
150.             } else {
151.                 // Wenn dieser Zustand ein zweites Mal mit mehr als 0 aber weniger als 9
                   Zeichen erreicht wird, muss ein Fehler in der Kommunikation aufgetreten sein
152.                 // Daher vorsichtshalber den Buffer leeren
153.                 if (is_broken_package == true) {
154.                     ESP_ERROR_CHECK(uart_flush(BOARD2BOARD_PORT_NUM));
155.                     is_broken_package = false;
156.                 } else is_broken_package = true;
157.             }
158.         }
159.         vTaskDelay(60000 / portTICK_PERIOD_MS);
160.         // MQTT-Nachricht für Hub erstellen
161.         asprintf(&print, "layer=%d, MAC_ROOM=%02X:%02X:%02X:%02X:%02X:%02X", \
162.             esp_mesh_get_layer(), mac[0], mac[1], mac[2], mac[3], mac[4], mac[5] );
163.         ESP_LOGI(MESH_TAG, "Publish %s", print);
164.         mqtt_app_publish("/sensors/rooms", print);
165.     }
166.     vTaskDelete(NULL);
167. }
168.
169. esp_err_t esp_mesh_comm_mqtt_task_start(void)
170. {
171.     static bool is_comm_mqtt_task_started = false;
172.     s_route_table_lock = xSemaphoreCreateMutex();
173.     if (!is_comm_mqtt_task_started) {
174.         xTaskCreate(esp_mesh_mqtt_task, "mqtt task", 3072, NULL, 5, NULL);
175.         is_comm_mqtt_task_started = true;
176.     }
177.     return ESP_OK;
178. }
179.
180. void mesh_event_handler(void *arg, esp_event_base_t event_base,
181.                         int32_t event_id, void *event_data)
182. {
183.     mesh_addr_t id = {0,};
184.     static uint8_t last_layer = 0;
185.     switch (event_id) {
186.     case MESH_EVENT_STARTED: {
187.         esp_mesh_get_id(&id);
188.         ESP_LOGI(MESH_TAG, "<MESH_EVENT_MESH_STARTED>ID:"MACSTR"", MAC2STR(id.addr));
189.         mesh_layer = esp_mesh_get_layer();
190.     }
191.     break;
192.     case MESH_EVENT_STOPPED: {
193.         ESP_LOGI(MESH_TAG, "<MESH_EVENT_STOPPED>");
194.         mesh_layer = esp_mesh_get_layer();
195.     }
196.     break;
197.     case MESH_EVENT_CHILD_CONNECTED: {
198.         mesh_event_child_connected_t *child_connected = (mesh_event_child_connected_t
            *)event_data;
199.         ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHILD_CONNECTED>aid:%d, "MACSTR"",
```

A7 – mesh_main.c

```
200.                 child_connected->aid,
201.                 MAC2STR(child_connected->mac));
202.     }
203.     break;
204.     case MESH_EVENT_CHILD_DISCONNECTED: {
205.         mesh_event_child_disconnected_t *child_disconnected =
            (mesh_event_child_disconnected_t *)event_data;
206.         ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHILD_DISCONNECTED>aid:%d, "MACSTR"",
207.                 child_disconnected->aid,
208.                 MAC2STR(child_disconnected->mac));
209.     }
210.     break;
211.     case MESH_EVENT_ROUTING_TABLE_ADD: {
212.         mesh_event_routing_table_change_t *routing_table =
            (mesh_event_routing_table_change_t *)event_data;
213.         ESP_LOGW(MESH_TAG, "<MESH_EVENT_ROUTING_TABLE_ADD>add %d, new:%d",
214.                 routing_table->rt_size_change,
215.                 routing_table->rt_size_new);
216.     }
217.     break;
218.     case MESH_EVENT_ROUTING_TABLE_REMOVE: {
219.         mesh_event_routing_table_change_t *routing_table =
            (mesh_event_routing_table_change_t *)event_data;
220.         ESP_LOGW(MESH_TAG, "<MESH_EVENT_ROUTING_TABLE_REMOVE>remove %d, new:%d",
221.                 routing_table->rt_size_change,
222.                 routing_table->rt_size_new);
223.     }
224.     break;
225.     case MESH_EVENT_NO_PARENT_FOUND: {
226.         mesh_event_no_parent_found_t *no_parent = (mesh_event_no_parent_found_t
                                                    *)event_data;
227.         ESP_LOGI(MESH_TAG, "<MESH_EVENT_NO_PARENT_FOUND>scan times:%d",
228.                 no_parent->scan_times);
229.     }
230.     /* TODO handler for the failure */
231.     break;
232.     case MESH_EVENT_PARENT_CONNECTED: {
233.         mesh_event_connected_t *connected = (mesh_event_connected_t *)event_data;
234.         esp_mesh_get_id(&id);
235.         mesh_layer = connected->self_layer;
236.         memcpy(&mesh_parent_addr.addr, connected->connected.bssid, 6);
237.         ESP_LOGI(MESH_TAG,
238.                 "<MESH_EVENT_PARENT_CONNECTED>layer=%d-->%d, parent="MACSTR"%s,
                    ID="MACSTR"",
239.                 last_layer, mesh_layer, MAC2STR(mesh_parent_addr.addr),
240.                 esp_mesh_is_root() ? "<ROOT>" :
241.                 (mesh_layer == 2) ? "<layer2>" : "", MAC2STR(id.addr));
242.         last_layer = mesh_layer;
243.         mesh_netifs_start(esp_mesh_is_root());
244.     }
245.     break;
246.     case MESH_EVENT_PARENT_DISCONNECTED: {
247.         mesh_event_disconnected_t *disconnected = (mesh_event_disconnected_t *)event_data;
248.         ESP_LOGI(MESH_TAG,
249.                 "<MESH_EVENT_PARENT_DISCONNECTED>reason:%d",
250.                 disconnected->reason);
251.         mesh_layer = esp_mesh_get_layer();
252.         mesh_netifs_stop();
253.     }
254.     break;
255.     case MESH_EVENT_LAYER_CHANGE: {
256.         mesh_event_layer_change_t *layer_change = (mesh_event_layer_change_t *)event_data;
257.         mesh_layer = layer_change->new_layer;
258.         ESP_LOGI(MESH_TAG, "<MESH_EVENT_LAYER_CHANGE>layer=%d-->%d%s",
259.                 last_layer, mesh_layer,
260.                 esp_mesh_is_root() ? "<ROOT>" :
261.                 (mesh_layer == 2) ? "<layer2>" : "");
262.         last_layer = mesh_layer;
263.     }
264.     break;
265.     case MESH_EVENT_ROOT_ADDRESS: {
```

```
266.          mesh_event_root_address_t *root_addr = (mesh_event_root_address_t *)event_data;
267.          ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_ADDRESS>root address:"MACSTR"",
268.                   MAC2STR(root_addr->addr));
269.      }
270.      break;
271.      case MESH_EVENT_VOTE_STARTED: {
272.          mesh_event_vote_started_t *vote_started = (mesh_event_vote_started_t *)event_data;
273.          ESP_LOGI(MESH_TAG,
274.                   "<MESH_EVENT_VOTE_STARTED>attempts:%d, reason:%d, rc_addr:"MACSTR"",
275.                   vote_started->attempts,
276.                   vote_started->reason,
277.                   MAC2STR(vote_started->rc_addr.addr));
278.      }
279.      break;
280.      case MESH_EVENT_VOTE_STOPPED: {
281.          ESP_LOGI(MESH_TAG, "<MESH_EVENT_VOTE_STOPPED>");
282.          break;
283.      }
284.      case MESH_EVENT_ROOT_SWITCH_REQ: {
285.          mesh_event_root_switch_req_t *switch_req = (mesh_event_root_switch_req_t
                                                        *)event_data;
286.          ESP_LOGI(MESH_TAG,
287.                   "<MESH_EVENT_ROOT_SWITCH_REQ>reason:%d, rc_addr:"MACSTR"",
288.                   switch_req->reason,
289.                   MAC2STR( switch_req->rc_addr.addr));
290.      }
291.      break;
292.      case MESH_EVENT_ROOT_SWITCH_ACK: {
293.          /* new root */
294.          mesh_layer = esp_mesh_get_layer();
295.          esp_mesh_get_parent_bssid(&mesh_parent_addr);
296.          ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_SWITCH_ACK>layer:%d, parent:"MACSTR"",
                     mesh_layer, MAC2STR(mesh_parent_addr.addr));
297.      }
298.      break;
299.      case MESH_EVENT_TODS_STATE: {
300.          mesh_event_toDS_state_t *toDs_state = (mesh_event_toDS_state_t *)event_data;
301.          ESP_LOGI(MESH_TAG, "<MESH_EVENT_TODS_REACHABLE>state:%d", *toDs_state);
302.      }
303.      break;
304.      case MESH_EVENT_ROOT_FIXED: {
305.          mesh_event_root_fixed_t *root_fixed = (mesh_event_root_fixed_t *)event_data;
306.          ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROOT_FIXED>%s",
307.                   root_fixed->is_fixed ? "fixed" : "not fixed");
308.      }
309.      break;
310.      case MESH_EVENT_ROOT_ASKED_YIELD: {
311.          mesh_event_root_conflict_t *root_conflict = (mesh_event_root_conflict_t
                                                         *)event_data;
312.          ESP_LOGI(MESH_TAG,
313.                   "<MESH_EVENT_ROOT_ASKED_YIELD>"MACSTR", rssi:%d, capacity:%d",
314.                   MAC2STR(root_conflict->addr),
315.                   root_conflict->rssi,
316.                   root_conflict->capacity);
317.      }
318.      break;
319.      case MESH_EVENT_CHANNEL_SWITCH: {
320.          mesh_event_channel_switch_t *channel_switch = (mesh_event_channel_switch_t
                                                           *)event_data;
321.          ESP_LOGI(MESH_TAG, "<MESH_EVENT_CHANNEL_SWITCH>new channel:%d", channel_switch-
                 >channel);
322.      }
323.      break;
324.      case MESH_EVENT_SCAN_DONE: {
325.          mesh_event_scan_done_t *scan_done = (mesh_event_scan_done_t *)event_data;
326.          ESP_LOGI(MESH_TAG, "<MESH_EVENT_SCAN_DONE>number:%d",
327.                   scan_done->number);
328.      }
329.      break;
330.      case MESH_EVENT_NETWORK_STATE: {
```

```
331.         mesh_event_network_state_t *network_state = (mesh_event_network_state_t
                                                         *)event_data;
332.         ESP_LOGI(MESH_TAG, "<MESH_EVENT_NETWORK_STATE>is_rootless:%d",
333.                  network_state->is_rootless);
334.     }
335.     break;
336.     case MESH_EVENT_STOP_RECONNECTION: {
337.         ESP_LOGI(MESH_TAG, "<MESH_EVENT_STOP_RECONNECTION>");
338.     }
339.     break;
340.     case MESH_EVENT_FIND_NETWORK: {
341.         mesh_event_find_network_t *find_network = (mesh_event_find_network_t *)event_data;
342.         ESP_LOGI(MESH_TAG, "<MESH_EVENT_FIND_NETWORK>new channel:%d, router BSSID:"MACSTR"",
343.                  find_network->channel, MAC2STR(find_network->router_bssid));
344.     }
345.     break;
346.     case MESH_EVENT_ROUTER_SWITCH: {
347.         mesh_event_router_switch_t *router_switch = (mesh_event_router_switch_t
                                                         *)event_data;
348.         ESP_LOGI(MESH_TAG, "<MESH_EVENT_ROUTER_SWITCH>new router:%s, channel:%d, "MACSTR"",
349.                  router_switch->ssid, router_switch->channel, MAC2STR(router_switch-
                      >bssid));
350.     }
351.     break;
352.     default:
353.         ESP_LOGI(MESH_TAG, "unknown id:%d", event_id);
354.         break;
355.     }
356. }
357.
358. void ip_event_handler(void *arg, esp_event_base_t event_base,
359.                       int32_t event_id, void *event_data)
360. {
361.     ip_event_got_ip_t *event = (ip_event_got_ip_t *) event_data;
362.     ESP_LOGI(MESH_TAG, "<IP_EVENT_STA_GOT_IP>IP:" IPSTR, IP2STR(&event->ip_info.ip));
363.     s_current_ip.addr = event->ip_info.ip.addr;
364. #if !CONFIG_MESH_USE_GLOBAL_DNS_IP
365.     esp_netif_t *netif = event->esp_netif;
366.     esp_netif_dns_info_t dns;
367.     ESP_ERROR_CHECK(esp_netif_get_dns_info(netif, ESP_NETIF_DNS_MAIN, &dns));
368.     mesh_netif_start_root_ap(esp_mesh_is_root(), dns.ip.u_addr.ip4.addr);
369. #endif
370.     esp_mesh_comm_mqtt_task_start();
371. }
372.
373. void app_main(void)
374. {
375.     ESP_ERROR_CHECK(nvs_flash_init());
376.     /*  tcpip initialization */
377.     ESP_ERROR_CHECK(esp_netif_init());
378.     /*  event initialization */
379.     ESP_ERROR_CHECK(esp_event_loop_create_default());
380.     /*  crete network interfaces for mesh (only station instance saved for further
          manipulation, soft AP instance ignored */
381.     ESP_ERROR_CHECK(mesh_netifs_init(recv_cb));
382.
383.     /*  wifi initialization */
384.     wifi_init_config_t config = WIFI_INIT_CONFIG_DEFAULT();
385.     ESP_ERROR_CHECK(esp_wifi_init(&config));
386.     ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT, IP_EVENT_STA_GOT_IP,
                        &ip_event_handler, NULL));
387.     ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_FLASH));
388.     ESP_ERROR_CHECK(esp_wifi_set_ps(WIFI_PS_NONE));
389.     ESP_ERROR_CHECK(esp_wifi_start());
390.     /*  mesh initialization */
391.     ESP_ERROR_CHECK(esp_mesh_init());
392.     ESP_ERROR_CHECK(esp_event_handler_register(MESH_EVENT, ESP_EVENT_ANY_ID,
                        &mesh_event_handler, NULL));
393.     ESP_ERROR_CHECK(esp_mesh_set_max_layer(CONFIG_MESH_MAX_LAYER));
394.     ESP_ERROR_CHECK(esp_mesh_set_vote_percentage(1));
395.     ESP_ERROR_CHECK(esp_mesh_set_ap_assoc_expire(10));
```

A7 – mesh_main.c

```
396.     mesh_cfg_t cfg = MESH_INIT_CONFIG_DEFAULT();
397.     /* mesh ID */
398.     memcpy((uint8_t *) &cfg.mesh_id, MESH_ID, 6);
399.     /* router */
400.     cfg.channel = CONFIG_MESH_CHANNEL;
401.     cfg.router.ssid_len = strlen(CONFIG_MESH_ROUTER_SSID);
402.     memcpy((uint8_t *) &cfg.router.ssid, CONFIG_MESH_ROUTER_SSID, cfg.router.ssid_len);
403.     memcpy((uint8_t *) &cfg.router.password, CONFIG_MESH_ROUTER_PASSWD,
404.            strlen(CONFIG_MESH_ROUTER_PASSWD));
405.     /* mesh softAP */
406.     ESP_ERROR_CHECK(esp_mesh_set_ap_authmode(CONFIG_MESH_AP_AUTHMODE));
407.     cfg.mesh_ap.max_connection = CONFIG_MESH_AP_CONNECTIONS;
408.     cfg.mesh_ap.nonmesh_max_connection = CONFIG_MESH_NON_MESH_AP_CONNECTIONS;
409.     memcpy((uint8_t *) &cfg.mesh_ap.password, CONFIG_MESH_AP_PASSWD,
410.            strlen(CONFIG_MESH_AP_PASSWD));
411.     ESP_ERROR_CHECK(esp_mesh_set_config(&cfg));
412. 
413.     // UART Parameter konfigurieren
414.     ESP_ERROR_CHECK(uart_param_config(BOARD2BOARD_PORT_NUM, &uart_config));
415.     // Setze UART Pins(TX: IO4, RX: IO5, RTS: IO18, CTS: IO19)
416.     ESP_ERROR_CHECK(uart_set_pin(BOARD2BOARD_PORT_NUM, BOARD2BOARD_TXD, BOARD2BOARD_RXD,
                        BOARD2BOARD_RTS, BOARD2BOARD_CTS));
417. 
418.     // UART  Treiber installieren
419.     ESP_ERROR_CHECK( uart_driver_install(BOARD2BOARD_PORT_NUM, BOARD2BOARD_BUF_SIZE, \
420.                         BOARD2BOARD_BUF_SIZE, 20, NULL, 0) ); // no event queue defined (NULL)
421.     /* mesh start */
422.     ESP_ERROR_CHECK(esp_mesh_start());
423.     ESP_LOGI(MESH_TAG, "mesh starts successfully, heap:%d, %s\n",  esp_get_free_heap_size(),
424.              esp_mesh_is_root_fixed() ? "root fixed" : "root not fixed");
425. 
426.     ESP_ERROR_CHECK(uart_flush(BOARD2BOARD_PORT_NUM));
427. }
428. 
```

A7 – mesh_main.c

A8 – handle_publish.c

```
1. /*
2. Copyright (c) 2009-2020 Roger Light <roger@atchoo.org>
3.
4. All rights reserved. This program and the accompanying materials
5. are made available under the terms of the Eclipse Public License 2.0
6. and Eclipse Distribution License v1.0 which accompany this distribution.
7.
8. The Eclipse Public License is available at
9.    https://www.eclipse.org/legal/epl-2.0/
10. and the Eclipse Distribution License is available at
11.   http://www.eclipse.org/org/documents/edl-v10.php.
12.
13. SPDX-License-Identifier: EPL-2.0 OR BSD-3-Clause
14.
15. Contributors:
16.    Roger Light - initial implementation and documentation.
17. */
18.
19. #include "config.h"
20.
21. #include <assert.h>
22. #include <stdio.h>
23. #include <string.h>
24.
25. #include "mosquitto_broker_internal.h"
26. #include "alias_mosq.h"
27. #include "mqtt_protocol.h"
28. #include "memory_mosq.h"
29. #include "packet_mosq.h"
30. #include "property_mosq.h"
31. #include "read_handle.h"
32. #include "send_mosq.h"
33. #include "sys_tree.h"
34. #include "util_mosq.h"
35. #include <time.h> // Um Zeitstempel zur aktuellen Uhrzeit abzufragen
36.
37. int handle__publish(struct mosquitto *context)
38. {
39.     uint8_t dup;
40.     int rc = 0;
41.     int rc2;
42.     uint8_t header = context->in_packet.command;
43.     int res = 0;
44.     struct mosquitto_msg_store *msg, *stored = NULL;
45.     struct mosquitto_client_msg *cmsg_stored = NULL;
46.     size_t len;
47.     uint16_t slen;
48.     char *topic_mount;
49.     mosquitto_property *properties = NULL;
50.     mosquitto_property *p, *p_prev;
51.     mosquitto_property *msg_properties_last;
52.     uint32_t message_expiry_interval = 0;
53.     int topic_alias = -1;
54.     uint8_t reason_code = 0;
55.     uint16_t mid = 0;
56.
57.     if(context->state != mosq_cs_active){
58.         return MOSQ_ERR_PROTOCOL;
59.     }
60.     // sadfsadfa
61.     msg = mosquitto__calloc(1, sizeof(struct mosquitto_msg_store));
62.     if(msg == NULL){
63.         return MOSQ_ERR_NOMEM;
64.     }
65.
66.     dup = (header & 0x08)>>3;
67.     msg->qos = (header & 0x06)>>1;
68.     if(dup == 1 && msg->qos == 0){
```

```
69.         log__printf(NULL, MOSQ_LOG_INFO,
70.                 "Invalid PUBLISH (QoS=0 and DUP=1) from %s, disconnecting.", context->id);
71.         db__msg_store_free(msg);
72.         return MOSQ_ERR_MALFORMED_PACKET;
73.     }
74.     if(msg->qos == 3){
75.         log__printf(NULL, MOSQ_LOG_INFO,
76.                 "Invalid QoS in PUBLISH from %s, disconnecting.", context->id);
77.         db__msg_store_free(msg);
78.         return MOSQ_ERR_MALFORMED_PACKET;
79.     }
80.     if(msg->qos > context->max_qos){
81.         log__printf(NULL, MOSQ_LOG_INFO,
82.                 "Too high QoS in PUBLISH from %s, disconnecting.", context->id);
83.         db__msg_store_free(msg);
84.         return MOSQ_ERR_QOS_NOT_SUPPORTED;
85.     }
86.     msg->retain = (header & 0x01);
87.
88.     if(msg->retain && db.config->retain_available == false){
89.         db__msg_store_free(msg);
90.         return MOSQ_ERR_RETAIN_NOT_SUPPORTED;
91.     }
92.
93.     if(packet__read_string(&context->in_packet, &msg->topic, &slen)){
94.         db__msg_store_free(msg);
95.         return MOSQ_ERR_MALFORMED_PACKET;
96.     }
97.     if(!slen && context->protocol != mosq_p_mqtt5){
98.         /* Invalid publish topic, disconnect client. */
99.         db__msg_store_free(msg);
100.        return MOSQ_ERR_MALFORMED_PACKET;
101.    }
102.
103.    if(msg->qos > 0){
104.        if(packet__read_uint16(&context->in_packet, &mid)){
105.            db__msg_store_free(msg);
106.            return MOSQ_ERR_MALFORMED_PACKET;
107.        }
108.        if(mid == 0){
109.            db__msg_store_free(msg);
110.            return MOSQ_ERR_PROTOCOL;
111.        }
112.        /* It is important to have a separate copy of mid, because msg may be
113.         * freed before we want to send a PUBACK/PUBREC. */
114.        msg->source_mid = mid;
115.    }
116.
117.    /* Handle properties */
118.    if(context->protocol == mosq_p_mqtt5){
119.        rc = property__read_all(CMD_PUBLISH, &context->in_packet, &properties);
120.        if(rc){
121.            db__msg_store_free(msg);
122.            return rc;
123.        }
124.
125.        p = properties;
126.        p_prev = NULL;
127.        msg->properties = NULL;
128.        msg_properties_last = NULL;
129.        while(p){
130.            switch(p->identifier){
131.                case MQTT_PROP_CONTENT_TYPE:
132.                case MQTT_PROP_CORRELATION_DATA:
133.                case MQTT_PROP_PAYLOAD_FORMAT_INDICATOR:
134.                case MQTT_PROP_RESPONSE_TOPIC:
135.                case MQTT_PROP_USER_PROPERTY:
136.                    if(msg->properties){
137.                        msg_properties_last->next = p;
138.                        msg_properties_last = p;
139.                    }else{
```

A8 – handle_publish.c

```
                        msg->properties = p;
                        msg_properties_last = p;
                    }
                    if(p_prev){
                        p_prev->next = p->next;
                        p = p_prev->next;
                    }else{
                        properties = p->next;
                        p = properties;
                    }
                    msg_properties_last->next = NULL;
                    break;

                case MQTT_PROP_TOPIC_ALIAS:
                    topic_alias = p->value.i16;
                    p_prev = p;
                    p = p->next;
                    break;

                case MQTT_PROP_MESSAGE_EXPIRY_INTERVAL:
                    message_expiry_interval = p->value.i32;
                    p_prev = p;
                    p = p->next;
                    break;

                case MQTT_PROP_SUBSCRIPTION_IDENTIFIER:
                    p_prev = p;
                    p = p->next;
                    break;

                default:
                    p = p->next;
                    break;
            }
        }
    }
    mosquitto_property_free_all(&properties);

    if(topic_alias == 0 || (context->listener && topic_alias > context->listener-
                                >max_topic_alias)){
        db__msg_store_free(msg);
        return MOSQ_ERR_TOPIC_ALIAS_INVALID;
    }else if(topic_alias > 0){
        if(msg->topic){
            rc = alias__add(context, msg->topic, (uint16_t)topic_alias);
            if(rc){
                db__msg_store_free(msg);
                return rc;
            }
        }else{
            rc = alias__find(context, &msg->topic, (uint16_t)topic_alias);
            if(rc){
                db__msg_store_free(msg);
                return MOSQ_ERR_PROTOCOL;
            }
        }
    }

#ifdef WITH_BRIDGE
    rc = bridge__remap_topic_in(context, &msg->topic);
    if(rc){
        db__msg_store_free(msg);
        return rc;
    }

#endif
    if(mosquitto_pub_topic_check(msg->topic) != MOSQ_ERR_SUCCESS){
        /* Invalid publish topic, just swallow it. */
        db__msg_store_free(msg);
        return MOSQ_ERR_MALFORMED_PACKET;
    }
```

A8 – handle_publish.c

```
210.
211.     msg->payloadlen = context->in_packet.remaining_length - context->in_packet.pos;
212.     G_PUB_BYTES_RECEIVED_INC(msg->payloadlen);
213.     if(context->listener && context->listener->mount_point){
214.         len = strlen(context->listener->mount_point) + strlen(msg->topic) + 1;
215.         topic_mount = mosquitto__malloc(len+1);
216.         if(!topic_mount){
217.             db__msg_store_free(msg);
218.             return MOSQ_ERR_NOMEM;
219.         }
220.         snprintf(topic_mount, len, "%s%s", context->listener->mount_point, msg->topic);
221.         topic_mount[len] = '\0';
222.
223.         mosquitto__free(msg->topic);
224.         msg->topic = topic_mount;
225.     }
226.
227.     if(msg->payloadlen){
228.         if(db.config->message_size_limit && msg->payloadlen > db.config-
                >message_size_limit){
229.             log__printf(NULL, MOSQ_LOG_DEBUG, "Dropped too large PUBLISH from %s (d%d, q%d,
                                                  r%d, m%d, '%s', ... (%ld bytes))", context->id,
                                                  dup, msg->qos, msg->retain, msg->source_mid,
                                                  msg->topic, (long)msg->payloadlen);
230.             reason_code = MQTT_RC_PACKET_TOO_LARGE;
231.             goto process_bad_message;
232.         }
233.
234.         /* Verkapselung mit des Payload mit {time=now(),data=payload} */
235.         // Abfrage der aktuellen Uhrzeit
236.         time_t current_time;
237.         time(&current_time);
238.
239.         // Erstellen eines neuen Strings, der die Zeit und das ursprüngliche JSON-Payload
             enthält
240.         char *new_json_str = mosquitto__malloc(msg->payloadlen + 64); // Zusätzlicher
                                                                          Speicherplatz für den
                                                                          Zeitstempel und Trennzeichen
241.         if (new_json_str == NULL) {
242.             fprintf(stderr, "Memory allocation failed\n");
243.             exit(1);
244.         }
245.
246.         // Erstellen eines neuen Strings, der die ursprüngliche Nachricht beinhaltet
247.         char *tmp_msg = mosquitto__malloc(msg->payloadlen);
248.         if (tmp_msg == NULL) {
249.             fprintf(stderr, "Memory allocation failed\n");
250.             exit(1);
251.         }
252.
253.         // Speicherzuweisung für die endgülttige Nachricht
254.         msg->payload = mosquitto__malloc(msg->payloadlen+64); // Zusätzlicher Speicherplatz
                                    für den Zeitstempel und Trennzeichen ( war vorher: msg-
                                    >payload = mosquitto__malloc(msg->payloadlen+1); )
255.         if(msg->payload == NULL){
256.             db__msg_store_free(msg);
257.             return MOSQ_ERR_NOMEM;
258.         }
259.         /* Ensure final payload is always zero terminated, this is the reason for the extra
             byte above */
260.         ((uint8_t *)msg->payload)[msg->payloadlen+63] = 0;
261.
262.         if(packet__read_bytes(&context->in_packet, tmp_msg, msg->payloadlen)){ // Nachricht
                                                                                  in tmp_msg speichern
263.             db__msg_store_free(msg);
264.             mosquitto_free(new_json_str); // Löschen, falls Error
265.                         mosquitto_free(tmp_msg);
266.             return MOSQ_ERR_MALFORMED_PACKET;
267.         }
268.                // terminate string!!
269.                 //(uint8_t *)tmp_msg[msg->payloadlen] = 0;
```

A8 – handle_publish.c

```
270.
271.         /* Ensure payload is always zero terminated */
272.         ((uint8_t *)tmp_msg)[msg->payloadlen] = 0;
273.
274.         // Umwandlung des Payload-void-Pointers in einen char-Pointer zur erleichterten
              Manipulation
275.         char *json_str = (char *)tmp_msg;
276.
277.         // Verketten von Zeitstempel und ursprünglichem Payload
278.         sprintf(new_json_str, "{\"time\":\"%ld\",\"data\":%s}", current_time, json_str);
279.
280.         // Anschließendes übergeben des String 'new_json_str' an nachfolgende Funktionen
281.         memcpy(msg->payload, new_json_str, (size_t)strlen(new_json_str));
282.         msg->payloadlen = (uint32_t)strlen(new_json_str);
283.         mosquitto_free(new_json_str); // clean up on error.
284.         mosquitto_free(tmp_msg);
285.
286.     }
287.     /* Check for topic access */
288.     rc = mosquitto_acl_check(context, msg->topic, msg->payloadlen, msg->payload, msg->qos,
                                 msg->retain, MOSQ_ACL_WRITE);
289.     if(rc == MOSQ_ERR_ACL_DENIED){
290.         log__printf(NULL, MOSQ_LOG_DEBUG,
291.                 "Denied PUBLISH from %s (d%d, q%d, r%d, m%d, '%s', ... (%ld bytes))",
292.                 context->id, dup, msg->qos, msg->retain, msg->source_mid, msg->topic,
293.                 (long)msg->payloadlen);
294.         reason_code = MQTT_RC_NOT_AUTHORIZED;
295.         goto process_bad_message;
296.     }else if(rc != MOSQ_ERR_SUCCESS){
297.         db__msg_store_free(msg);
298.         return rc;
299.     }
300.
301.     log__printf(NULL, MOSQ_LOG_DEBUG, "Received PUBLISH from %s (d%d, q%d, r%d, m%d, '%s',
... (%ld bytes))", context->id, dup, msg->qos, msg->retain, msg->source_mid, msg->topic,
(long)msg->payloadlen);
302.
303.     if(!strncmp(msg->topic, "$CONTROL/", 9)){
304. #ifdef WITH_CONTROL
305.         rc = control__process(context, msg);
306.         db__msg_store_free(msg);
307.         return rc;
308. #else
309.         reason_code = MQTT_RC_IMPLEMENTATION_SPECIFIC;
310.         goto process_bad_message;
311. #endif
312.     }
313.
314.     {
315.         rc = plugin__handle_message(context, msg);
316.         if(rc == MOSQ_ERR_ACL_DENIED){
317.             log__printf(NULL, MOSQ_LOG_DEBUG,
318.                     "Denied PUBLISH from %s (d%d, q%d, r%d, m%d, '%s', ... (%ld bytes))",
319.                     context->id, dup, msg->qos, msg->retain, msg->source_mid, msg->topic,
320.                     (long)msg->payloadlen);
321.
322.             reason_code = MQTT_RC_NOT_AUTHORIZED;
323.             goto process_bad_message;
324.         }else if(rc != MOSQ_ERR_SUCCESS){
325.             db__msg_store_free(msg);
326.             return rc;
327.         }
328.     }
329.
330.     if(msg->qos > 0){
331.         db__message_store_find(context, msg->source_mid, &cmsg_stored);
332.     }
333.
334.     if(cmsg_stored && cmsg_stored->store && msg->source_mid != 0 &&
335.             (cmsg_stored->store->qos != msg->qos
336.              || cmsg_stored->store->payloadlen != msg->payloadlen
```

A8 – handle_publish.c

```
337.                 || strcmp(cmsg_stored->store->topic, msg->topic)
338.                 || memcmp(cmsg_stored->store->payload, msg->payload, msg->payloadlen) )){
339.
340.         log__printf(NULL, MOSQ_LOG_WARNING, "Reused message ID %u from %s detected. Clearing
from storage.", msg->source_mid, context->id);
341.         db__message_remove_incoming(context, msg->source_mid);
342.         cmsg_stored = NULL;
343.     }
344.
345.     if(!cmsg_stored){
346.         if(msg->qos == 0
347.                 || db__ready_for_flight(context, mosq_md_in, msg->qos)
348.                 ){
349.
350.             dup = 0;
351.             rc = db__message_store(context, msg, message_expiry_interval, 0,
mosq_mo_client);
352.             if(rc) return rc;
353.         }else{
354.             /* Client isn't allowed any more incoming messages, so fail early */
355.             reason_code = MQTT_RC_QUOTA_EXCEEDED;
356.             goto process_bad_message;
357.         }
358.         stored = msg;
359.         msg = NULL;
360.         dup = 0;
361.     }else{
362.         db__msg_store_free(msg);
363.         msg = NULL;
364.         stored = cmsg_stored->store;
365.         cmsg_stored->dup++;
366.         dup = cmsg_stored->dup;
367.     }
368.
369.     switch(stored->qos){
370.         case 0:
371.             rc2 = sub__messages_queue(context->id, stored->topic, stored->qos, stored-
>retain, &stored);
372.             if(rc2 > 0) rc = 1;
373.             break;
374.         case 1:
375.             util__decrement_receive_quota(context);
376.             rc2 = sub__messages_queue(context->id, stored->topic, stored->qos, stored-
>retain, &stored);
377.             /* stored may now be free, so don't refer to it */
378.             if(rc2 == MOSQ_ERR_SUCCESS || context->protocol != mosq_p_mqtt5){
379.                 if(send__puback(context, mid, 0, NULL)) rc = 1;
380.             }else if(rc2 == MOSQ_ERR_NO_SUBSCRIBERS){
381.                 if(send__puback(context, mid, MQTT_RC_NO_MATCHING_SUBSCRIBERS, NULL)) rc =
1;
382.             }else{
383.                 rc = rc2;
384.             }
385.             break;
386.         case 2:
387.             if(dup == 0){
388.                 res = db__message_insert(context, stored->source_mid, mosq_md_in, stored-
>qos, stored->retain, stored, NULL, false);
389.             }else{
390.                 res = 0;
391.             }
392.
393.             /* db__message_insert() returns 2 to indicate dropped message
394.              * due to queue. This isn't an error so don't disconnect them. */
395.             /* FIXME - this is no longer necessary due to failing early above */
396.             if(!res){
397.                 if(dup == 0 || dup == 1){
398.                     rc2 = send__pubrec(context, stored->source_mid, 0, NULL);
399.                     if(rc2) rc = rc2;
400.                 }else{
401.                     return MOSQ_ERR_PROTOCOL;
```

A8 – handle_publish.c

```
                }
            }else if(res == 1){
                rc = 1;
            }
            break;
    }

    db__message_write_queued_in(context);
    return rc;
process_bad_message:
    rc = 1;
    if(msg){
        switch(msg->qos){
            case 0:
                rc = MOSQ_ERR_SUCCESS;
                break;
            case 1:
                rc = send__puback(context, msg->source_mid, reason_code, NULL);
                break;
            case 2:
                rc = send__pubrec(context, msg->source_mid, reason_code, NULL);
                break;
        }
        db__msg_store_free(msg);
    }
    if(context->out_packet_count >= db.config->max_queued_messages){
        rc = MQTT_RC_QUOTA_EXCEEDED;
    }
    return rc;
}

```

A8 – handle_publish.c

A9 – SQL-DB

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
      SQL_MODE='STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBS
      TITUTION';

-- -----------------------------------------------------
-- Schema ATSDB
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `ATSDB` DEFAULT CHARACTER SET utf8mb4;
USE `ATSDB`;

-- -----------------------------------------------------
-- Table `ATSDB`.`bereich`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`bereich` (
  `bereich_id` INT NOT NULL AUTO_INCREMENT,
  `bereich_name` VARCHAR(45) NOT NULL,
  `bereich_aktiv_seit` DATETIME NULL DEFAULT NULL,
  `bereich_inaktiv_seit` DATETIME NULL DEFAULT NULL,
  `bereich_aktiv` BIT(1) NOT NULL DEFAULT b'0' COMMENT 'Boolean. Flag für aktiven Bereich',
  PRIMARY KEY (`bereich_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARACTER SET=utf8mb4;

-- -----------------------------------------------------
-- Table `ATSDB`.`raum`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`raum` (
  `raum_id` INT NOT NULL AUTO_INCREMENT,
  `raum_name` VARCHAR(255) NOT NULL,
  `raum_aktiv_seit` DATETIME NULL DEFAULT NULL,
  `raum_inaktiv_seit` DATETIME NULL DEFAULT NULL,
  `raum_bereich_id` INT NOT NULL,
  `raum_aktiv` BIT(1) NULL DEFAULT NULL COMMENT 'Boolean. Zeigt einen aktiven Raum.',
  PRIMARY KEY (`raum_id`),
  INDEX `fk_Raum_Bereich1_idx` (`raum_bereich_id` ASC),
  CONSTRAINT `fk_Raum_Bereich1`
    FOREIGN KEY (`raum_bereich_id`)
    REFERENCES `ATSDB`.`bereich` (`bereich_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARACTER SET=utf8mb4;

-- -----------------------------------------------------
-- Table `ATSDB`.`hub`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`hub` (
  `hub_id` INT NOT NULL AUTO_INCREMENT,
  `hub_MAC` VARCHAR(255) NOT NULL,
  `hub_aktiv` TINYINT NOT NULL,
  `hub_aktiv_seit` DATETIME NULL DEFAULT NULL,
  `hub_inaktiv_seit` DATETIME NULL DEFAULT NULL,
  `hub_raum_id` INT NULL DEFAULT NULL,
  `hub_raum_ts` INT NULL DEFAULT NULL,
  `hub_timestamp` INT NULL DEFAULT 0 COMMENT 'Timestamp der letzten \'aktiv\'-Meldung',
  PRIMARY KEY (`hub_id`),
  UNIQUE INDEX `hub_UUID_UNIQUE` (`hub_MAC` ASC),
  INDEX `fk_Hub_Raum1_idx` (`hub_raum_id` ASC),
  CONSTRAINT `fk_Hub_Raum1`
    FOREIGN KEY (`hub_raum_id`)
    REFERENCES `ATSDB`.`raum` (`raum_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARACTER SET=utf8mb4;

-- -----------------------------------------------------
-- Table `ATSDB`.`beacon`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`beacon` (
  `beacon_id` INT NOT NULL AUTO_INCREMENT,
  `beacon_MAC` VARCHAR(45) NOT NULL,
  `beacon_aktiv` TINYINT NOT NULL DEFAULT 0,
```

```
67.   `beacon_aktiv_seit` DATETIME NOT NULL DEFAULT '0000-00-00 00:00:00',
68.   `beacon_RSSI` INT NOT NULL DEFAULT 0,
69.   `beacon_batterie` INT NOT NULL DEFAULT 100,
70.   `beacon_inaktiv_seit` DATETIME NULL DEFAULT NULL,
71.   `beacon_hub_id` INT NULL DEFAULT NULL,
72.   `beacon_timestamp` INT NOT NULL DEFAULT 0,
73.   `beacon_hub_ts_beginn` INT NULL DEFAULT NULL,
74.   PRIMARY KEY (`beacon_id`),
75.   INDEX `fk_beacon_hub1_idx` (`beacon_hub_id` ASC),
76.   CONSTRAINT `fk_beacon_hub1`
77.     FOREIGN KEY (`beacon_hub_id`)
78.     REFERENCES `ATSDB`.`hub` (`hub_id`)
79.     ON DELETE NO ACTION
80.     ON UPDATE NO ACTION
81. ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARACTER SET=utf8mb4;
82.
83. -- -----------------------------------------------------
84. -- Table `ATSDB`.`beaconpair`
85. -- -----------------------------------------------------
86. CREATE TABLE IF NOT EXISTS `ATSDB`.`beaconpair` (
87.   `beaconpair_beacon_id_1` INT NOT NULL,
88.   `beaconpair_beacon_id_2` INT NOT NULL,
89.   `beaconpair_timestamp` INT NOT NULL,
90.   `beaconpair_hub_id` INT NOT NULL,
91.   PRIMARY KEY (`beaconpair_beacon_id_1`, `beaconpair_beacon_id_2`),
92.   INDEX `fk_beacon_has_beacon_beacon2_idx` (`beaconpair_beacon_id_2` ASC),
93.   INDEX `fk_beacon_has_beacon_beacon1_idx` (`beaconpair_beacon_id_1` ASC),
94.   INDEX `fk_beaconpair_hub1_idx` (`beaconpair_hub_id` ASC),
95.   CONSTRAINT `fk_beacon_has_beacon_beacon1`
96.     FOREIGN KEY (`beaconpair_beacon_id_1`)
97.     REFERENCES `ATSDB`.`beacon` (`beacon_id`)
98.     ON DELETE NO ACTION
99.     ON UPDATE NO ACTION,
100.  CONSTRAINT `fk_beacon_has_beacon_beacon2`
101.    FOREIGN KEY (`beaconpair_beacon_id_2`)
102.    REFERENCES `ATSDB`.`beacon` (`beacon_id`)
103.    ON DELETE NO ACTION
104.    ON UPDATE NO ACTION,
105.  CONSTRAINT `fk_beaconpair_hub1`
106.    FOREIGN KEY (`beaconpair_hub_id`)
107.    REFERENCES `ATSDB`.`hub` (`hub_id`)
108.    ON DELETE NO ACTION
109.    ON UPDATE NO ACTION
110. ) ENGINE=InnoDB DEFAULT CHARACTER SET=utf8mb4;
111.
112. -- -----------------------------------------------------
113. -- Table `ATSDB`.`hub_historie`
114. -- -----------------------------------------------------
115. CREATE TABLE IF NOT EXISTS `ATSDB`.`hub_historie` (
116.   `hub_historie_id` BIGINT(20) NOT NULL AUTO_INCREMENT,
117.   `hub_historie_hub_id` INT NOT NULL,
118.   `hub_historie_raum_id` INT NOT NULL,
119.   `hub_historie_raum_ts_start` INT NULL DEFAULT NULL,
120.   `hub_historie_raum_ts_end` INT NULL DEFAULT NULL,
121.   PRIMARY KEY (`hub_historie_id`),
122.   INDEX `hub_historie_hub_id_idx` (`hub_historie_hub_id` ASC),
123.   INDEX `hub_historie_raum_id_idx` (`hub_historie_raum_id` ASC),
124.   CONSTRAINT `hub_historie_hub_id`
125.     FOREIGN KEY (`hub_historie_hub_id`)
126.     REFERENCES `ATSDB`.`hub` (`hub_id`)
127.     ON DELETE NO ACTION
128.     ON UPDATE NO ACTION,
129.   CONSTRAINT `hub_historie_raum_id`
130.     FOREIGN KEY (`hub_historie_raum_id`)
131.     REFERENCES `ATSDB`.`raum` (`raum_id`)
132.     ON DELETE NO ACTION
133.     ON UPDATE NO ACTION
134. ) ENGINE=InnoDB DEFAULT CHARACTER SET=utf8mb4 COMMENT='Änderung der Raumzuweisung von
Hubs.\\nNutzung: Absicherung der Gerätehistorie, da diese IDs referenziert.\\nBefüllung durch
AfterUpdate Trigger auf der Tabelle hub.';
135.
```

```
136. -- -----------------------------------------------------
137. -- Table `ATSDB`.`mp_typ`
138. -- -----------------------------------------------------
139. CREATE TABLE IF NOT EXISTS `ATSDB`.`mp_typ` (
140.   `mp_typ_id` INT NOT NULL AUTO_INCREMENT,
141.   `mp_typ_name` VARCHAR(255) NOT NULL,
142.   `mp_typ_aktiv` BIT(1) NULL DEFAULT b'0' COMMENT 'Boolean. Zeigt einen aktiven MP Typen.',
143.   PRIMARY KEY (`mp_typ_id`)
144. ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARACTER SET=utf8mb4;
145.
146. -- -----------------------------------------------------
147. -- Table `ATSDB`.`mp`
148. -- -----------------------------------------------------
149. CREATE TABLE IF NOT EXISTS `ATSDB`.`mp` (
150.   `mp_id` INT NOT NULL AUTO_INCREMENT,
151.   `mp_name` VARCHAR(255) NOT NULL,
152.   `mp_SN` VARCHAR(255) NOT NULL,
153.   `mp_aktiv_seit` DATETIME NULL DEFAULT NULL,
154.   `mp_inaktiv_seit` DATETIME NULL DEFAULT NULL,
155.   `mp_mp_typ_id` INT NOT NULL,
156.   `mp_beacon_id` INT NULL DEFAULT NULL,
157.   `mp_aktiv` BIT(1) NULL DEFAULT b'1' COMMENT 'Boolean. Zeigt eine aktives MP.',
158.   PRIMARY KEY (`mp_id`),
159.   INDEX `fk_mp_mptyp1_idx` (`mp_mp_typ_id` ASC),
160.   INDEX `fk_mp_beacon1_idx` (`mp_beacon_id` ASC),
161.   CONSTRAINT `fk_mp_beacon1`
162.     FOREIGN KEY (`mp_beacon_id`)
163.     REFERENCES `ATSDB`.`beacon` (`beacon_id`)
164.     ON DELETE NO ACTION
165.     ON UPDATE NO ACTION,
166.   CONSTRAINT `fk_mp_mptyp1`
167.     FOREIGN KEY (`mp_mp_typ_id`)
168.     REFERENCES `ATSDB`.`mp_typ` (`mp_typ_id`)
169. ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARACTER SET=utf8mb4;
170.
171. -- -----------------------------------------------------
172. -- Table `ATSDB`.`mp_historie`
173. -- -----------------------------------------------------
174. CREATE TABLE IF NOT EXISTS `ATSDB`.`mp_historie` (
175.   `mp_historie_id` BIGINT(20) NOT NULL AUTO_INCREMENT,
176.   `mp_historie_raum_id` INT NOT NULL,
177.   `mp_historie_mp_id` INT NOT NULL,
178.   `mp_historie_raum_ts_start` INT NOT NULL,
179.   `mp_historie_raum_ts_end` INT NOT NULL,
180.   PRIMARY KEY (`mp_historie_id`),
181.   INDEX `mp_historie_raum_id_idx` (`mp_historie_raum_id` ASC),
182.   INDEX `mp_historie_mp_id_idx` (`mp_historie_mp_id` ASC),
183.   CONSTRAINT `mp_historie_mp_id`
184.     FOREIGN KEY (`mp_historie_mp_id`)
185.     REFERENCES `ATSDB`.`mp` (`mp_id`)
186.     ON DELETE NO ACTION
187.     ON UPDATE NO ACTION,
188.   CONSTRAINT `mp_historie_raum_id`
189.     FOREIGN KEY (`mp_historie_raum_id`)
190.     REFERENCES `ATSDB`.`raum` (`raum_id`)
191.     ON DELETE NO ACTION
192.     ON UPDATE NO ACTION
193. ) ENGINE=InnoDB DEFAULT CHARACTER SET=utf8mb4;
194.
195. -- -----------------------------------------------------
196. -- Table `ATSDB`.`mp_mapping`
197. -- -----------------------------------------------------
198. CREATE TABLE IF NOT EXISTS `ATSDB`.`mp_mapping` (
199.   `mp_mapping_id` INT NOT NULL AUTO_INCREMENT,
200.   `mp_mapping_mp_typ_id_1` INT NOT NULL,
201.   `mp_mapping_mp_typ_id_2` INT NOT NULL,
202.   PRIMARY KEY (`mp_mapping_id`),
203.   INDEX `mp_mapping_mp_typ1_idx` (`mp_mapping_mp_typ_id_1` ASC),
204.   INDEX `mp_mapping_mp_typ2_idx` (`mp_mapping_mp_typ_id_2` ASC),
205.   CONSTRAINT `mp_mapping_mp_typ_id_1`
206.     FOREIGN KEY (`mp_mapping_mp_typ_id_1`)
```

```
    REFERENCES `ATSDB`.`mp_typ` (`mp_typ_id`),
  CONSTRAINT `mp_mapping_mp_typ_id_2`
    FOREIGN KEY (`mp_mapping_mp_typ_id_2`)
    REFERENCES `ATSDB`.`mp_typ` (`mp_typ_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARACTER SET=utf8mb4;

-- -----------------------------------------------------
-- Placeholder table for view `ATSDB`.`beacon_left_join_mp`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`beacon_left_join_mp` (
  `beacon_id` INT,
  `beacon_hub_id` INT,
  `beacon_RSSI` INT,
  `beacon_timestamp` INT,
  `beacon_hub_ts_beginn` INT,
  `beacon_batterie` INT,
  `beacon_MAC` INT,
  `mp_mp_typ_id` INT
);

-- -----------------------------------------------------
-- Placeholder table for view `ATSDB`.`beacon_mp`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`beacon_mp` (
  `beacon_id` INT,
  `beacon_MAC` INT,
  `beacon_RSSI` INT,
  `beacon_batterie` INT,
  `beacon_hub_id` INT,
  `beacon_timestamp` INT,
  `beacon_hub_ts_beginn` INT,
  `mp_id` INT,
  `mp_name` INT,
  `mp_mp_typ_id` INT
);

-- -----------------------------------------------------
-- Placeholder table for view `ATSDB`.`beacon_raum_bereich`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`beacon_raum_bereich` (
  `beacon_id` INT,
  `beacon_MAC` INT,
  `beacon_aktiv` INT,
  `beacon_aktiv_seit` INT,
  `beacon_inaktiv_seit` INT,
  `beacon_timestamp` INT,
  `beacon_hub_ts_beginn` INT,
  `beacon_hub_id` INT,
  `raum_id` INT,
  `raum_name` INT,
  `raum_aktiv` INT,
  `raum_aktiv_seit` INT,
  `raum_inaktiv_seit` INT,
  `bereich_name` INT,
  `bereich_id` INT
);

-- -----------------------------------------------------
-- Placeholder table for view `ATSDB`.`hub_raum_bereich`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`hub_raum_bereich` (
  `hub_id` INT,
  `hub_MAC` INT,
  `hub_aktiv` INT,
  `hub_aktiv_seit` INT,
  `hub_inaktiv_seit` INT,
  `hub_timestamp` INT,
  `hub_raum_ts` INT,
  `raum_id` INT,
  `raum_name` INT,
  `raum_aktiv` INT,
```

```
  `raum_aktiv_seit` INT,
  `raum_inaktiv_seit` INT,
  `bereich_name` INT,
  `bereich_id` INT
);

-- -----------------------------------------------------
-- Placeholder table for view `ATSDB`.`mapping_typ_namen`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`mapping_typ_namen` (
  `mp_mapping_id` INT,
  `mp_typ_name_1` INT,
  `mp_typ_name_2` INT,
  `mp_typ_id_1` INT,
  `mp_typ_id_2` INT
);

-- -----------------------------------------------------
-- Placeholder table for view `ATSDB`.`mp_raum_bereich`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`mp_raum_bereich` (
  `mp_id` INT,
  `mp_name` INT,
  `mp_SN` INT,
  `mp_mp_typ_name` INT,
  `mp_mp_typ_id` INT,
  `mp_aktiv` INT,
  `mp_aktiv_seit` INT,
  `mp_inaktiv_seit` INT,
  `bereich_name` INT,
  `raum_name` INT,
  `hub_id` INT,
  `mp_beacon_id` INT
);

-- -----------------------------------------------------
-- Placeholder table for view `ATSDB`.`raum_bereich`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `ATSDB`.`raum_bereich` (
  `raum_id` INT,
  `raum_name` INT,
  `raum_aktiv` INT,
  `raum_aktiv_seit` INT,
  `raum_inaktiv_seit` INT,
  `bereich_name` INT,
  `bereich_id` INT
);

-- -----------------------------------------------------
-- View `ATSDB`.`beacon_left_join_mp`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ATSDB`.`beacon_left_join_mp`;
USE `ATSDB`;
CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`ATSDB`.`beacon_left_join_mp` AS
SELECT `ATSDB`.`beacon`.`beacon_id` AS `beacon_id`,
       `ATSDB`.`beacon`.`beacon_hub_id` AS `beacon_hub_id`,
       `ATSDB`.`beacon`.`beacon_RSSI` AS `beacon_RSSI`,
       `ATSDB`.`beacon`.`beacon_timestamp` AS `beacon_timestamp`,
       `ATSDB`.`beacon`.`beacon_hub_ts_beginn` AS `beacon_hub_ts_beginn`,
       `ATSDB`.`beacon`.`beacon_batterie` AS `beacon_batterie`,
       `ATSDB`.`beacon`.`beacon_MAC` AS `beacon_MAC`,
       `ATSDB`.`mp`.`mp_mp_typ_id` AS `mp_mp_typ_id`
FROM (`ATSDB`.`beacon`
      LEFT JOIN `ATSDB`.`mp` ON (`ATSDB`.`mp`.`mp_beacon_id` =
`ATSDB`.`beacon`.`beacon_id`));

-- -----------------------------------------------------
-- View `ATSDB`.`beacon_mp`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `ATSDB`.`beacon_mp`;
```

```
347. USE `ATSDB`;
348. CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`ATSDB`.`beacon_mp` AS
349. SELECT `ATSDB`.`beacon`.`beacon_id` AS `beacon_id`,
350.        `ATSDB`.`beacon`.`beacon_MAC` AS `beacon_MAC`,
351.        `ATSDB`.`beacon`.`beacon_RSSI` AS `beacon_RSSI`,
352.        `ATSDB`.`beacon`.`beacon_batterie` AS `beacon_batterie`,
353.        `ATSDB`.`beacon`.`beacon_hub_id` AS `beacon_hub_id`,
354.        `ATSDB`.`beacon`.`beacon_timestamp` AS `beacon_timestamp`,
355.        `ATSDB`.`beacon`.`beacon_hub_ts_beginn` AS `beacon_hub_ts_beginn`,
356.        `ATSDB`.`mp`.`mp_id` AS `mp_id`,
357.        `ATSDB`.`mp`.`mp_name` AS `mp_name`,
358.        `ATSDB`.`mp`.`mp_mp_typ_id` AS `mp_mp_typ_id`
359. FROM (`ATSDB`.`beacon`
360.       JOIN `ATSDB`.`mp` ON (`ATSDB`.`mp`.`mp_beacon_id` = `ATSDB`.`beacon`.`beacon_id`));
361.
362. -- -----------------------------------------------------
363. -- View `ATSDB`.`beacon_raum_bereich`
364. -- -----------------------------------------------------
365. DROP TABLE IF EXISTS `ATSDB`.`beacon_raum_bereich`;
366. USE `ATSDB`;
367. CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`ATSDB`.`beacon_raum_bereich` AS
368. SELECT `ATSDB`.`beacon`.`beacon_id` AS `beacon_id`,
369.        `ATSDB`.`beacon`.`beacon_MAC` AS `beacon_MAC`,
370.        `ATSDB`.`beacon`.`beacon_aktiv` AS `beacon_aktiv`,
371.        `ATSDB`.`beacon`.`beacon_aktiv_seit` AS `beacon_aktiv_seit`,
372.        `ATSDB`.`beacon`.`beacon_inaktiv_seit` AS `beacon_inaktiv_seit`,
373.        `ATSDB`.`beacon`.`beacon_timestamp` AS `beacon_timestamp`,
374.        `ATSDB`.`beacon`.`beacon_hub_ts_beginn` AS `beacon_hub_ts_beginn`,
375.        `ATSDB`.`beacon`.`beacon_hub_id` AS `beacon_hub_id`,
376.        `hub_raum_bereich`.`raum_id` AS `raum_id`,
377.        `hub_raum_bereich`.`raum_name` AS `raum_name`,
378.        `hub_raum_bereich`.`raum_aktiv` AS `raum_aktiv`,
379.        `hub_raum_bereich`.`raum_aktiv_seit` AS `raum_aktiv_seit`,
380.        `hub_raum_bereich`.`raum_inaktiv_seit` AS `raum_inaktiv_seit`,
381.        `hub_raum_bereich`.`bereich_name` AS `bereich_name`,
382.        `hub_raum_bereich`.`bereich_id` AS `bereich_id`
383. FROM (`ATSDB`.`beacon`
384.       LEFT JOIN `ATSDB`.`hub_raum_bereich` ON (`hub_raum_bereich`.`hub_id` =
`ATSDB`.`beacon`.`beacon_hub_id`));
385.
386. -- -----------------------------------------------------
387. -- View `ATSDB`.`hub_raum_bereich`
388. -- -----------------------------------------------------
389. DROP TABLE IF EXISTS `ATSDB`.`hub_raum_bereich`;
390. USE `ATSDB`;
391. CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`ATSDB`.`hub_raum_bereich` AS
392. SELECT `ATSDB`.`hub`.`hub_id` AS `hub_id`,
393.        `ATSDB`.`hub`.`hub_MAC` AS `hub_MAC`,
394.        `ATSDB`.`hub`.`hub_aktiv` AS `hub_aktiv`,
395.        `ATSDB`.`hub`.`hub_aktiv_seit` AS `hub_aktiv_seit`,
396.        `ATSDB`.`hub`.`hub_inaktiv_seit` AS `hub_inaktiv_seit`,
397.        `ATSDB`.`hub`.`hub_timestamp` AS `hub_timestamp`,
398.        `ATSDB`.`hub`.`hub_raum_ts` AS `hub_raum_ts`,
399.        `raum_bereich`.`raum_id` AS `raum_id`,
400.        `raum_bereich`.`raum_name` AS `raum_name`,
401.        `raum_bereich`.`raum_aktiv` AS `raum_aktiv`,
402.        `raum_bereich`.`raum_aktiv_seit` AS `raum_aktiv_seit`,
403.        `raum_bereich`.`raum_inaktiv_seit` AS `raum_inaktiv_seit`,
404.        `raum_bereich`.`bereich_name` AS `bereich_name`,
405.        `raum_bereich`.`bereich_id` AS `bereich_id`
406. FROM (`ATSDB`.`hub`
407.       LEFT JOIN `ATSDB`.`raum_bereich` ON (`raum_bereich`.`raum_id` =
`ATSDB`.`hub`.`hub_raum_id`));
408.
409. -- -----------------------------------------------------
410. -- View `ATSDB`.`mapping_typ_namen`
411. -- -----------------------------------------------------
412. DROP TABLE IF EXISTS `ATSDB`.`mapping_typ_namen`;
```

```
413. USE `ATSDB`;
414. CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`ATSDB`.`mapping_typ_namen` AS
415. SELECT `ATSDB`.`mp_mapping`.`mp_mapping_id` AS `mp_mapping_id`,
416.        `mp_typ_1`.`mp_typ_name` AS `mp_typ_name_1`,
417.        `mp_typ_2`.`mp_typ_name` AS `mp_typ_name_2`,
418.        `mp_typ_1`.`mp_typ_id` AS `mp_typ_id_1`,
419.        `mp_typ_2`.`mp_typ_id` AS `mp_typ_id_2`
420. FROM ((`ATSDB`.`mp_mapping`
421.        JOIN `ATSDB`.`mp_typ` `mp_typ_1` ON (`ATSDB`.`mp_mapping`.`mp_mapping_mp_typ_id_1` =
`mp_typ_1`.`mp_typ_id`))
422.       JOIN `ATSDB`.`mp_typ` `mp_typ_2` ON (`ATSDB`.`mp_mapping`.`mp_mapping_mp_typ_id_2` =
`mp_typ_2`.`mp_typ_id`));
423.
424. -- -----------------------------------------------------
425. -- View `ATSDB`.`mp_raum_bereich`
426. -- -----------------------------------------------------
427. DROP TABLE IF EXISTS `ATSDB`.`mp_raum_bereich`;
428. USE `ATSDB`;
429. CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`ATSDB`.`mp_raum_bereich` AS
430. SELECT `ATSDB`.`mp`.`mp_id` AS `mp_id`,
431.        `ATSDB`.`mp`.`mp_name` AS `mp_name`,
432.        `ATSDB`.`mp`.`mp_SN` AS `mp_SN`,
433.        `ATSDB`.`mp_typ`.`mp_typ_name` AS `mp_mp_typ_name`,
434.        `ATSDB`.`mp_typ`.`mp_typ_id` AS `mp_mp_typ_id`,
435.        `ATSDB`.`mp`.`mp_aktiv` AS `mp_aktiv`,
436.        `ATSDB`.`mp`.`mp_aktiv_seit` AS `mp_aktiv_seit`,
437.        `ATSDB`.`mp`.`mp_inaktiv_seit` AS `mp_inaktiv_seit`,
438.        `ATSDB`.`bereich`.`bereich_name` AS `bereich_name`,
439.        `ATSDB`.`raum`.`raum_name` AS `raum_name`,
440.        `ATSDB`.`hub`.`hub_id` AS `hub_id`,
441.        `ATSDB`.`mp`.`mp_beacon_id` AS `mp_beacon_id`
442. FROM (((((`ATSDB`.`mp`
443.          JOIN `ATSDB`.`mp_typ` ON (`ATSDB`.`mp`.`mp_mp_typ_id` =
`ATSDB`.`mp_typ`.`mp_typ_id`))
444.         LEFT JOIN `ATSDB`.`beacon` ON (`ATSDB`.`mp`.`mp_beacon_id` =
`ATSDB`.`beacon`.`beacon_id`))
445.        LEFT JOIN `ATSDB`.`hub` ON (`ATSDB`.`beacon`.`beacon_hub_id` =
`ATSDB`.`hub`.`hub_id`))
446.       LEFT JOIN `ATSDB`.`raum` ON (`ATSDB`.`hub`.`hub_raum_id` = `ATSDB`.`raum`.`raum_id`))
447.      LEFT JOIN `ATSDB`.`bereich` ON (`ATSDB`.`raum`.`raum_bereich_id` =
`ATSDB`.`bereich`.`bereich_id`));
448.
449. -- -----------------------------------------------------
450. -- View `ATSDB`.`raum_bereich`
451. -- -----------------------------------------------------
452. DROP TABLE IF EXISTS `ATSDB`.`raum_bereich`;
453. USE `ATSDB`;
454. CREATE OR REPLACE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY DEFINER VIEW
`ATSDB`.`raum_bereich` AS
455. SELECT `ATSDB`.`raum`.`raum_id` AS `raum_id`,
456.        `ATSDB`.`raum`.`raum_name` AS `raum_name`,
457.        `ATSDB`.`raum`.`raum_aktiv` AS `raum_aktiv`,
458.        `ATSDB`.`raum`.`raum_aktiv_seit` AS `raum_aktiv_seit`,
459.        `ATSDB`.`raum`.`raum_inaktiv_seit` AS `raum_inaktiv_seit`,
460.        `ATSDB`.`bereich`.`bereich_name` AS `bereich_name`,
461.        `ATSDB`.`bereich`.`bereich_id` AS `bereich_id`
462. FROM (`ATSDB`.`raum`
463.       JOIN `ATSDB`.`bereich` ON (`ATSDB`.`bereich`.`bereich_id` =
`ATSDB`.`raum`.`raum_bereich_id`));
464.
465. -- -----------------------------------------------------
466. -- Trigger `ATSDB`.`hub_AFTER_UPDATE`
467. -- -----------------------------------------------------
468. DELIMITER $$
469. CREATE DEFINER=`root`@`localhost` TRIGGER `ATSDB`.`hub_AFTER_UPDATE`
470. AFTER UPDATE ON `ATSDB`.`hub`
471. FOR EACH ROW
472. BEGIN
473.   IF NOT ISNULL(OLD.hub_raum_id) THEN
```

```
474.     IF OLD.hub_raum_id != NEW.hub_raum_id AND NOT ISNULL(NEW.hub_raum_ts) THEN
475.       -- Trage Raumänderung in Historientabelle ein.
476.       INSERT INTO hub_historie (hub_historie_hub_id, hub_historie_raum_id,
hub_historie_raum_ts_start, hub_historie_raum_ts_end)
477.       VALUES (OLD.hub_id, OLD.hub_raum_id, OLD.hub_raum_ts, NEW.hub_raum_ts);
478.     END IF;
479.   END IF;
480. END$$
481. DELIMITER ;
482.
483. -- -----------------------------------------------------
484. -- Trigger `ATSDB`.`beacon_AFTER_UPDATE`
485. -- -----------------------------------------------------
486. DELIMITER $$
487. CREATE DEFINER=`root`@`localhost` TRIGGER `ATSDB`.`beacon_AFTER_UPDATE`
488. AFTER UPDATE ON `ATSDB`.`beacon`
489. FOR EACH ROW
490. BEGIN
491.   IF OLD.beacon_hub_id != NEW.beacon_hub_id THEN
492.     IF (SELECT mp_id FROM mp WHERE mp_beacon_id = OLD.beacon_id) IS NOT NULL THEN
493.       -- Trage Beacon-Änderung in Historientabelle ein.
494.       INSERT INTO mp_historie (mp_historie_raum_id, mp_historie_mp_id,
mp_historie_raum_ts_start, mp_historie_raum_ts_end)
495.       VALUES ((SELECT hub_raum_id FROM hub WHERE hub_id = OLD.beacon_hub_id), (SELECT mp_id
FROM mp WHERE mp_beacon_id = OLD.beacon_id), OLD.beacon_hub_ts_beginn, NEW.beacon_hub_ts_beginn);
496.     END IF;
497.   END IF;
498. END$$
499. DELIMITER ;
500.
501. -- -----------------------------------------------------
502. -- Trigger `ATSDB`.`beaconpair_AFTER_INSERT`
503. -- -----------------------------------------------------
504. DELIMITER $$
505. CREATE DEFINER=`root`@`localhost` TRIGGER `ATSDB`.`beaconpair_AFTER_INSERT`
506. AFTER INSERT ON `ATSDB`.`beaconpair`
507. FOR EACH ROW
508. BEGIN
509.   -- Implement logic here if needed
510. END$$
511. DELIMITER ;
512.
513. -- Restore SQL modes
514. SET SQL_MODE=@OLD_SQL_MODE;
515. SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
516. SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
517.
```

A10 – Python Microservice

```
1. import paho.mqtt.client as mqtt  # für die MQTT Verbindung
2. import mysql.connector  # für die Datenbank-Verbindung
3. import time  # zur Nutzung von Methoden zur Zeitberechnung
4. import json  # zum einfachen Separieren der MQTT Nachricht
5. from pymemcache.client import base  # eine Bibliothek für serverseitigen memcache
6. import sys  # um Fehlercode bei einem Abbruch zurückzugeben
7. import os
8. from dotenv import load_dotenv, find_dotenv
9.
10. # Laden der ungebungsabhängigen Konfigurationsparameter
11. load_dotenv(find_dotenv('Config.env'))
12. # Konfiguration der MySQL-Datenbankverbindung, des Memcached und des MQTT
13. db_config = {
14.     'host': os.getenv('DB_HOST'),
15.     'port': int(os.getenv('DB_PORT')),
16.     'user': os.getenv('DB_USER'),
17.     'password': os.getenv('DB_PASSWORD'),
18.     'database': os.getenv('DB_DATABASE')
19. }
20. db_connection = None    # Globale Variable zum Halten des DB connectors
21. # MQTT Broker Konfiguration
22. memcache_server = os.getenv('MEMCACHE_SERVER')
23. memcache_port = int(os.getenv('MEMCACHE_PORT'))
24. mqtt_server = os.getenv('MQTT_SERVER')
25. mqtt_port = int(os.getenv('MQTT_PORT'))
26. mqtt_topic = os.getenv('MQTT_TOPIC')
27. #client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1)
28. client = mqtt.Client()
29. client.username_pw_set(os.getenv('MQTT_USER'), os.getenv('MQTT_PW'))
30. # Raumwechsel bei niedrigerem RSSI nach Zeitlücke
31. room_timegap = int(os.getenv('ROOM_TIMEGAP'))
32. pairing_timegap = int(os.getenv('TIMEGAP'))
33. db_update_cycle_beacon = int(os.getenv('DB_UPDATE_CYCLE_BEACON'))
34. db_update_cycle_hub = int(os.getenv('DB_UPDATE_CYCLE_HUB'))
35.
36. # Debug Level und Filter Funktion, Debug Meldungen aus dem Text können bitweise auf Leveln
       definiert werden
37. debug_level = int(os.getenv('DEBUG_LEVEL'))
38. debug_count = 0
39. def do_debug(k):
40.     temp = debug_level >> (k - 1)
41.     return temp & 1 != 0
42.
43. # Serializer zur Memcache Nutzung, erleichten den Zugriff (Strings/Arrays statt bytes).
44. def json_serializer(key, value):
45.     return json.dumps(value), 1
46. def json_deserializer(key, value, flags):
47.     if flags == 1:
48.         return json.loads(value)
49.     raise Exception("Unknown serialization format")
50.
51. # Verbindung zur Datenbank herstellen. Hier als Methode, da
52. def connect_to_db():
53.     try:
54.         return mysql.connector.connect(**db_config)
55.     except mysql.connector.Error as err:
56.         print(f"Error connecting to MySQL: {err}")
57.         return None
58.
59. # Optimierungsbedarf: Globale Datenbankverbindung, einmal geöffnet für die Laufzeit des
       Programms
60. db_connection = connect_to_db()
61.
62. # Aufbau der globalen Verbindung zum memcached
63. memcache = base.Client((memcache_server, memcache_port), serializer=json_serializer,
                           deserializer=json_deserializer)
64.
65. # Initalisieren von relevanten Daten bei Programmstart aus der Maria-DB
```

```
66. def load_initial_data():
67.     # Initialisieren von Beacondaten mit und ohne MP
68.     try:
69.         cursor = db_connection.cursor(dictionary=True)
70.         query = ''' SELECT * from beacon_left_join_mp '''
71.         cursor.execute(query)
72.         beacon_mp_cache = cursor.fetchall()
73.         for data_values in beacon_mp_cache:
74.             if isinstance(data_values, dict):
75.                 beacon_id = data_values['beacon_id']
76.                 beacon_hub_id = data_values['beacon_hub_id']
77.                 beacon_rssi = data_values['beacon_RSSI']
78.                 beacon_timestamp = data_values['beacon_timestamp']
79.                 beacon_hub_ts_beginn = data_values['beacon_hub_ts_beginn']
80.                 beacon_batterie = data_values['beacon_batterie']
81.                 mp_mp_typ_id = data_values['mp_mp_typ_id']  # Null, falls noch nicht
                                                                zugeordnet
82.                 beacon_MAC = data_values['beacon_MAC']
83.                 beacon_db_sync_timestamp = data_values['beacon_timestamp']
84.                 beacon_neudaten = (beacon_id, beacon_hub_id, beacon_rssi, beacon_timestamp,
                    beacon_hub_ts_beginn, beacon_batterie, mp_mp_typ_id,
                    beacon_db_sync_timestamp)
85.                 # Falls neu oder aktueller als im Memcache, dort aktualisieren
86.                 if memcache.get(beacon_MAC) is None or memcache.get(beacon_MAC)[7] <=
                       beacon_timestamp:
87.                     beacon_initial_anlegen(beacon_MAC, beacon_neudaten)
88.                 else:
89.                     beacon_aktualisieren(beacon_MAC, beacon_neudaten)
90.             else:
91.                 print(f"Unexpected data format: {data_values}")
92.     except Exception as e:
93.         print(f"An error occurred: {e}")
94.
95.     # Initialisieren von zulässigen MP-Verbindungen (für Beaconpaare relevant)
96.     try:
97.         cursor = db_connection.cursor(dictionary=True)
98.         query = ''' SELECT * from mp_mapping '''
99.         cursor.execute(query)
100.        mp_mapping_cache = cursor.fetchall()
101.        # Aktualisieren des Memcache für alle Paarungen
102.        for data_values in mp_mapping_cache:
103.            if isinstance(data_values, dict):
104.                mp_typ_id1 = data_values['mp_mapping_mp_typ_id_1']
105.                mp_typ_id2 = data_values['mp_mapping_mp_typ_id_2']
106.                update_mp_typ_mapping(mp_typ_id1, mp_typ_id2)
107.            else:
108.                print(f"Unexpected data format: {data_values}")
109.    except mysql.connector.Error as err:
110.        print(f"Error loading initial data: {err}")
111.
112.    # Initialisieren von bereits vorhandenen Beaconpaaren
113.    try:
114.        cursor = db_connection.cursor(dictionary=True)
115.        query = ''' SELECT * FROM beaconpair '''
116.        cursor.execute(query)
117.        mp_beaconpair_cache = cursor.fetchall()
118.        # Aktualisieren des Memcache für alle Paarungen
119.        for data_values in mp_beaconpair_cache:
120.            if isinstance(data_values, dict):
121.                beaconpair_beacon_id_1 = int(data_values['beaconpair_beacon_id_1'])
122.                beaconpair_beacon_id_2 = int(data_values['beaconpair_beacon_id_2'])
123.                update_beaconpairs(beaconpair_beacon_id_1, beaconpair_beacon_id_2)
124.            else:
125.                print(f"Unexpected data format: {data_values}")
126.    except mysql.connector.Error as err:
127.        print(f"Error loading initial data: {err}")
128.
129. def beacon_initial_anlegen(beacon_MAC, beacon_neudaten):
130.     memcache.set(beacon_MAC, beacon_neudaten)
131.     return None
132.
```

```
133. # Befüllen des Memcache mit zulässigen Mappings (aus der Datenbank)
134. # Notation: mp_typ_mapping_mp_typ_ID: [MappedMpID1, MappedMpID2, MappedMpID1...]
135. def update_mp_typ_mapping(mp_typ_id_1, mp_typ_id_2):
136.     # Eintrag in der ersten Richtung
137.     # Holen bestehender Mappings für Typ 1
138.     mp_typ_mapping_key = f'mp_typ_mapping_{mp_typ_id_1}'
139.     mp_typ_mapping_temp = memcache.get(mp_typ_mapping_key)
140.     if mp_typ_mapping_temp is None: mp_typ_mapping_temp = []
141.     # Hinzufügen der neuen Erlaubnis und Speichern
142.     if mp_typ_id_2 not in mp_typ_mapping_temp:
143.         mp_typ_mapping_temp.append(mp_typ_id_2)
144.         memcache.set(mp_typ_mapping_key, mp_typ_mapping_temp)
145.     if do_debug(2): print(f"Mapping Data von {mp_typ_id_1} in Memcache:
                              {memcache.get(mp_typ_mapping_key)}")
146.
147.     # Eintrag in der zweiten Richtung
148.     mp_typ_mapping_key = f'mp_typ_mapping_{mp_typ_id_2}'
149.     mp_typ_mapping_temp = memcache.get(mp_typ_mapping_key)
150.     if mp_typ_mapping_temp is None: mp_typ_mapping_temp = []
151.     if mp_typ_id_1 not in mp_typ_mapping_temp:
152.         mp_typ_mapping_temp.append(mp_typ_id_1)
153.         memcache.set(mp_typ_mapping_key, mp_typ_mapping_temp)
154.     if do_debug(2): print(f"Mapping Data von {mp_typ_id_2} in Memcache:
                              {memcache.get(mp_typ_mapping_key)}")
155.
156. # Funktion um ein Beaconpaar einzutragen
157. # Notation: beaconpairs_BeaconID: [MappedBeaconID1, MappendBeacon_ID2, MappedBeaconID...]
158. def update_beaconpairs(beacon_id_1, beacon_id_2):
159.     # Für jedes Beacon die jeweils andere ID in den Pairs hinzufügen
160.     beaconpairs_key = f'beaconpairs_{beacon_id_1}'
161.     # Bestehende Paarungen holen.
162.     beaconpairs_temp = memcache.get(beaconpairs_key)
163.     if beaconpairs_temp is None: beaconpairs_temp = []
164.     # Neue ID anhängen und Speichern
165.     if beacon_id_2 not in beaconpairs_temp: beaconpairs_temp.append(beacon_id_2)
166.     memcache.set(beaconpairs_key, beaconpairs_temp)
167.     # Für zweite ID entsprechend
168.     beaconpairs_key = f'beaconpairs_{beacon_id_2}'
169.     beaconpairs_temp = memcache.get(beaconpairs_key)
170.     if beaconpairs_temp is None: beaconpairs_temp = []
171.     if beacon_id_1 not in beaconpairs_temp: beaconpairs_temp.append(beacon_id_1)
172.     memcache.set(beaconpairs_key, beaconpairs_temp)
173.
174.     # Hinzufügen der Beaconpaare in Array für Crawler
175.     tmp_pair_1 = f"{beacon_id_1},{beacon_id_2}"
176.     tmp_pair_2 = f"{beacon_id_2},{beacon_id_1}"
177.     tmp_beaconpairs = memcache.get('beaconpairs')
178.     # Gibt es noch keine Paare? Dann leer.
179.     if tmp_beaconpairs is None: tmp_beaconpairs = []
180.     # Nur hinzufügen, wenn nicht schon existent
181.     if tmp_pair_1 not in tmp_beaconpairs and tmp_pair_2 not in tmp_beaconpairs:
182.         tmp_beaconpairs.append(f"{beacon_id_1},{beacon_id_2}")
183.         memcache.set('beaconpairs', tmp_beaconpairs)
184.     if do_debug(2): print(f"Beaconpair Data von {beacon_id_2} in Memcache:
                              {memcache.get(beaconpairs_key)}")
185.
186. # Hub-Zuordnung zu Beacon in DB und Memcache aktualisieren
187. def hub_aktualisieren(hub_MAC, timestamp):
188.     hub_data = memcache.get(hub_MAC)  # Suche im Cache
189.     if hub_data is not None:  # [0]: id, [1]: ts, [2]: ts des letzten updates der Datenbank
190.         if do_debug(2): print("Aktualisierung Hub: Bereits im Cache: ",
                                  memcache.get(hub_MAC))
191.         hub_id = hub_data[0]
192.         # Hub Timestamp in DB aktualisieren, dies dient der Nachvollziehbarkeit, dass der
               Hub noch aktiv ist
193.         if hub_data[2] + db_update_cycle_hub < timestamp:
194.             if do_debug(4): print("Aktualisierung Hub: Bereits im Cache, aber DB Eintrag zu
                                       alt: ", memcache.get(hub_MAC))
195.             try:
196.                 my_cursor = db_connection.cursor()
```

```
197.                 my_query = 'update hub set hub_timestamp = %(ts)s where hub_id = "%(id)s"' %
                                {'ts': timestamp, 'id': hub_id}
198.                 my_cursor.execute(my_query)
199.                 db_connection.commit()
200.                 memcache.set(hub_MAC, [hub_id, timestamp, timestamp])
201.             except mysql.connector.Error as err:
202.                 print(f"Error updating hub in MySQL: {err}")
203.         else:
204.             if hub_data[1] < timestamp:
205.                 memcache.set(hub_MAC, [hub_id, timestamp, hub_data[2]])
206.                 if do_debug(2): print("Aktualisierung Hub: Bereits im Cache, aber neuer TS:
                                      ", memcache.get(hub_MAC))
207.     else:
208.         # Hub neu/noch nicht im Cache
209.         try:
210.             my_cursor = db_connection.cursor()
211.             my_query = 'Select hub_id, hub_timestamp from hub where hub_MAC = "%(hub)s"' %
                            {'hub': hub_MAC}
212.             my_cursor.execute(my_query)
213.             my_result = my_cursor.fetchone()
214.             if my_result is not None:  # hub gefunden, im memcache mit aktuellem ts
eintragen
215.                 hub_id = my_result[0]
216.                 memcache.set(hub_MAC, [hub_id, timestamp, timestamp])
217.             if do_debug(2): print("Hub noch nicht im Cache. Cache geladen mit: ",
                                    memcache.get(hub_MAC))
218.         except mysql.connector.Error as err:
219.             print(f"Error querying hub from MySQL: {err}")
220.     return hub_id
221.
222. # Holen der zuletzt bekannten Beacondaten. Aus Memcache oder bei nach Start des Programms
      hinzugekommenen Beacons aus der Datenbank.
223. def beacon_altdaten_holen(beacon_MAC, timestamp):
224.     beacon_altdaten = memcache.get(beacon_MAC)
225.     if do_debug(2): print("beacon-Altdaten im cache: ", beacon_altdaten)
226.     if beacon_altdaten is None:
227.         try:
228.             my_cursor = db_connection.cursor()
229.             my_query = 'SELECT beacon_id, beacon_hub_id, beacon_RSSI, beacon_timestamp,
                            beacon_hub_ts_beginn, beacon_batterie FROM beacon where beacon_MAC =
                            "%(beacon)s"' % {'beacon': beacon_MAC}
230.             my_cursor.execute(my_query)
231.             beacon_altdaten = my_cursor.fetchone()
232.             if beacon_altdaten is not None:
233.                 # In der Datenbank existiert noch kein Timestamp, zur weiteren Verarbeitung
                      ist er erforderlich
234.                 beacon_altdaten = beacon_altdaten + (timestamp,)
235.                 memcache.set(beacon_MAC, beacon_altdaten)
236.             if do_debug(2): print("beacon-Altdaten im cache: ", beacon_altdaten)
237.         except mysql.connector.Error as err:
238.             print(f"Error querying beacon from MySQL: {err}")
239.     return beacon_altdaten
240.
241. # Funktion, die einem existenten Beacon zum ersten Mal einem Hub zuweist.
242. def beacon_erstspeicherung(beacon_MAC, beacon_neudaten):
243.     if do_debug(2): print("beacon Neudaten speichern (Erstzuweisung)")
244.     try:
245.         my_cursor = db_connection.cursor()
246.         my_query = 'update beacon set beacon_hub_id = "%(hub_id)s", beacon_RSSI =
                        "%(rssi)s", beacon_timestamp = "%(ts)s", beacon_hub_ts_beginn =
                        "%(ts)s", beacon_batterie = "%(bat)s" where beacon_id = "%(id)s"' % {
247.             'hub_id': beacon_neudaten[1], 'rssi': beacon_neudaten[2], 'ts':
                            beacon_neudaten[3], 'id': beacon_neudaten[0], 'bat':
                            beacon_neudaten[5]}
248.         my_cursor.execute(my_query)
249.         db_connection.commit()
250.         memcache.set(beacon_MAC, beacon_neudaten)
251.     except mysql.connector.Error as err:
252.         print(f"Error inserting new beacon data into MySQL: {err}")
253.
254. # Beacondaten im Memcache und bei Bedarf (z.B. letzter Eintrag mehr als 10 Minuten alt)
```

```
255. # auch in der DB aktualisieren. Es hat aber kein Hubwechsel stattgefunden.
256. def beacon_aktualisieren(beacon_MAC, beacon_neudaten):
257.     if do_debug(2): print("beacon im Vergleich speichern")
258.     if beacon_neudaten[7] + db_update_cycle_beacon < beacon_neudaten[3]:
259.         if do_debug(2): print("beacon Update in DB speichern (Timer)")
260.         try:
261.             my_cursor = db_connection.cursor()
262.             my_query = 'update beacon set beacon_hub_id = "%(hub_id)s", beacon_RSSI =
                            "%(rssi)s", beacon_timestamp = "%(ts)s", beacon_hub_ts_beginn =
                            "%(ts)s", beacon_batterie = "%(bat)s" where beacon_id = "%(id)s"' % {
263.                 'hub_id': beacon_neudaten[1], 'rssi': beacon_neudaten[2], 'ts':
                              beacon_neudaten[3], 'id': beacon_neudaten[0], 'bat':
                              beacon_neudaten[5]}
264.             my_cursor.execute(my_query)
265.             db_connection.commit()
266.             beacon_neudaten[7] = beacon_neudaten[3]
267.         except mysql.connector.Error as err:
268.             print(f"Error updating beacon data in MySQL: {err}")
269.     memcache.set(beacon_MAC, beacon_neudaten)
270.     if do_debug(2): print("beacon im Cache/in DB aktualisiert.")
271.
272. # Hubwechsel. Beacondaten im Memcache und der DB aktualisieren.
273. def beacon_hubwechsel(beacon_MAC, beacon_neudaten):
274.     if do_debug(2): print("beacon zu anderem Hub wechseln (DB und Cache)")
275.     try:
276.         my_cursor = db_connection.cursor()
277.         my_query = 'update beacon set beacon_hub_id = "%(hub_id)s", beacon_RSSI =
                        "%(rssi)s", beacon_timestamp = "%(ts)s", beacon_hub_ts_beginn =
                        "%(ts)s", beacon_batterie = "%(bat)s" where beacon_id = "%(id)s"' % {
278.             'hub_id': beacon_neudaten[1], 'rssi': beacon_neudaten[2], 'ts':
                          beacon_neudaten[3], 'id': beacon_neudaten[0], 'bat':
                          beacon_neudaten[5]}
279.         my_cursor.execute(my_query)
280.         db_connection.commit()
281.         memcache.set(beacon_MAC, beacon_neudaten)
282.         if do_debug(2): print("beacon im Cache/in DB aktualisiert.")
283.     except mysql.connector.Error as err:
284.         print(f"Error updating beacon hub data in MySQL: {err}")
285.
286. # Neues Beaconpair in die Datenbank eintragen
287. def beaconpairing_DB_insert(beacon_id_1, beacon_id_2, hub_id, timestamp):
288.     try:
289.         my_cursor = db_connection.cursor()
290.         insert_query = '''
291.             INSERT INTO beaconpair (beaconpair_beacon_id_1, beaconpair_beacon_id_2,
                            beaconpair_timestamp, beaconpair_hub_id)
292.             VALUES (%(beacon_id_1)s, %(beacon_id_2)s, %(timestamp)s, %(hub_id)s)
293.             '''
294.         insert_data = {
295.             'beacon_id_1': min(int(beacon_id_1), int(beacon_id_2)),
296.             'beacon_id_2': max(int(beacon_id_1), int(beacon_id_2)),
297.             'timestamp': timestamp,
298.             'hub_id': hub_id,
299.             }
300.         my_cursor.execute(insert_query, insert_data)
301.         db_connection.commit()
302.         if do_debug(2): print("Eintrag in die DB-Tabelle 'beaconpair' erfolgt oder
                                   aktualisiert")
303.     except mysql.connector.Error as err:
304.         print(f"Error handling beacon pairing data in MySQL: {err}")
305.
306. # Funktion, die zwei zulässige Beacons miteinander paart und die Paarung im Memcache und der
       DB einträgt
307. # Notation des Caches: beaconpairing_cache_hub_id_mp_typ: {beacon_id , timestamp}
308. def beaconpairing(hub_id, mp_typ_id, beacon_id, timestamp):
309.     try:
310.         beacon_mapping_mp_typen = memcache.get(f"mp_typ_mapping_{mp_typ_id}")
311.         # Darf dieses Beacon mit anderen Typen gepaart werden?
312.         if beacon_mapping_mp_typen is not None:
313.             array_length = len(beacon_mapping_mp_typen)
314.             i = 0
```

```
315.             # Für jeden Typen nach Partnern suchen, die auf ein Paaren warten
316.             while i < array_length:
317.                 beaconpairing_temp_key =
                     f"beaconpairing_cache_{hub_id}_{beacon_mapping_mp_typen[i]}"
318.                 moeglicher_partner = memcache.get(beaconpairing_temp_key)
319.                 # Wenn Partner gefunden und Anfrage noch nicht zu alt
320.                 if moeglicher_partner is not None:
321.                     if timestamp - moeglicher_partner[1] <= pairing_timegap:
322.                         # Eintragen des Paares in Memcached und Datenbank, Löschen der
                              temporären Paarungsanfrage
323.                         update_beaconpairs(beacon_id, moeglicher_partner[0])
324.                         beaconpairing_DB_insert(beacon_id, moeglicher_partner[0], hub_id,
                             timestamp)
325.                         memcache.delete(beaconpairing_temp_key)
326.                         return
327.                     else:
328.                         print("Fehler bei Löschen von Cachedaten älter " +
                                  str(pairing_timegap) + " Sekunden")
329.                         return
330.                 i += 1
331.             # Wenn Funktion bis hierhin kein Return ausgelöst hat, wurde kein Beacon zum
                  Mapping gefunden.
332.             # Also folgt ein Eintrag in den Cache, der pairing_timegap Sekunden gespeichert
                  wird
333.             # Der Timestamp ist lediglich eine Sicherheitsmaßnahme
334.             beaconpairing_temp_key = f"beaconpairing_cache_{hub_id}_{mp_typ_id}"
335.             beaconpairing_temp_value = (beacon_id, timestamp)
336.             memcache.set(beaconpairing_temp_key, beaconpairing_temp_value, pairing_timegap)
337.         else:
338.             if do_debug(2):
339.                 print("Zu diesem MP gibt es keinen zugelassenen Partner")
340.             return
341.
342.     except Exception as e:
343.         print(f"An error occurred: {e}")
344.
345.
346. # Funktion, die ein Beaconpaar als 'kritisch' markiert, wenn einer der Beacons den Hub/Raum
      gewechselt hat
347. def beacon_pairing_hubwechsel(beacon_id, new_hub_id, timestamp):
348.     try:
349.         beaconpair_key = f'beaconpairs_{beacon_id}'
350.         beaconpairs = memcache.get(beaconpair_key)
351.         # Gibt es Beacon-Paare für das zu prüfende Beacon? Wenn nein: Ende.
352.         if not beaconpairs:
353.             if do_debug(2): print(f"Keine Mappings für Beacon ID {beacon_id} gefunden.")
354.             return
355.         # Für jede ID, die in den destehenden Paarungen angegeben is, eine Prüfung
              durchführen
356.         for mapped_beacon_id in beaconpairs:
357.             # Vorbereitung der Key-Namen, Holen der Paarungen zu den IDs, Prüfen
358.             beaconpair_krit_key_1 = f'beacon_mapping_krit_{mapped_beacon_id}_{beacon_id}'
359.             beaconpair_krit_key_2 = f'beacon_mapping_krit_{beacon_id}_{mapped_beacon_id}'
360.             beaconpair_krit_data_1 = memcache.get(beaconpair_krit_key_1)
361.             beaconpair_krit_data_2 = memcache.get(beaconpair_krit_key_2)
362.             if beaconpair_krit_data_1:
363.                 # Wenn auch das Paarungsziel auf neuem Hub bekannt, kritischen Eintrag
                      löschen
364.                 if beaconpair_krit_data_1[1] == new_hub_id:
365.                     memcache.delete(beaconpair_krit_key_1)
366.                     memcache.delete(beaconpair_krit_key_2)
367.                     if do_debug(2):
368.                         print(f"Gefährdungseintrag {beaconpair_krit_key_1} und
                                  {beaconpair_krit_key_2} gelöscht.")
369.                 else:
370.                     # Timestamp des bestehenden Eintrags aktualisieren
371.                     memcache.set(beaconpair_krit_key_2, [timestamp, new_hub_id])
372.                     if do_debug(2): print(f"Gefährdungseintrag {beaconpair_krit_key_2}
                                              aktualisiert.")
373.             elif beaconpair_krit_data_2:
374.                 if beaconpair_krit_data_2[1] == new_hub_id:
```

```
                    memcache.delete(beaconpair_krit_key_1)
                    memcache.delete(beaconpair_krit_key_2)
                    if do_debug(2):
                        print(f"Gefährdungseintrag {beaconpair_krit_key_1} und
                              {beaconpair_krit_key_2} gelöscht.")
                else:
                    memcache.set(beaconpair_krit_key_1, [timestamp, new_hub_id])
                    if do_debug(2): print(f"Gefährdungseintrag {beaconpair_krit_key_1}
                                          aktualisiert.")
            else:
                # Es gibt noch keinen Gefährdungseintrag. Erster Beacon einer Paarung, der
                  einen anderen Raum meldet.
                memcache.set(beaconpair_krit_key_1, [timestamp, new_hub_id])
                if do_debug(2): print(f"Neuer Gefährdungseintrag {beaconpair_krit_key_1}
                                      erstellt.")
    except Exception as e:
        print(f"An error occurred: {e}")

# Vorverarbeiten der MQTT Nachricht: Zerlegen in die einzelnen Werte, Weitergabe an die
  Verarbeitung, Laufzeitmessung
def process_message(message):
    global debug_count
    start_time = time.time() # Festhalten des Starts der Verarbeitung zur Laufzeitmessung
    # Startparameter und Positionen von Werten in den Memcached Datenstrukturen
    hub_id = 0  # 0 = kein Hub gefunden. -> keine weitere Verarbeitung.
    beacon_altdaten = None  # None = keine Altdaten/ kein Beacon Datensatz gefunden
    index_beacon_id = 0  # Ab hier: Positionen der einzelnen Werte innerhalb eines
                           gespeicherten Arrays rsp. der Datenbankrückmeldung
    index_beacon_hub_id = 1  # Position der Hub ID im Array
    index_beacon_rssi = 2  # Position des RSSI Wertes im Array
    index_beacon_timestamp = 3  # Position des Timestamp im Array
    index_beacon_hub_timestamp_beginn = 4  # Position der Erstmeldung des beacons am Hub
    index_beacon_batterie = 5  # Position des Batterie-Werts, nur von Relevanz bei
                                 Aktualisierung des beacons
    index_beacon_mp_typ = 6 # Position des dem zugeordneten MP eigenen MP Typs
    index_beacon_db_sync_timestamp = 7  # Position des Zeitpunktes, zu dem der letzte DB-
                                          Abgleich erfolgte

    # Verarbeiten der Nachricht
    try:
        # Konvertierung des JSON-Strings in ein Dictionary
        message_dict = json.loads(message)
        # Direkter Zugriff auf den Zeitpunkt, zu dem die Nachricht im MQTT angekommen ist
        timestamp = int(message_dict["time"])
        # Verarbeitung des data-Strings: Auslesen und den Text in Einzelwerte aufteilen
        data_parts = message_dict["data"].split(", ")
        data_values = {part.split('=')[0]: part.split('=')[1] for part in data_parts}
        # Direkter Zugriff auf die Werte der MQTT Nachricht
        hub_MAC = data_values.get('MAC_ROOM', None)
        beacon_MAC = data_values.get('MAC_SENSOR', None)
        beacon_batterie = int(data_values.get('BATT', 0))
        beacon_taster = int(data_values.get('BUTTON', 0))
        beacon_rssi = int(data_values.get('RSSI', 0))

        # Verarbeitet wird nur bei gültigem Hub!
        # Hub-aktiv-Timestamp aktualisieren. Wenn es den Hub gibt, ist nach Aufruf die
          hub_id > 0.
        if hub_MAC is not None:
            hub_id = hub_aktualisieren(hub_MAC, timestamp)
        # Wenn es den Hub gibt: Holen der letzten Meldungsdaten des beacons. Wenn es den
          beacon gibt, ist nach Aufruf die beacon_id > 0
        if hub_id > 0 and beacon_MAC is not None:
            beacon_altdaten = beacon_altdaten_holen(beacon_MAC, timestamp)
        else:
            if do_debug(4): print("Unberkannte Beacondaten: ", data_parts)
        # Wenn es den beacon und Hub in der Datenbank gibt, verarbeite die neuen Daten
        if beacon_altdaten is not None:
            if do_debug(2): print("beacon Altdaten: ", beacon_altdaten)
            # Wenn Timestamp der neuen Daten noch nicht im Cache, akualisiere,
            # sonst ignorieren, da Nachrict des Beacons vor Reconnect schon prozessiert
              wurde
```

```
435.             if timestamp != beacon_altdaten[index_beacon_timestamp]:
436.                 # Wenn noch kein Hub zugewiesen, wurde ist noch kein Zeitstempel der
                     Zuweisung in den Altdaten, daher aktuellen TS nutzen
437.                 # Wenn schon ein Hub zugewiesen wurde, aber ein Datensatz mit anderm Hub und
                     höherem RSSI kommt, ebenfalls aktuellen TS nutzen
438.                 # Ansonsten Timestamp der Erstverbindung weiterführen, ebenso der letzten
                     Datenbanksynchronisation
439.                 if beacon_altdaten[index_beacon_hub_id] is None:
440.                     # Case 1: Beacon wurde noch nie in einem Hub gefunden. Daten eintragen.
441.                     beacon_neudaten = [beacon_altdaten[index_beacon_id], hub_id,
                         beacon_rssi, timestamp, timestamp, \
442.                                        beacon_batterie,
                        beacon_altdaten[index_beacon_mp_typ], timestamp]
443.                     beacon_erstspeicherung(beacon_MAC, beacon_neudaten)
444.                     if do_debug(2): print("Ersteintrag Beacon erfolgt.")
445.                     if do_debug(2): print("Beacon Neudaten sind: ", beacon_neudaten)
446.                 else:
447.                     if beacon_altdaten[index_beacon_hub_id] == hub_id:
448.                         # Case 2: Hub gleich? Dann RSSI, Batterie und Zeit des beacons
                             aktualisieren.
449.                         # Timestamp der Erstverbindung halten, ebenso der letzten DB-
                             Synchronisation
450.                         beacon_neudaten = [beacon_altdaten[index_beacon_id], hub_id,
                            beacon_rssi, timestamp, \
451.                         beacon_altdaten[index_beacon_hub_timestamp_beginn],
                            beacon_batterie,\
452.                         beacon_altdaten[index_beacon_mp_typ], \
453.                         beacon_altdaten[index_beacon_db_sync_timestamp]]
454.                         beacon_aktualisieren(beacon_MAC, beacon_neudaten)
455.                         if do_debug(2): print("Beacon aktualisiert")
456.                         if do_debug(2): print("Beacon Neudaten sind: ", beacon_neudaten)
457.                         # Taster gedrückt? Aufforderung zur Verknüpfung.
458.                         if beacon_taster == 1:
459.                             if do_debug(2): print("Taster wurde gedrückt, Beaconpairing
                                                      eingeleitet")
460.                             beaconpairing(hub_id,beacon_altdaten[index_beacon_mp_typ]
                                ,beacon_altdaten[index_beacon_id] , timestamp)
461.                     else:
462.                         # Case 3: Wenn neuer Hub mit besserem RSSI
463.                         # oder neuer Hub, aber alter Hub-Eintrag älter als 5 Minuten,
                             aktualisiere Hub.
464.                         if (beacon_altdaten[index_beacon_rssi] < beacon_rssi) or \
465.                             (beacon_altdaten[index_beacon_timestamp] + room_timegap <
                             timestamp):
466.                             beacon_neudaten = [beacon_altdaten[index_beacon_id], hub_id,
                             beacon_rssi, \
467.                             timestamp, timestamp, beacon_batterie, \
468.                             beacon_altdaten[index_beacon_mp_typ], timestamp]
469.                             beacon_hubwechsel(beacon_MAC, beacon_neudaten)
470.                             # Überprüfung von Beaconpaaren auf vorliegenden Standortwechsel
471.                             beacon_pairing_hubwechsel(beacon_altdaten[index_beacon_id],
                             hub_id, timestamp)
472.                             if do_debug(4): print("beacon hat den Hub gewechselt")
473.                             if do_debug(4): print("beacon Neudaten sind: ", beacon_neudaten)
474.         else:
475.             if do_debug(4): print("Unberkannte Beacondaten: ", data_parts)
476.         if do_debug(2): print(timestamp, data_values)
477.         if do_debug(2): # Messen der Laufzeit einer Verarbeitung
478.             debug_count += 1
479.             print("Extraktion und Schicken der Nachricht ", "{0:05d}".format(debug_count), "
                     hat %.5f" % (time.time() - start_time), " Sekunden gedauert")
480.     except Exception as e:
481.         print(f"Fehler bei der Verarbeitung der Nachricht: {e}") # ohne debug-filter, das
                                                                      darf nicht passieren.
482.
483. # MQTT methoden zum Bedienen der connect, disconnect und message events der MQTT Verbindung
484. # on_mqtt_message überprüft die Datenbankverbindung und gibt die empfangene Nachricht zur
      Verarbeitung.
485. # on_mqtt_connect registriert den Client zum Horchen auf das definierte Topic
486. # on_mqtt_disconnect sorgt bei einem Wegfallen der MQTT Verbindung für einen erneuten
      Verbindungsaufbau, sowie MQTT wieder zur Verfügung steht
```

```
487. def on_mqtt_message(client, userdata, msg):
488.     global db_connection
489.     # Sicherstellen, dass die Datenbankverbindung noch steht, Neuaufbau falls nicht
490.     loop = 0
491.     while db_connection.is_connected() == False:
492.         try:
493.             db_connection = mysql.connector.connect(**db_config)
494.         except mysql.connector.Error as err:
495.             time.sleep(5)
496.         loop = loop + 1
497.         if loop > 50:
498.             exit(51)    # Beendigen des Programms nach 50 erfolglosen Verbindungsversuchen.
499.     # Verbindung zur Datenbank steht, die Verarbeitung der Nachricht kann starten
500.     message = msg.payload.decode()
501.     process_message(message)
502.
503. def on_mqtt_connect(client, userdata, flags, rc):
504.     if rc == 0:
505.         if do_debug(2): print("Mit MQTT-Broker verbunden.")
506.         client.subscribe(mqtt_topic)
507.     else:
508.         if do_debug(2): print(f"Verbindung fehlgeschlagen mit Code {rc}")
509.
510. def on_mqtt_disconnect(client, userdata, rc):
511.     if rc != 0:
512.         if do_debug(2): print("Unerwarteter Verbindungsverlust zum MQTT-Broker.")
513.         try:
514.             client.reconnect()
515.         except Exception as e:
516.             if do_debug(2): print(f"Fehler beim Versuch, die Verbindung wiederherzustellen:
{e}")
517.
518. # Hauptprogram - Initialisieren und in die Endlosschleife, auf Nachrichten warten
519. def main():
520.     # Prüfen, ob memcache und Datenbank verbunden sind
521.     try:
522.         memcache.set('some_key', 'some value')
523.     except Exception as e:
524.         sys.exit(50) # erfolgt, wenn obiger Befehl fehlschlägt / memchaed nicht verfügbar
                         ist
525.     # memcache.flush_all() # nur zum Testen mit einem leeren memcached
526.     # Verbindung zur Datenbank herstellen. Die Connection wird in der globalen
        'db_connection' gehalten.
527.     try:
528.         db_connection = mysql.connector.connect(**db_config)
529.     except mysql.connector.Error as err:
530.         exit(1) # Beenden des Programms mit dem Returnwert 1. Ohne Datenbank ist keine
                    Verarbeitung möglich.
531.     # Aufbau der initialen Verbindung zum MQTT. Da dieser durchaus neu Starten kann, wird
          diese Verbindung je nach Nutzung wiederhergestellt.
532.     # memcache.flush_all()
533.     # Vorbefüllung des memcache mit den letzten Daten aus der Datenbank
534.     load_initial_data()
535.     # Zuweisung der Methoden zur MQTT Verarbeitung
536.     client.on_connect = on_mqtt_connect          # Bei Aufbau einer Verbindung
537.     client.on_message = on_mqtt_message          # Bei Eingang einer Nachricht auf dem
                                                      konfigurierten Topic
538.     client.on_disconnect = on_mqtt_disconnect    # Bei (i.B. unerwarteten) Verlust der
                                                      Verbindung
539.
540.     # Starten der Verbindung zum MQTT - und damit Abarbeiten der Nachrichten
541.     while True:
542.         try:
543.             client.connect(mqtt_server, mqtt_port, 60)
544.             client.loop_start()
545.             # Einmal in der Schleife wird bei Wegfall der MQTT Verbindung die für diesen
                  Fall konfigurierten Methode zum Re-Connect aufgerufen
546.             while True:
547.                 time.sleep(1)
548.         except Exception as e:
```

```
549.             if do_debug(2): print(f"Fehler bei der Verbindung zum MQTT-Broker: {e}.
                                       Versuche, in 5 Sekunden erneut zu verbinden...")
550.             time.sleep(5)
551.
552. main()
```

```
  1. import mysql.connector
  2. import time
  3. from pymemcache.client import base
  4. import json
  5. import sys
  6. import os
  7. from dotenv import load_dotenv, find_dotenv
  8.
  9. # Laden der ungebungsabhängigen Konfigurationsparameter
 10. load_dotenv(find_dotenv('Config.env'))
 11. # Konfiguration der MySQL-Datenbankverbindung und ded Memcached
 12. db_config = {
 13.     'host': os.getenv('DB_HOST'),
 14.     'port': int(os.getenv('DB_PORT')),
 15.     'user': os.getenv('DB_USER'),
 16.     'password': os.getenv('DB_PASSWORD'),
 17.     'database': os.getenv('DB_DATABASE')
 18. }
 19. memcache_server = os.getenv('MEMCACHE_SERVER')
 20. memcache_port = os.getenv('MEMCACHE_PORT')
 21. db_connection = None    # Globale Variable zum Halten des DB connectors
 22. timegap = os.getenv('TIMEGAP')
 23.
 24. # Debug Level und Filter Funktion, Debug Meldungen aus dem Text können bitweise auf Leveln
      definiert werden
 25. debug_level = 2
 26. def do_debug(k):
 27.     temp = debug_level >> (k - 1)
 28.     return temp & 1 != 0
 29.
 30. # Serializer zur Memcache Nutzung, erleichten den Zugriff (Strings/Arrays statt bytes).
 31. def json_serializer(key, value):
 32.     return json.dumps(value), 1
 33. def json_deserializer(key, value, flags):
 34.     if flags == 1:
 35.         return json.loads(value)
 36.     raise Exception("Unknown serialization format")
 37.
 38. # Aufbau der globalen Verbindung zum memcached
 39. memcache = base.Client((memcache_server, memcache_port), serializer=json_serializer,
                deserializer=json_deserializer)
 40.
 41. # Funktion zum Auflösen eines Beaconpaars in der Datenbank und im Memcache
 42. def delete_beaconpair(beacon_id_1, beacon_id_2, DeleteFromArray):
 43.     try:
 44.         cursor = db_connection.cursor()
 45.         # Löschen der beiden möglichen Paarungen 1 + 2 oder 2 + 1 in der Datenbank
 46.         delete_query = '''
 47.             DELETE FROM beaconpair
 48.             WHERE (beaconpair_beacon_id_1 = %(id1)s AND beaconpair_beacon_id_2 = %(id2)s)
 49.                OR (beaconpair_beacon_id_1 = %(id2)s AND beaconpair_beacon_id_2 = %(id1)s)
 50.         '''
 51.         cursor.execute(delete_query, {'id1': beacon_id_1, 'id2': beacon_id_2})
 52.         db_connection.commit()
 53.         if do_debug(2): print(f"Beacon pair {beacon_id_1} and {beacon_id_2} dissolved in
                                   DB.")
 54.
 55.         # Erstellen der Namen der Datensätze im Memcache und holen der Datensätze
 56.         beaconpairs_key_1 = f'beaconpairs_{beacon_id_1}'
 57.         beaconpairs_key_2 = f'beaconpairs_{beacon_id_2}'
 58.         beaconpairs_temp_1 = memcache.get(beaconpairs_key_1)
 59.         beaconpairs_temp_2 = memcache.get(beaconpairs_key_2)
 60.
 61.         # Für jeden der Datensätze: Entfernen der jeweils anderen Beacon ID.
```

```
        # Zurückschreiben, wenn Paarungen verbleiben, sonst löschen
        if beaconpairs_temp_1:
            beaconpairs_temp_1 = [id for id in beaconpairs_temp_1 if id != beacon_id_2]
            if beaconpairs_temp_1 == []:
                memcache.delete(beaconpairs_key_1)
            else:
                memcache.set(beaconpairs_key_1, beaconpairs_temp_1)
        if beaconpairs_temp_2:
            beaconpairs_temp_2 = [id for id in beaconpairs_temp_2 if id != beacon_id_1]
            if beaconpairs_temp_2 == []:
                memcache.delete(beaconpairs_key_2)
            else:
                memcache.set(beaconpairs_key_2, beaconpairs_temp_2)
        if do_debug(2): print(f"Beacon pair {beacon_id_1} and {beacon_id_2} dissolved in
                              Memcache.")

        beaconpairs_temp_all = memcache.get('beaconpairs')
        beaconpairs_temp_all.pop(DeleteFromArray)
        memcache.set('beaconpairs', beaconpairs_temp_all)

    except mysql.connector.Error as err:
        print(f"Error dissolving beacon pair in MySQL: {err}")

# Funktion zum Überprüfen und Auflösen kritischer Mappings
def beaconpaar_validity_check():
    # Holen aller aktuell eingetragenen Beacon-Pairs
    beaconpairs = memcache.get('beaconpairs')
    if do_debug(4): print(beaconpairs)
    # Wenn die Liste nicht leer ist, wird über einen Positionsindex durch die Liste
      iteriert.
    # Der Positionsindex wird ebenfalls zum Löschen eines Eintrags der laufenden Liste
      verwendet.
    if beaconpairs != None:
        array_length = len(beaconpairs) # Bestimmen, wann der Aublauf beendet werden muss
        pairposition = 0 # Die Position des aktuellen Pairs in der Liste. Kein! Hochzählen
                         bei Löschen des aktuellen.
        while pairposition < array_length:
            beacon_id_1 = beaconpairs[pairposition].split(",")[0]
            beacon_id_2 = beaconpairs[pairposition].split(",")[1]
            # Erzeugung der Zugriffsnamen für die kritischen Mapping-Einträge und holen
              dieser Werte
            krit_key_1 = f'beacon_mapping_krit_{beacon_id_1}_{beacon_id_2}'
            krit_key_2 = f'beacon_mapping_krit_{beacon_id_2}_{beacon_id_1}'
            krit_data_1 = memcache.get(krit_key_1)
            krit_data_2 = memcache.get(krit_key_2)
            # Nutzung der lokalen Zeit zur Prüfung, ob die Zeitgrenze zum Löschen erreicht
ist
            current_time = int(time.time())
            # Wenn auch nur eines der möglichen zwei Einträge zu alt ist, dann Löschen des
              Paares
            if krit_data_1 and current_time - krit_data_1[0] > 60 or krit_data_2 and
               current_time - krit_data_2[0] > 60:
                delete_beaconpair(beacon_id_1, beacon_id_2, pairposition)
                memcache.delete(krit_key_1)
                if do_debug(2):
                    print(f"Critical mapping {krit_key_1} dissolved.")
                memcache.delete(krit_key_2)
                if do_debug(2):
                    print(f"Critical mapping {krit_key_2} dissolved.")
            else:
                pairposition +=1    # nächste Position, da aktuelle nicht gelöscht wurde

# Hauptprogramm
def main():
    # Prüfen, ob memcache und Datenbank verbunden sind
    try:
        memcache.set('some_key', 'some value')
    except Exception as e:
        sys.exit(50) # erfolgt, wenn obiger Befehl fehlschlägt / memchaed nicht verfügbar
ist
    # memcache.flush_all() # nur zum Testen mit einem leeren memcached
```

```
124.     # Verbindung zur Datenbank herstellen
125.     try:
126.         db_connection = mysql.connector.connect(**db_config)
127.     except mysql.connector.Error as err:
128.         exit(1) # Beenden des Programms mit dem Returnwert 1. Ohne Datenbank ist keine
                    Verarbeitung möglich.
129.     # Dauerhaftes Prüfen aller Beacon Paare. Die DB connection wird jede Minute genutzt,
           bleibt daher geöffnet.
130.     while True:
131.         beaconpaar_validity_check()
132.         time.sleep(60)  # 1 Minute warten
133. main()
```

A11 – index.js

```
require('dotenv').config()
const express = require('express');
const mariadb = require('mariadb');
const bodyParser = require('body-parser');
// Initialisieren und umgebungsspezifische Konfiguration einlesen

const app = express();
const port = process.env.API_PORT;

// Datenbankverbindung herstellen
const pool = mariadb.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
  database: process.env.DB_DATABASE,
  connectionLimit: process.env.DB_CONNECTION_LIMIT
});

// Dantenbank-Pool exportieren, damit die Routes darauf zugreifen können
module.exports.pool = pool;

// Middleware
app.use(bodyParser.json());
// Die Routes, die die jeweils aufgerufenen Pfade beantworten
app.use(require('./routes/bereich'));
app.use(require('./routes/raum'));
app.use(require('./routes/hub'));
app.use(require('./routes/beacon'));
app.use(require('./routes/mp'));
app.use(require('./routes/mp_mapping'));
app.use(require('./routes/mp_typ'));

// Start server
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});

```

A12 – bereich.js

```
 1. var express = require('express');
 2. var router = express.Router();
 3.
 4. const { pool } = require('../index');  // Datenbank-Pool
 5.
 6. // Bereiche auslesen
 7. router.get('/api/bereich', async (req, res) => {
 8.   let conn;
 9.   try {
10.     conn = await pool.getConnection();
11.     const rows = await conn.query('SELECT * FROM bereich');
12.     res.json(rows);
13.   } catch (err) {
14.     console.error(err);
15.     res.status(500).json({ message: 'Internal server error' });
16.   } finally {
17.     if (conn) conn.release();
18.   }
19. });
20.
21. // Werte eines einzelnen Bereiches lesen
22. router.get('/api/bereich/:id', async (req, res) => {
23. const { id } = req.params;
24. let conn;
25. try {
26.   conn = await pool.getConnection();
27.   const [bereich] = await conn.query('SELECT * FROM bereich WHERE bereich_id = ?', [id]);
28.   if (!bereich) {
29.     return res.status(404).json({ message: 'Area not found' });
30.   }
31.   res.json(bereich);
32. } catch (err) {
33.   console.error(err);
34.   res.status(500).json({ message: 'Internal server error' });
35. } finally {
36.   if (conn) conn.release();
37. }
38. });
39.
40. // Einen neuen Bereich hinzufügen
41. router.post('/api/bereich', async (req, res) => {
42.   const { bereich_name, bereich_aktiv, bereich_aktiv_seit } = req.body;
43.
44.   if (!bereich_name) {
45.     return res.status(400).json({ message: 'Missing required fields' });
46.   }
47.
48.   let conn;
49.   try {
50.     conn = await pool.getConnection();
51.     await conn.query(
52.       'INSERT INTO bereich (bereich_name, bereich_aktiv, bereich_aktiv_seit) VALUES (?, ?,
?)',
53.       [bereich_name, bereich_aktiv, bereich_aktiv_seit]
54.     );
55.     res.status(201).json({ message: 'Area created successfully' });
56.   } catch (err) {
57.     console.error(err);
58.     res.status(500).json({ message: 'Internal server error' });
59.   } finally {
60.     if (conn) conn.release();
61.   }
62. });
63.
64. // Einen Bereich aktualisieren
65. router.put('/api/bereich/:id', async (req, res) => {
66.   const { id } = req.params;
```

```
67.   const { bereich_name, bereich_aktiv, bereich_aktiv_seit, bereich_inaktiv_seit } =
            req.body;
68.   console.error(req.body);
69.
70.   if (!bereich_name) {
71.       return res.status(400).json({ message: 'Missing required fields' });
72.   }
73.
74.   let conn;
75.   try {
76.     conn = await pool.getConnection();
77.     await conn.query(
78.       'UPDATE bereich SET bereich_name = ?, bereich_aktiv = ?, bereich_aktiv_seit = ?,
           bereich_inaktiv_seit = ? WHERE bereich_id = ?',
79.       [bereich_name, bereich_aktiv, bereich_aktiv_seit, bereich_inaktiv_seit, id]
80.     );
81.     res.status(200).json({ message: 'Area updated successfully' });
82.   } catch (err) {
83.     console.error(err);
84.     res.status(500).json({ message: 'Internal server error' });
85.   } finally {
86.     if (conn) conn.release();
87.   }
88. });
89.
90. // Einen Bereich löschen
91. router.delete('/api/bereich/:id', async (req, res) => {
92. const { id } = req.params;
93. if (!id) {
94.   return res.status(400).json({ message: 'Missing required fields' });
95. }
96. let conn;
97. try {
98.   conn = await pool.getConnection();
99.   await conn.query(
100.     'DELETE from bereich where bereich_id = ?', [id]
101.     );
102.     res.status(200).json({ message: 'Area deleted successfully' });
103.   } catch (err) {
104.     console.error(err);
105.     res.status(500).json({ message: 'Internal server error' });
106.   } finally {
107.     if (conn) conn.release();
108.   }
109. });
110.
111. module.exports = router;
112.
```

A12 – bereich.js

A13 – app.vue

```
1.  <template>
2.    <div>
3.      <NuxtLayout>
4.        <nav class="navbar">
5.          <ul class="nav-list">
6.            <li class="nav-item"><nuxt-link to="/" exact-active-class="active-
                        link">Home</nuxt-link></li>
7.            <li class="nav-item"><nuxt-link to="/bereich" exact-active-class="active-
                        link">Bereiche</nuxt-link></li>
8.            <li class="nav-item"><nuxt-link to="/raum" exact-active-class="active-
                        link">Räume</nuxt-link></li>
9.            <li class="nav-item"><nuxt-link to="/hub" exact-active-class="active-
                        link">Hubs</nuxt-link></li>
10.           <li class="nav-item"><nuxt-link to="/beacon" exact-active-class="active-
                        link">Beacons</nuxt-link></li>
11.           <li class="nav-item"><nuxt-link to="/mp" exact-active-class="active-
                        link">Medizinprodukte</nuxt-link></li>
12.           <li class="nav-item"><nuxt-link to="/mp_typ" exact-active-class="active-
                        link">Medizinprodukt Typen</nuxt-link></li>
13.           <li class="nav-item"><nuxt-link to="/mp_mapping" exact-active-class="active-
                        link">Medizinprodukt Mapping</nuxt-link></li>
14.       </ul>
15.     </nav>
16.       <NuxtPage />
17.     </NuxtLayout>
18.   </div>
19. </template>
```

A14 – hub.vue

```
<template>
  <div class="container">
      <router-link to="/hub_add" custom v-slot="{ navigate }">
        <button @click="navigate" role="link">Neuen Hub hinzufügen</button>
      </router-link>
      <p><br></p>
      <table class="table">
        <thead>
          <tr>
            <th @click="sort('hub_id')">ID <span v-show="sortColumn === 'hub_id'">{{ sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
            <th @click="sort('hub_MAC')">MAC <span v-if="sortColumn === 'hub_MAC'">{{ sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
            <th @click="sort('hub_aktiv')">Aktiv <span v-if="sortColumn === 'hub_aktiv'">{{ sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
            <th @click="sort('hub_timestamp')">Zuletzt aktiv am/um <span v-if="sortColumn === 'hub_timestamp'">{{ sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
            <th @click="sort('hub_aktiv_seit')">Aktiv seit <span v-if="sortColumn === 'hub_aktiv_seit'">{{ sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
            <th @click="sort('hub_inaktiv_seit')">Deaktiviert am <span v-if="sortColumn === 'hub_inaktiv_seit'">{{ sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
            <th @click="sort('bereich_name')">Bereich <span v-if="sortColumn === 'bereich_name'">{{ sortOrder === 'asc' ? '↑' : '↓' }}</span></th>
            <th @click="sort('raum_name')">Raum <span v-if="sortColumn === 'raum_name'">{{ sortOrder === 'asc' ? '↑' : '↓' }}</span></th>


            <th>Bearbeiten</th>
            <th>Löschen</th>
          </tr>
        </thead>
        <tbody>
          <tr v-for="hub in hubs" :key="hub.hub_id">
            <td>{{ hub.hub_id }}</td>
            <td>{{ hub.hub_MAC }}</td>
            <td><template v-if="hub.hub_aktiv == true">✓</template></td>
            <td>{{ hub.hub_timestamp }}</td>
            <td>{{ formatDate(hub.hub_aktiv_seit) }}</td>
            <td>{{ formatDate(hub.hub_inaktiv_seit) }}</td>
            <td>{{ hub.bereich_name }}</td>
            <td>{{ hub.raum_name }}</td>
            <td><router-link :to="'/hub_edit?id=' + hub.hub_id">Bearbeiten</router-link></td>
            <td><button @click="delete_hub(hub.hub_id)">Löschen</button></td>
          </tr>
        </tbody>
      </table>
    </div>
  </template>

  <script>
  export default {
    data() {
      return {
        sortColumn: null,
        sortOrder: 'asc',
        hubs: []
      };
    },
    name: 'hubs',
    data() {
      return {
        hubs: []
      };
    },
    mounted() {
        Error('Network response was not ok');
        console.log('Fetching hubs...');
```

```
60.         this.fetch_hubs();
61.     },
62.     methods: {
63.       sort(sortKey) {
64.         if (this.sortColumn === sortKey) {
65.           this.sortOrder = this.sortOrder === 'asc' ? 'desc' : 'asc';
66.         } else {
67.           this.sortOrder = 'asc';
68.           this.sortColumn = sortKey;
69.         }
70.
71.         this.hubs.sort((a, b) => {
72.           let comparison = 0;
73.           let valueA = typeof a[sortKey] === 'string' ? a[sortKey].toLowerCase() :
                                                           a[sortKey];
74.           let valueB = typeof b[sortKey] === 'string' ? b[sortKey].toLowerCase() :
                                                           b[sortKey];
75.
76.           if (valueA > valueB) comparison = 1;
77.           if (valueA < valueB) comparison = -1;
78.           if (this.sortOrder === 'desc') comparison *= -1;
79.           return comparison;
80.         });
81.       },
82.
83.       formatDate(dateString) {
84.         if (dateString) {
85.           const date = new Date(dateString);
86.           return new Intl.DateTimeFormat('default', {dateStyle: 'long'}).format(date);
87.         } else {
88.           return '';
89.         }
90.       },
91.       fetch_hubs() {
92.             fetch('http://localhost:3000/api/hub')
93.             .then(response => {
94.                 console.log('response: ',response)
95.                 if (!response.ok) {
96.                     throw new Error('Network response was not ok');
97.                 }
98.                 return response.json();
99.             })
100.            .then(data => {
101.                this.hubs = data;
102.            })
103.            .catch(error => {
104.                console.error('Error fetching hubs:', error);
105.            });
106.        },
107.
108.      delete_hub(hubID) {
109.
110.        console.error('Try to delete Hub with ID :', hubID);
111.        fetch(`http://localhost:3000/api/hub/${hubID}`, { method: 'DELETE' })
112.          .then(response => {
113.            console.log('Response 1', response);
114.            if (!response.ok) {
115.              throw new Error('Network response was not ok');
116.            }
117.
118.            this.fetch_hubs();
119.            })
120.            .catch(error => {
121.              console.error('Error:', error);
122.            });
123.    }
124.  }
125. }
126.
127. </script>
128.
```

A14 – hub.vue

A14 – hub.vue

A15 – hub_add.vue

```
<template>
    <div>
      <h1>Hub hinzufügen</h1>
        <form @submit.prevent="hinzufuegen_Hub">
              <table class="table">
                <thead>
                    <tr>
            <th>MAC</th>
            <th>Aktiv seit</th>
            <th>Bereich</th>
            <th>Raum-Name</th>
                </tr>
                </thead>
                <tbody>
                    <tr>
                        <td><input type="text" id="hub_MAC" v-model="neuer_Hub.hub_MAC"
                              required></td>

                        <td><input type="date" id="hub_aktiv_seit" v-
                              model="formattedActiveSince"></td>

                        <td>
                            <select id="hub_bereich_id" v-model="neuer_Hub.bereich_id"
                              @change="fetch_raeume(neuer_Hub.bereich_id)">
                                <option v-for="bereich in bereiche"
                                 :key="bereich.bereich_id" :value="bereich.bereich_id">{{
                                 bereich.bereich_name }}</option>
                            </select>
                        </td>
                        <td>
                            <select id="hub_raum_id" v-model="neuer_Hub.raum_id" >
                                <option v-for="raum in raeume" :key="raum.raum_id"
                                 :value="raum.raum_id">{{ raum.raum_name }}</option>
                            </select>
                        </td>
                </tr>
                </tbody>
              </table>
              <br>
              <button type="submit">Hub hinzufügen</button>
        </form>
      </div>
  </template>

  <script setup>
  import { ref } from 'vue'
  import { useRouter } from 'vue-router'

  const router = useRouter()

  const neuer_Hub = ref({
    hub_MAC: '',
    hub_aktiv: false,
    hub_aktiv_seit: null,
    hub_bereich_id: null,
    hub_raum_id: null
  })

  const bereiche = ref([])
  const raeume = ref([])

  const formattedActiveSince = computed({
    get: () => neuer_Hub.value.hub_aktiv_seit ?
                neuer_Hub.value.hub_aktiv_seit.toISOString().substr(0, 10) : '',
    set: (value) => neuer_Hub.value.hub_aktiv_seit = new Date(value)
  })

// Holen aller Bereiche aus der Datenbank
```

```
62. // 'zuruecksetzen' wird benutzt, um zwischem erstem Laden und Änderung im Formular zu
        unterscheiden
63. async function fetch_bereiche() {
64.   try {
65.     const response = await fetch('http://localhost:3000/api/bereich')
66.     if (!response.ok) {
67.       throw new Error('Failed to fetch areas')
68.     }
69.     bereiche.value = await response.json()
70.     console.log('Bereiche geholt. Bereiche: ', bereiche.value, "Aktuelle BereichsID ist ",
                    neuer_Hub.value.bereich_id)
71.
72.
73.   } catch (error) {
74.     console.error('Error fetching areas:', error)
75.   }
76. }
77.
78. // Holen aller Räume eines Bereiches aus der Datenbank
79. async function fetch_raeume(bereich) {
80.     try {
81.         // Holen der Räume des selektierten Bereiches! Also nur genau dieser Räume
82.         // `` zur Interpretation der Variablen ...
83.         const response = await fetch(`http://localhost:3000/api/raum_bereich/${bereich}`)
84.         if (!response.ok) {
85.         throw new Error('Fehler beim Lesen der Räume')
86.         }
87.         raeume.value = await response.json()
88.     } catch (error) {
89.         console.error('Error Fehler beim Holen der Räume:', error)
90.     }
91.     console.log('Räume geholt: ', raeume.value)
92.     // Wenn ein neuer Bereich gewählt wurde, muss das Feld im Formular geleert werden
93.     neuer_Hub.raum_id = null // Bereichswechsel
94. }
95.
96. // Holen der Bereiche beim Laden
97. onMounted(fetch_bereiche)
98.
99. async function hinzufuegen_Hub() {
100.     try {
101.       if (neuer_Hub.value.hub_aktiv_seit) {
102.         neuer_Hub.value.hub_aktiv = 1
103.        console.log("aktiv datum gesetzt, setze aktiv flag")
104.       }
105.       const response = await fetch('http://localhost:3000/api/hub', {
106.         method: 'POST',
107.         headers: {
108.           'Content-Type': 'application/json'
109.         },
110.         body: JSON.stringify(neuer_Hub.value)
111.       })
112.       console.log('Response', response)
113.       if (!response.ok) {
114.         throw new Error('Network response was not ok')
115.       }
116.       // zu "hub" weiterleiten
117.       router.push('/hub')
118.     } catch (error) {
119.       console.error('Error adding room:', error)
120.     }
121.   }
122.   </script>
123.
```

A15 – hub_add.vue

A16 – Python-Belastungstest

```python
import time
import threading
import paho.mqtt.client as mqtt
import random
import os
from dotenv import load_dotenv, find_dotenv

# Laden der umgebungsabhängigen Konfigurationsparameter
load_dotenv(find_dotenv('Config.env'))

# MQTT Broker Konfiguration
mqtt_server = os.getenv('MQTT_SERVER')
mqtt_port = int(os.getenv('MQTT_PORT'))
mqtt_topic = os.getenv('MQTT_TOPIC')
mqtt_username = os.getenv('MQTT_USER')
mqtt_password = os.getenv('MQTT_PW')
messages_per_second = 100

# MAC-Adressen aus der Datei laden
def load_mac_addresses(file_path):
    with open(file_path, 'r') as file:
        mac_addresses = [line.strip() for line in file.readlines()]
    return mac_addresses

mac_addresses = load_mac_addresses("C:/users/linus/downloads/beacon_MAC.txt")

def on_disconnect(client, userdata, rc):
    if rc != 0:
        print("Unerwarteter Verbindungsverlust zum MQTT-Broker.")
        while True:
            try:
                print("Versuche, die Verbindung zum MQTT-Broker wiederherzustellen...")
                client.reconnect()
                break
            except Exception as e:
                print(f"Verbindung fehlgeschlagen: {e}")
                time.sleep(5)

def publish_messages(client):
    i = 1
    while True:
        for _ in range(messages_per_second):
            mac_index = (i - 1) % len(mac_addresses)
            mac_address = mac_addresses[mac_index]
            randRSSI = random.randint(100,250)
            if i % 4 == 0:
                lastNum = random.randint(1, 9)
            else:
                lastNum = 2
            if i % 44 == 0 or i % 45 == 0:
                randButton = random.randint(0, 1)
            else:
                randButton = 0
            message_content = f"layer=1, MAC_ROOM=00:00:00:00:00:0{lastNum},
                                MAC_SENSOR={mac_address}, BATT=100, BUTTON={randButton},
                                RSSI={randRSSI}"

            client.publish(mqtt_topic, message_content)
            print(message_content)
            i += 1
        time.sleep(1)

def main():
    mqtt_client = mqtt.Client()
    mqtt_client.username_pw_set(mqtt_username, mqtt_password)
    mqtt_client.on_disconnect = on_disconnect

    while True:
```

```
67.         try:
68.             mqtt_client.connect(mqtt_server)
69.             publish_thread = threading.Thread(target=publish_messages, args=(mqtt_client,))
70.             publish_thread.start()
71.             mqtt_client.loop_forever()
72.         except Exception as e:
73.             print(f"Ein Fehler ist aufgetreten: {e}")
74.             time.sleep(5)
75.
76. main()
77.
78.
```