

# Lokalisierung von Dokumenten und Tracking des Workflows mit Hilfe von IoT-Geräten

## STUDIENARBEIT

des Studienganges Informatik  
an der Dualen Hochschule Baden-Württemberg Karlsruhe

**Max Heidinger**  
2361793

**Pascal Riesinger**  
7586259

Abgabedatum	18.05.2020
Bearbeitungszeitraum	01.10.2019 - 18.05.2020
Kurs	TINF17B1
Betreuer	Prof. Dr. Marcus Strand

# **Eidesstattliche Erklärung**

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema:

*Lokalisierung von Dokumenten und Tracking des Workflows mit Hilfe von IoT-Geräten*

gemäß § 5 der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. September 2017 selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, den 22. April 2020

---

Max Heidinger, Pascal Riesinger

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>IV</b>
<b>Abbildungsverzeichnis</b>	<b>VI</b>
<b>Listings</b>	<b>VII</b>
<b>1 Einführung</b>	<b>1</b>
1.1 Idee des Paper-Tracker . . . . .	1
<b>2 Grundlagen</b>	<b>2</b>
2.1 Projektumfeld . . . . .	2
2.2 Funktionsweise WLAN . . . . .	2
2.2.1 BSS . . . . .	2
2.2.2 ESS . . . . .	3
2.2.3 Signalstärke RSSI . . . . .	3
2.3 Beispiel eines Workflows . . . . .	3
<b>3 Analyse</b>	<b>6</b>
3.1 Ist-Analyse . . . . .	6
3.2 Soll-Analyse . . . . .	6
3.3 Anforderungsanalyse . . . . .	6
<b>4 Entwurf</b>	<b>8</b>
4.1 Systemarchitektur . . . . .	8
4.2 Komponentenarchitektur . . . . .	10
4.2.1 Tracking mit IoT-Hardware . . . . .	10
4.2.2 Backend-Anwendung . . . . .	11
4.2.3 App für Mobilgeräte . . . . .	22
4.2.4 Flasher-Tool . . . . .	32
<b>5 Implementierung</b>	<b>34</b>
5.1 Backend-Server . . . . .	34
5.1.1 Verwendete Technologien . . . . .	34
5.1.2 REST-Schnittstelle . . . . .	36
5.1.3 CoAP-Schnittstelle . . . . .	36
5.1.4 Tracker-Lokalisierung . . . . .	36
5.1.5 Daten-Export . . . . .	37
5.1.6 Tests . . . . .	38

5.2	Hardware und Firmware . . . . .	39
5.2.1	Verwendete Technologien . . . . .	39
5.2.2	Flasher . . . . .	40
5.3	App . . . . .	43
5.3.1	Verwendete Technologien . . . . .	43
5.4	Hardware-Gehäuse . . . . .	44
5.4.1	Layout Controller auf der Batterie . . . . .	45
5.4.2	Layout Controller neben der Batterie . . . . .	47
5.5	Bestehende Herausforderungen . . . . .	49
5.5.1	Ungenauigkeit des Tracking . . . . .	49
5.5.2	UI-Details in der App . . . . .	49
<b>6</b>	<b>Validierung</b>	<b>50</b>
6.1	Genauigkeitsmessung des Tracking . . . . .	50
6.1.1	Beschreibung . . . . .	50
6.1.2	Durchführung . . . . .	50
6.1.3	Ergebnisse . . . . .	50
6.2	Nutzerumfrage zur UX der App . . . . .	50
6.2.1	Beschreibung . . . . .	50
6.2.2	Durchführung . . . . .	50
6.2.3	Ergebnisse . . . . .	50
6.2.4	Gesamtergebniss der Validierung . . . . .	50
<b>7</b>	<b>Ausblick und Weiterentwicklungen</b>	<b>51</b>
7.1	Audiovisuelle Signale . . . . .	51
7.2	Verbessertes Gehäuse . . . . .	51
7.3	Tracking von Smartphones . . . . .	51
<b>8</b>	<b>Zusammenfassung</b>	<b>52</b>
<b>Glossar</b>		<b>53</b>
<b>Literatur</b>		<b>VIII</b>

# Abkürzungsverzeichnis

**Akku** Akkumulator. 39, 40

**AMQP** Advanced Message Queuing Protocol. 15

**AP** Access Point. 2, 10, 11

**API** Application Programming Interface. 36, 39

**BDD** Behavior-driven development. 35

**BSS** Basic Service Set. II, 2, 3

**BSSID** Basic Service Set Identifiers. 2, 3, 12

**CBOR** Concise Binary Object Representation. 15, 35

**CoAP** Constrained Application Protocol. 15, 19, 34, 35

**CSV** Comma Separated Values. 21

**DHBW** Duale Hochschule Baden-Württemberg. VI, 1–4

**ESS** Extended Service Set. II, 3

**GUI** Graphical User Interface. 32, 40, 41

**HTTP** Hypertext Transfer Protocol. 15, 16, 19, 34, 43

**ID** Identifier. 36

**IEEE** Institute of Electrical and Electronics Engineers. 2

**IoT** Internet of Things. 15

**IP** Internet Protocol. 32

**JSON** JavaScript Object Notation. 15, 16, 35

**LAN** Local Area Network. 3

**LED** Light Emitting Diode. 40

**MAC** Media Access Control. 2

**MQTT** Message Queuing Telemetry Transport. 15

**ORM** Object-relational mapping. 34

**REST** Representational State Transfer. 16, 22, 38

**RPCI** Received Channel Power Indicator. 3, 10

**RSSI** Received Signal Strength Indication. II, 3, 10, 11

**SSID** Service Set Identifier. 2, 3, 28, 32

**UART** Universal Asynchronous Receiver Transmitter. 39

**UDP** User Datagram Protocol. 36

**UI** User Interface. 22, 32

**URL** Uniform Resource Locator. 44

**USB** Universal Serial Bus. 39, 40, 45, 47

**WLAN** Wireless Local Area Network. II, 2, 10, 11, 32, 36, 39, 41

**XLSX** Microsoft Excel Open XML Spreadsheet. 35

# Abbildungsverzeichnis

2.1	Beschaffungs-Genehmigungsverfahren der Duale Hochschule Baden-Württemberg (DHBW) . . . . .	4
4.1	Architektur des Paper-Tracker . . . . .	9
4.2	UML-Diagramm für die Backend-Anwendung . . . . .	14
4.3	Software-Architektur für die Backend-Anwendung . . . . .	20
4.4	Design-Prototyp der Hauptseite . . . . .	23
4.5	Design-Prototyp der Darstellung von Workflows . . . . .	25
4.6	Design-Prototyp der Detailseite eines Trackers . . . . .	27
4.7	Design-Prototyp der Einlern-Seite . . . . .	28
4.8	Design-Prototyp eines Pop-ups zum Erstellen eines Raumes . . . . .	29
4.9	Design-Prototyp des Tutorials . . . . .	30
4.10	Icons für Klassen . . . . .	32
5.1	Template-Sheet eines Excel-Exports . . . . .	38
5.2	Ursprüngliche Revisionen-Sheet eines Excel-Exports . . . . .	38
5.3	Layout des Eingabe-Fenster . . . . .	42
5.4	Layout des Programmier-Fenster . . . . .	43
5.5	Gehäuse-Layout mit Mikrocontroller auf der Batterie . . . . .	45
5.6	Körper des Gehäuse-Modell mit Controller auf der Batterie . . . . .	46
5.7	Gehäuse-Modell mit Controller auf der Batterie . . . . .	46
5.8	Gehäuse-Layout mit Mikrocontroller neben der Batterie . . . . .	47
5.9	Körper des Gehäuse-Modell mit Controller neben der Batterie . . . . .	48
5.10	Deckel des Gehäuse-Modell mit Controller neben der Batterie . . . . .	48
5.11	Clip des Gehäuse-Modell mit Controller neben der Batterie . . . . .	49

# **Listings**

# 1 Einführung

An der DHBW in Karlsruhe basieren, wie auch noch in vielen anderen Organisationen, viele Prozesse auf Papier. Dies bedeutet, dass beispielsweise für die Genehmigung eines Einkaufes, ein oder mehrere Blätter Papier von unterschiedlichen Personen unterzeichnet werden müssen. Dafür wird das Dokument durch die einzelnen Büros getragen, bis letztendlich der Antragsteller es wieder bekommt.

Auch für simple Prozesse kann dies einige Zeit in Anspruch nehmen. Während dieser ganzen Zeit hat der Antragsteller leider keine Einsicht, bei welcher Person sich die Papiere befinden. Damit kann er auch nicht versuchen den Prozess zu beschleunigen, ohne Schritt für Schritt bei allen Personen des Prozesses nachzufragen. Zusätzlich kann nach Abschluss des Prozesses nicht nachvollzogen werden, wo der Flaschenhals des Prozesses liegt.

Um dieses Problem anzugehen, wird der „Paper-Tracker“ entwickelt. Dieser soll eine Transparenz in alle auf Papier basierten Prozesse bringen und zusätzlich Daten liefern, um die Prozesse selbst zu optimieren.

## 1.1 Idee des Paper-Tracker

Der Paper-Tracker ist ein kleines Gerät basierend auf einem Mikrocontroller, welcher an den Dokumenten befestigt wird. Dieser soll eine Lokalisierung der Papiere ermöglichen. Auf diese Lokalisierung soll der Antragsteller eines Prozesses über eine App Zugriff bekommen. Weiter soll auch in der App für Mobilgeräte der komplette Prozess oder Workflow selbst verfolgt werden. Somit kann zum Beispiel auch eine Notifizierung zuständiger Personen für einen Schritt in einem Workflow erfolgen.

## 2 Grundlagen

### 2.1 Projektumfeld

Das Projekt wird im Umfeld der DHBW Karlsruhe durchgeführt.

### 2.2 Funktionsweise WLAN

Unter WLAN versteht man im Allgemeinen ein Funknetz nach einem Standard der Institute of Electrical and Electronics Engineers (IEEE) 802.11 Familie. Alle Standards der Familie basieren auf dem ursprünglichen 802.11 Standard und erweitern diesen um neue Funktionalität, Frequenzen und höheren Datenraten. (vgl. [1])

Primärquelle finden

Für die Funktionsweise des Paper-Trackers muss der Teil der „Service Sets“ der Spezifikation detailliert erläutert werden. Unter einem Service Set versteht man im Allgemeinen alle Geräte in einem WLAN-Netzwerk. Auf diesen basiert die Lokalisierung der Dokumente.

#### 2.2.1 BSS

Ein BSS ist Service Set, in dem es nur einen Access Point (AP) und damit ein Gerät, dass das Funknetz aufbaut, gibt. Zu diesem AP können beliebig viele Klienten verbunden sein. In einem solchen Netzwerk besitzt der AP eine Service Set Identifier (SSID) und eine Basic Service Set Identifiers (BSSID). Die SSID ist der Name des Netzwerks, das von einem Endbenutzer ausgewählt wird, um sich mit dem WLAN zu verbinden. Da mehrere BSS die gleiche SSID verwenden können, besitzt der AP auch eine BSSID. Diese ist meist die Media Access Control (MAC)-Adresse des AP, welche per Definition einzigartig ist. (vgl. [2]) Mit Hilfe dieser BSSID können Klienten zwischen den verschiedenen Netzwerken mit identischer SSID unterscheiden.

Primärquelle finden (IEEE RFC)

## 2.2.2 ESS

In einem ESS existieren mehrere BSS, die miteinander z.B. über ein Local Area Network (LAN) verbunden sind. Alle BSS in diesem Service Set haben eine eigene BSSID aber bauen ein Netz mit der gleichen SSID auf. So können Klienten z.B. in größeren Gebäuden zwischen verschiedenen BSS wechseln, ohne es zu bemerken. (vgl. [3])

Primärquelle finden

## 2.2.3 Signalstärke RSSI

Die RSSI ist eine Größe, welche im Allgemeinen die Energie eines Radiosignals repräsentiert. Es gibt keine Standardisierung hinsichtlich der Messmethode oder der entstehenden Einheit, in welcher die RSSI angegeben wird. Da sie ein Maß für Dämpfung ist, wird jedoch immer eine negative Zahl angegeben. Je näher sich diese Zahl null nähert, desto stärker ist das Signal. Typischerweise liegt der Messwert im Bereich zwischen null und  $-100$ .

Als standardisierter Ersatz für die RSSI wurde der Received Channel Power Indicator (RPCI) eingeführt. Diese Größe wird immer in der Einheit dBm angegeben und wird über ein definiertes Intervall gemessen. Der Wertebereich bewegt sich dabei zwischen  $-110dBm$  und  $0dBm$ . Repräsentiert wird die Messung als 8-Bit-Ganzzahl, was abzüglich einiger reservierter Werte zu einer maximalen Auflösung von  $0.5dBm$  führt. Auch standardisiert ist die zu erreichende Genauigkeit, so soll die Abweichung bei maximal  $5dBm$  liegen. (vgl. [4])

## 2.3 Beispiel eines Workflows

In folgender Abbildung 2.1 ist ein Beispiel für einen Prozess an der DHBW Karlsruhe dargestellt. Es ist das „Beschaffungs-Genehmigungsverfahren“, das benötigt wird, um eine Bestellung für die DHBW durchzuführen.

Der Prozess beginnt mit dem Antragsteller, welcher den Beschaffungsantrag stellt. Der Antrag geht zum Fachvorgesetzten des Antragstellers, welcher diesen fachlich prüft. Ist die Prüfung nicht erfolgreich, wird eine Ablehnung begründet und der Prozess beendet.

Bei erfolgreicher Prüfung muss der Antrag, wenn er über 5.000 liegt, zudem vom Dekan

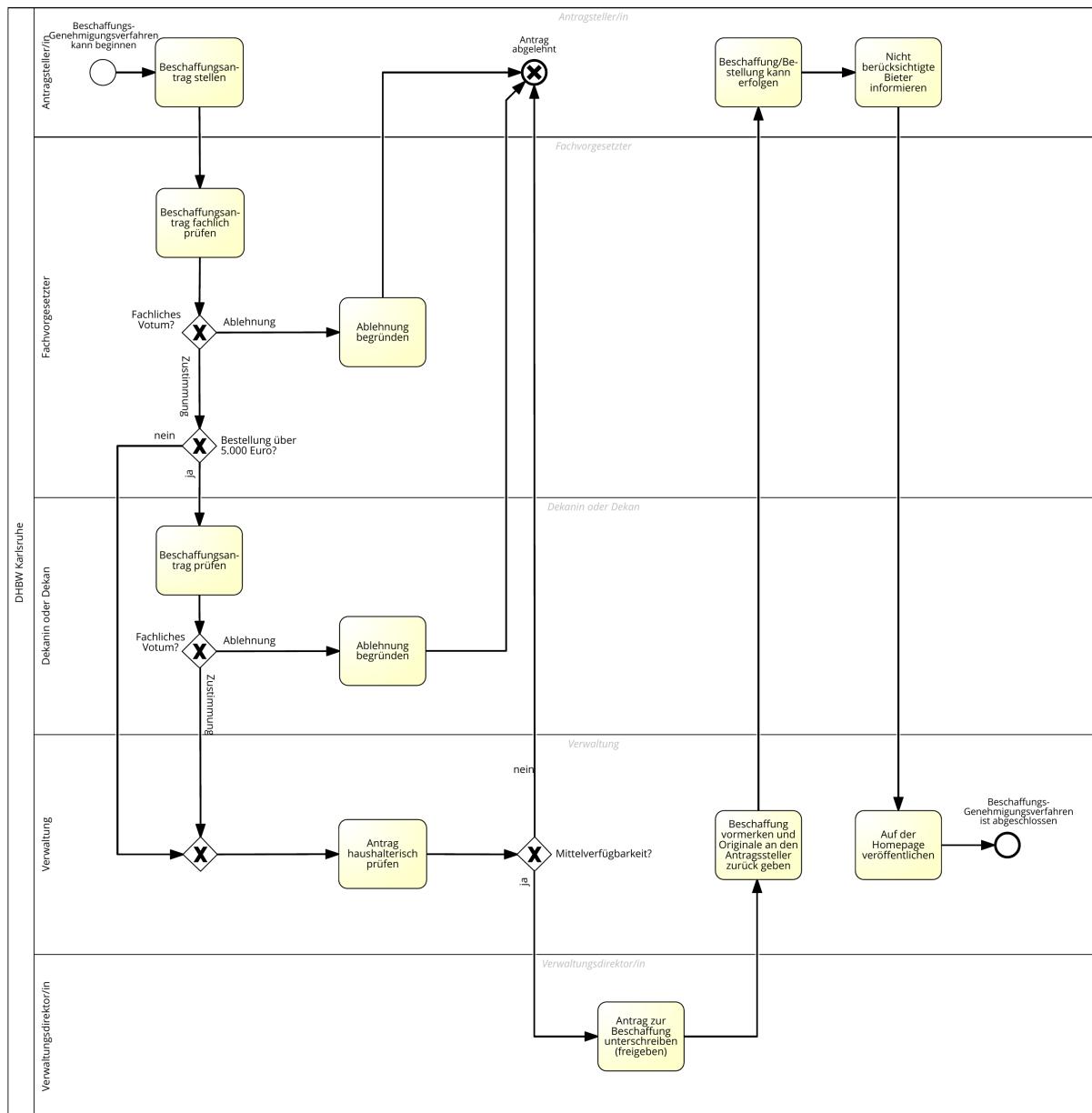


Abbildung 2.1: Beschaffungs-Genehmigungsverfahren der DHBW

geprüft werden. Dieser kann entweder den Antrag wie oben beschrieben ablehnen oder stimmt diesem zu.

Ist der Antrag für eine Bestellung unter 5.000 oder er bekommt die Zustimmung des Dekans, erhält als nächstes die Verwaltung den Antrag. Die Verwaltung muss prüfen, ob die Mittel für diese Bestellung vorhanden sind. Sind die Mittel nicht vorhanden, wird der Antrag abgelehnt. Bei vorhandenen Mitteln wird der Antrag von dem Verwaltungsdirektor unterschrieben und damit freigegeben. Damit wird die Beschaffung in der Verwaltung vorgemerkt und der Antrag geht an den ursprünglichen Antragsteller zurück.

Sobald der Antragsteller den unterzeichneten Antrag erhält, kann die Beschaffung erfolgen. In diesem Schritt müssen mögliche nicht berücksichtigte Bieter im Falle einer Ausschreibung informiert werden.

Im letzten Schritt veröffentlicht die Verwaltung das Ergebnis der Beschaffung auf der Homepage und der Prozess ist erfolgreich abgeschlossen.

# 3 Analyse

In den folgenden Abschnitten werden der Ist-Zustand, aus welchem sich die Problemstellung ergibt, der Soll-Zustand, sowie die gestellten Anforderungen beschrieben.

## 3.1 Ist-Analyse

Es besteht das Problem, dass der aktuelle Status eines auf Papier basierten Prozesses nicht nachvollzogen werden kann. Für den Antragsteller eines Prozesses ist es nicht möglich herauszufinden, bei welcher Person oder in welchem Raum sich ein Dokument zu einem bestimmten Zeitpunkt befindet. Zusätzlich kann, sobald der Prozess beendet ist, nicht herausgefunden werden, an welcher Stelle im Prozess zum Beispiel eine besonders hohe Wartezeit stattgefunden hat.

## 3.2 Soll-Analyse

Ziel der Entwicklung und damit dieser Studienarbeit ist das Erarbeiten einer Lösung, mit welcher die aktuelle Position eines Dokuments und der aktuelle Fortschritt des Prozesses sichtbar gemacht werden kann. Es soll ein möglichst unauffälliges Gerät sein und eine intuitive Bedienung ermöglichen. Über aktuelle und abgeschlossene Prozesse sollen Daten analysiert werden können, um Möglichkeiten zur Prozessoptimierungen erkennen zu können.

## 3.3 Anforderungsanalyse

Im Folgenden werden die gestellten Anforderungen in funktionale und nichtfunktionale Anforderungen unterteilt. Dabei beschreiben funktionale Anforderungen konkrete Funktionen, die die entwickelte Lösung bieten muss, nichtfunktionale Anforderungen hingegen zeigen Rahmenbedingungen, sowie Anforderungen an die technische Umsetzung auf.

Evtl. noch  
eine Anfor-  
derungsana-  
lyse mit an-  
deren Leu-  
ten durch-  
führen

Desweiteren werden den Anforderungen Prioritäten zugewiesen, wobei ein Wert von 0 die höchste Priorität, ein Wert von 3 die niedrigste Priorität beschreibt, sodass die Lösung ohne Erfüllung der Anforderungen mit Priorität 0 nicht einsetzbar ist und Priorität 3 optionale Funktionalitäten beschreibt, die nicht zum Betrieb notwendig sind.

- F-1** *P0* Der Standort von Dokumenten kann raumgenau verfolgt werden
- F-2** *P0* Der aktuelle Status des mit dem Dokument verbundenen Workflows kann nachvollzogen werden
- F-3** *P2* Neue Workflows können vom Nutzer definiert werden
- F-4** *P1* Neue Tracker können dem System vom Nutzer hinzugefügt werden
- F-5** *P2* Der Raumplan kann vom Nutzer eingelernt werden
- F-6** *P3* Nutzer werden vom System benachrichtigt, wenn die Akkuladung eines Trackers unter einen gegebenen Prozentsatz fällt oder die Batterie ausgetauscht werden muss
- F-7** *P1* Einzelne Schritte von Workflows können optional sein und übersprungen werden
- F-8** *P3* Für einen Schritt im Workflow können mehrere Orte hinterlegt werden
- F-9** *P1* Es soll ein Daten-Export in ein passendes Format möglich sein, das für Analysen zu Optimierungen genutzt werden kann
- NF-1** *P2* Die Laufzeit der Tracker im Batteriebetrieb beläuft sich auf mindestens eine Woche
- NF-2** *P1* Die Tracker sollen möglichst unscheinbar und daher klein und leicht sein
- NF-3** *P0* Die Daten können in einer Applikation für Mobiltelefone abgefragt werden. Dabei ist besonders die Verwendbarkeit unter dem Betriebssystem Android wichtig, die Unterstützung von iOS ist optional
- NF-4** *P1* Die Anwendung muss stabil laufen und sollte sich nicht unerwartet beenden
- NF-5** *P2* Die Lösung sollte skalierbar hinsichtlich der Anzahl der Tracker und der Größe des Tracking-Bereiches sein
- NF-6** *P2* Zur optimalen Nachvollziehbarkeit von Workflows sollten deren auch nach Abschluss nicht gelöscht werden

# 4 Entwurf

## 4.1 Systemarchitektur

In Abbildung 4.1 ist eine grobe Übersicht über die verschiedenen Komponenten des Projektes gegeben. Es besteht aus drei Komponenten:

- Hardware-Tracker
- Backend-Anwendung
- App

Der Hardware-Tracker ist die physische Komponente, die direkt an ein zu trackendes Papier angeheftet wird. Er wird regelmäßig Daten erfassen und diese zur Lokalisierung an die Backend-Anwendung senden.

Die Backend-Anwendung dient als zentrales Element der Architektur. Sie bekommt Daten des Trackers zugeschickt, wertet diese aus und speichert sie. Auch das Management der Räume und Workflows wird von der Anwendung übernommen.

Nutzer verwenden zur Kommunikation mit dem Backend eine Smartphone-App. Sie kommuniziert ebenfalls mit der Backend-Anwendung und fragt von dieser Informationen an oder löst Aktionen aus.

Die Entwürfe der drei Komponenten werden in den nachfolgenden Abschnitten im Detail erläutert.

Zusätzlich zu den drei Hauptkomponenten soll es eine Tool geben, dass das Setup eines Hardware-Trackers erleichtert. Dieses Tool, genannt „Flasher“, dient zur Konfiguration der Firmware und dem Programmieren der Firmware auf die Hardware.



Abbildung 4.1: Architektur des Paper-Tracker

## 4.2 Komponentenarchitektur

In den folgenden Abschnitten wird detaillierter auf die einzelnen Komponenten des in Abschnitt 4.1 beschriebenen Systems eingegangen. Dabei werden die Komponenten in derselben Reihenfolge beschrieben, in welcher die Daten im Gesamtsystem fließen.

### 4.2.1 Tracking mit IoT-Hardware

Zur Erfassung der Standortdaten der Dokumente werden, wie in Abschnitt 3.2 beschrieben, Hardware-Tracker eingesetzt. Diese verwenden jedoch keine absolute Positionierungs-technologie wie beispielsweise GPS, da das hierfür benötigte Satellitensignal in Gebäuden zu schwach, und somit die Positionierung fehlschlagen kann.

Für die Lokalisierung von Geräten innerhalb eines Gebäudes gibt es daher mehrere Ansätze. Eine Möglichkeit ist es, ein Mesh-Netzwerk aus den Tracker-Geräten aufzubauen. Ist die Position einiger weniger Knoten des Meshes bekannt, kann durch die Signalstärke zu umliegenden Knoten oder über die sogenannte Time-of-Arrival bestimmt werden, wo sich ein Knoten relativ zu anderen Knoten und damit auch zu den Knoten mit bekannter Position befindet. Mit dieser Methode kann eine Genauigkeit von bis zu einem Meter bei der Lokalisierung erzielt werden. (vgl. [5])

Eine weitere Möglichkeit ist, auf bestehende Infrastruktur zurückzugreifen. WLAN ist heutzutage in praktisch jedem Gebäude verfügbar. In Gebäuden, in welchen es mehrere APs gibt, kann die Position dadurch bestimmt werden, dass der Tracker auswertet, von welchen APs er ein Signal empfängt und wie stark der Empfang zum jeweiligen AP ist. Da die Positionsdaten ohnehin zur Auswertung an das Backend übertragen werden müssen, wofür eine WLAN-Verbindung notwendig ist, wird diese Methode eingesetzt. Die gesammelten Daten werden dann im Anschluss an das Backend übermittelt.

Als Metrik für das Tracking über WLAN können die in Abschnitt 2.2 erläuterten RSSI oder RPCI verwendet werden. Da diese Werte jedoch nicht unbedingt proportional zum Abstand zum AP wachsen oder fallen, da Funksignale durch unterschiedliche Materialien unterschiedlich gedämpft werden, können sie nicht als absolut angesehen und verwendet werden.

Aus diesem Grund muss für jeden Raum die vorherrschende WLAN-Umgebung in das

Es wäre schön,  
Glossar-  
Referenzen  
zu kenn-  
zeichnen  
(oft mit  
Glos, oder  
Gls)

Quelle für  
die Genauig-  
keit von  
GPS in Ge-  
bäuden

Ist das hier  
eher Analy-  
se?

Glossar:  
Mesh-  
Netzwerk

Time-of-  
Arrival er-  
läutern

Papers  
zu RSSI-  
Basiertem  
Tracking  
einfügen  
(Microsoft)

System „eingelernt“ werden. Zu diesem Zweck gibt es einen eigenen Modus in der Anwendung, in welchem ein ausgewählter Tracker ständig die verfügbaren APs und die zu diesen ermittelte Signalstärke an den Server sendet. Dieser sammelt die Messwerte und aggregiert sie nach Abschluss der Messung, also nach einem definierten Zeitintervall. Aggregiert werden die Metriken so, dass aus den RSSI-Werten für jeden AP das Minimum und Maximum, sowie das erste und dritte Quartil und Median und arithmetisches Mittel berechnet werden.

Soll nun ein Tracker lokalisiert werden, scannt dieser die aktuelle WLAN-Umgebung und sendet die Scanergebnisse an den Server. Dieser berechnet nun für jeden bereits eingelernten Raum einen *Score*. Der Score berechnet sich aus der Übereinstimmung der nun gemessenen RSSIs mit den vorher eingelernten. Hierfür wird über jeden für den Raum ermittelten AP iteriert und abhängig davon, in welches statistische Maß der neue Messwert fällt, Punkte vergeben. Die konkrete Punkteverteilung ist hierbei noch nicht spezifiziert, es handelt sich um Parameter, welche genutzt werden können, um die Genauigkeit des Trackings zu optimieren. Daher werden diese in Kapitel 5 beschrieben.

Lifecycle  
des Paper-  
Trackers  
erläutern

## 4.2.2 Backend-Anwendung

### UML Daten Modellierung

Da die Backend-Anwendung die zentrale Komponente des Systems ist und auch die Daten verwaltet, werden für sie die Daten-Klassen modelliert. Das resultierende UML-Diagramm ist in Abbildung 4.2 abgebildet.

Die zentrale Klasse des Diagramms bildet die „Tracker“-Klasse. Sie modelliert einen Hardware-Tracker. Einige Attribute des Trackers sind eine ID, das Label, der Batteriestatus, eine Flag, ob ein niedriger Batteriestatus notifiziert wurde und einige Zeitsstempel. Die Zeitstempel geben zum Beispiel an, wann zuletzt ein Kommando abgeholt wurde oder wann zuletzt der Batteriestatus aktualisiert wurde. Mit Hilfe der letzten dem Tracker gesendeten Zeit zum Schlafen kann mit der Funktion „GetSecondsToNextPoll()“ abgefragt werden, wann sich der Tracker vorraussichtlich das nächste Kommando abholt. Neben den Attributen besitzt sie eine Referenz auf einen Status. Dieser Status gibt an, zu was der Tracker zur Zeit verwendet wird und in welchen anderen Zustand gewechselt werden kann.

Es gibt vier verschiedene Zustände, die in Integern kodiert sind:

#### **Idle = 1**

Tracker ist frei verfügbar und kann in „Learning“ oder „Tracking“ übergehen.

#### **Learning = 2**

Tracker wird für das Lernen eines Raumes verwendet und kann bei erfolgreichem Abschluss in „LearningFinished“ oder bei Abbruch in „Idle“ übergehen.

#### **LearningFinished = 3**

Das Lernen eines Raumes mit diesem Tracker wurde beendet. Die Ergebnisse können berechnet und einem Raum zugewiesen werden. In Folge darauf, geht der Tracker in den Zustand „Idle“ über.

#### **Tracking = 4**

Der Tracker besitzt aktuell einen Workflow, den er trackt. Nach Abschluss dieses Workflows oder nach Abbruch, geht der Tracker in den Status „Idle“ über.

Weiter besitzt der Tracker eine Referenz auf einen aktiven Workflow und den aktuellen Raum. Beide Referenzen können nichtexistent sein, falls der Tracker keinen aktiven Workflow besitzt oder der Raum nicht bestimmbar ist.

Ein „Room“ ist ein physikalischer Raum, der von einem Tracker erkannt werden soll. Neben einem Identifizierer und einem „Label“ (Name) für den Raum werden zusätzlich mehrere „BSSIDTrackingData“ gespeichert. Diese beinhalten für einen spezifischen „Access Point“/BSSID einige Messdaten zu der gemessenen Signalstärke. Über diese kann über eine Antwort des Trackers mit „ScanResults“ die Position des Tracker bestimmt werden.

Damit der Tracker eine Antwort sendet, wird diese mit Hilfe eines Kommandos angefragt. Diese Kommandos werden jeweils in einer Instanz der „Command“-Klasse abgebildet. Die Kommandos besitzen einen spezifischen Typ der Enumeration „ CommandType“, der die Aktion des Kommandos bestimmt. Auch wird pro Kommando eine „SleepTimeSec“ festgelegt, die bestimmt wie lange der Tracker sich nach der Aktion abschalten darf.

Spezielle Antworten der Tracker auf Kommandos und auch des Servers auf Anfragen der App sind in dem Unterpaket „communication“ modelliert. Dies ist zum Beispiel eine Fehler-Antwort und eine Basisklasse für Antworten des Trackers. Diese „TrackerResponse“-Klasse besitzt auch ein Feld für den aktuellen Batteriestand des Trackers, um diesen überwachen zu können.

Unter den Klassen „WorkflowTemplate“ und „WorkflowExec“ sind die Workflows modelliert. „WorkflowTemplate“ stellt eine Blaupause dar, wie ein Workflow aussieht. In diesem werden die einzelnen Schritte („Steps“) des Workflows festgelegt. Für einen „Step“ werden ein Label und eine Referenz auf den dazugehörigen Raum gespeichert. Über die Klasse „NextStep“ werden Referenzen auf nachfolgende Schritte modelliert. Die „NextStep“ Klasse besitzt ein Attribut „DecisionLabel“. Ist diese ein leerer String, so ist dieser verbundene Schritt ein linearer. Mit einem nicht-leeren String, gibt es mit diesem Schritt verschiedene Auswahlmöglichkeiten. Durch das transitive Attribut „StepEditingLocked“ der „WorkflowTemplate“-Klasse, wird die Bearbeitung blockiert. Dieses Attribut ist gesetzt, sobald es eine Ausführung des Templates gab.

Die „WorkflowExec“-Klasse stellt einen konkreten Workflow in der Durchführung dar. Als Attribute besitzt diese Klasse ein Label, einen Status und Start-/Endzeit. Der Status ist entweder „Running“, „Completed“ oder „Canceled“. Weiter besitzt die Klasse eine Referenz auf das Template, das ausgeführt wird, eine Menge an „ExecStepInfo“ und an den aktuellen Schritt. Die Klasse „ExecStepInfo“ besitzt Informationen über einzelne Schritte des Workflows. Dies sind zum einen die Start-/Endzeit des Schrittes sowie, ob der Schritt übersprungen wurde, und zum anderen die „Decision“-Information. Der „Decision“ String wird im Falle von mehreren Auswahlmöglichkeiten in einem Workflow dazu genutzt, in der Ausführung den richtigen Weg zu wählen.

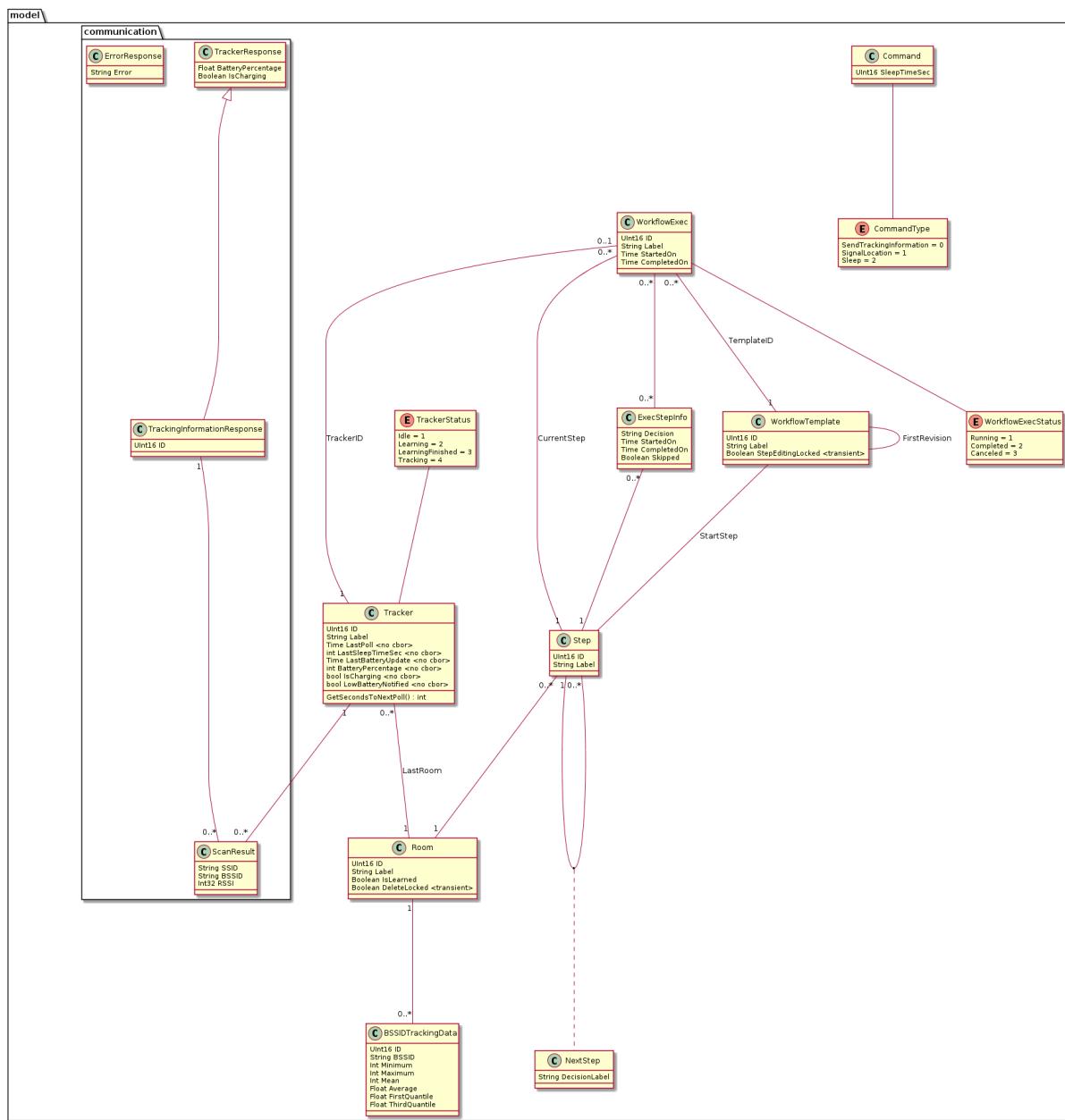


Abbildung 4.2: UML-Diagramm für die Backend-Anwendung

## Schnittstellen

Für die Backend-Anwendung sind zwei verschiedene Schnittstellen vorgesehen. Eine Schnittstelle ist für den Tracker und eine für die App vorgesehen. Die unterschiedlichen Schnittstellen sollen die aus der Analyse entwickelten Ansprüche entsprechend erfüllen.

**Tracker-Schnittstelle** Für die Schnittstelle des Trackers bedeutet dies, dass sie möglichst simpel und effizient sein muss. Dies wird über ein entsprechendes Protokoll und einem Datenformat erreicht. Das verwendete Protokoll ist das Constrained Application Protocol (CoAP) Protokoll und das Datenformat ist Concise Binary Object Representation (CBOR).

Bei Constrained Application Protocol (CoAP) handelt es sich um ein spezialisiertes Protokoll für die Verwendung in Internet of Things (IoT)-Hardware, welches dafür optimiert wurde möglichst wenig Overhead zu haben.

Im IoT-Bereich haben sich einige Protokolle durchsetzen können. Dazu zählen vor allem Hypertext Transfer Protocol (HTTP), Message Queuing Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP) und CoAP. Da die Tracker batteriebetrieben sind, ist für diesen Anwendungsfall besonders die Performance des eingesetzten Protokolles wichtig. Je weniger Overhead das Protokoll aufweist, desto weniger Instruktionen müssen vom Prozessor des Trackers durchgeführt werden, was zu einer längeren Laufzeit führt. Werden die Protokolle anhand dieses Aspektes verglichen, erweist sich CoAP als effizientestes Protokoll. (vgl. [6], [7])

Concise Binary Object Representation (CBOR) ist ein Datenformat, das auf kleine Code- und Nachrichtengrößen optimiert ist. Demnach ist es auch für der IoT-Bereich sehr gut geeignet. Vom Aufbau des Formats, ist es sehr gut mit JavaScript Object Notation (JSON) zu vergleichen, da es auch Schlüssel-Wert-Paare verwendet. Der große Unterschied ist jedoch, dass CBOR binär kodiert wird und JSON in Text kodiert wird. Dadurch ist CBOR nicht menschenlesbar aber sehr gut maschinenlesbar. Bei einem Vergleich zwischen der CBOR Bibliothek „libCBOR“ und der JSON Bibliothek „ArduinoJson“ konnte zudem festgestellt werden, dass bei gleicher Nachricht das CBOR Format ca. 26.5% kleinere Nachrichten in 83.3% der Zeit schnellerer Zeit serialisiert hat.

Die Tracker-Schnittstelle soll drei Endpunkte zur Verfügung stellen. Diese sind in Tabel-

Quellen verwenden, die in diesem Paper zitiert werden

Hier das Polling-Prinzip oder unter Hardware?!

Verb	Pfad	Aktion	Rückgabe-Wert
POST	/tracker/new	Neuen Tracker erstellen	Tracker ID
GET	/tracker/poll	Kommando für Tracker abfragen	Tracker Kommando
POST	/tracker/tracking	Ergebnisse des Tracking speichern	-

Tabelle 4.1: Verfügbare Endpunkte der Tracker-Schnittstelle

le 4.1 aufgelistet. Bis auf den Endpunkt zum erstellen eines neuen Trackers, erhalten alle Endpunkte als Parameter die Tracker ID des Trackers, der die Anfrage stellt. Der Endpunkt zum Speichern der Tracking Ergebnisse hat zudem als Parameter die Ergebnisse selbst.

**App-Schnittstelle** Die Schnittstelle für die App hat keine besonderen Anforderungen und kann daher frei gewählt werden. Aus diesem Grund werden Protokolle und Datenformate verwendet, die momentan dem aktuellen Stand der Technik entsprechen. Dies ergibt sich der Vorteil, dass sie mit allen anderen gewählten Technologien sehr gut kombinierbar sind und entsprechende Bibliotheken vorhanden sind. Das verwendete Protokoll ist das HTTP-Protokoll und das Datenformat ist JSON. Die Schnittstelle basiert auf dem Representational State Transfer (REST) Paradigma (vgl. [8]). Alle verfügbaren Endpunkte der App-Schnittstelle sind in Tabelle 4.2 dargestellt.

Quelle für aktuelle Protokolle

Soll REST erläutert werden?

Tabelle 4.2: Liste aller Endpunkte der App-Schnittstelle

Verb	Pfad	Aktion	Parameter	Rückgabe-Wert
<i>Rooms</i>				
GET	/room	Auflistung der Räume	-	Liste aller Räume
POST	/room	Erstellen eines neuen Raums	Raum	Erstellter Raum
PUT	/room/:id	Raum aktualisieren	Raum	Aktualisierter Raum
DELETE	/room/:id	Raum löschen	-	-
<i>Tracker</i>				
GET	/tracker	Auflistung der Tracker	-	Liste aller Tracker
PUT	/tracker/:id	Tracker aktualisieren	Tracker	Aktualisierter Tracker
DELETE	/tracker/:id	Tracker löschen	-	-
GET	/tracker/:id/next_poll	Zeit bis nächste Kommando Abholung	-	Sekunden
POST	/tracker/:id/learn	Lernen mit Tracker starten	-	Lenzeitz
GET	/tracker/:id/learn	Lernstatus abfragen	-	Abgeschlossen
DELETE	/tracker/:id/learn	Lernen abbrechen	-	Gefundene SSIDs
POST	/tracker/:id/learn/finish	Lernen abschließen	Raum ID SSIDs	-
GET	/tracker	Auflistung der Tracker	-	Liste aller Tracker

Verb	Pfad	Aktion	Parameter	Rückgabe Wert
<i>Workflows</i>				
GET	/workflow/template	Auflistung der Templates	-	Liste aller Templates
POST	/workflow/template	Template erstellen	Template	Erstelltes Template
PUT	/workflow/template/:id	Template aktualisieren	Tempalte	Aktualisiertes Template
DELETE	/workflow/template/:id	Template löschen	-	-
POST	/workflow/template/:id/start	Start Schritt erstellen	Step	Erstellter Step
POST	/workflow/template/:id/step	Schritt hinzufügen	Step	Erstellter Step
GET	/workflow/template/:id/step/:id	Schritt abfragen	-	Step
PUT	/workflow/template/:id/step/:id	Schritt aktualisieren	Step	Aktualisierter Step
DELETE	/workflow/template/:id/step/:id	Schritt löschen	-	-
POST	/workflow/template/:id/step/:id/move	Schritt verschieben	Richtung	-
POST	/workflow/template/revision	Template Revision erstellen	Label	Erstellte Revision
GET	/workflow/exec	Auflistung der Ausführungen	-	Execution
POST	/workflow/exec	Ausführung erstellen	Execution	Erstelle Execution
POST	/workflow/exec/:id/progress/:id	Ausführung zu Schritt progressieren	-	-
POST	/workflow/exec/:id/cancel	Ausführung abbrechen	-	-
<i>Sonstiges</i>				
GET	/export.xlsx	Datenexport für Analyse	-	Excel-Datei

## Software-Architektur

Die Architektur des Backend-Servers ist in folgendem Klassendiagramm 4.3 dargestellt.

Generell kann die Architektur in drei verschiedene Schichten aufgeteilt werden. Die erste Schicht bildet das Paket „router“, die zweite Schicht das Paket „managers“ und die dritte Schicht das Paket „repositories“.

Erstellung  
der DB Ver-  
bindung  
reinpacken

Im „router“ Paket befinden sich die Router für die zwei externen Schnittstellen des Servers. Die HTTP Schnittstelle für die App befindet sich in der „HttpRouter“-Klasse und die CoAP Schnittstelle für den Tracker befindet sich in der „CoapRouter“-Klasse. Beide Klassen besitzen eine Methode „Serve“. Mit dieser Methode werden die Router letztendlich gestartet so, dass sie Verbindungen annehmen können. Als Parameter bekommen sie die Adresse, auf die sie Verbindungen annehmen, und im Falle des „CoapRouter“ auch das Transport-Protokoll.

Das „managers“ Paket beinhaltet sechs verschiedene Manager, die die Businesslogik für verschiedene Teilbereiche des Servers beinhalten. Ein Beispiel ist der „TrackerManager“, über den Tracker angelegt, abgefragt, verändert und auch gelöscht werden können. Weitere Methoden für z.B. das Abfragen eines Kommandos für einen Tracker sind auch beinhaltet. Alle Manager sind als Singleton implementiert so, dass sie sich auch untereinander problemlos aufrufen können. Konfiguriert werden die Manager über die global verfügbare „Config“ Klasse. Über diese Klasse können verschiedene Einstellungen über einzigartige Identifizierer abgefragt werden.

Spezielle  
Fkt einzel-  
ner Manager  
erklären?

Die unterste Schicht bildet das „repositories“ Paket. Dieses enthält verschiedene Datenbank-Repositories, die die Datenbank und damit die Datenpersistenz des Servers abbilden. Alle Methoden eines Repository bilden direkte Anfragen an die Datenbank. Dabei ist ein Repository meist für eine Modellklasse verantwortlich.

Die main-Klasse ist für das Starten des Servers und das Erstellen der einzelnen Komponenten verantwortlich. In dieser wird zum Beispiel auch die Konfiguration eingelesen und initialisiert. In der Utils-Klasse sind einige verschiedene Hilfsmittel untergebracht. Dies sind zum einen das Versenden von E-Mails und zum anderen einige mathematische Hilfsfunktionen.



Abbildung 4.3: Software-Architektur für die Backend-Anwendung

## Daten-Export

Für einen Daten-Export muss zuerst ausgewählt werden, welche Daten exportiert werden sollen. Da der Export des Paper-Trackers bei einer Optimierung der Workflows helfen soll, müssen hauptsächlich Daten der Klassen „WorkflowTemplate“/„WorkflowExecution“ und deren Umfeld exportiert werden. Konkret wären dies die Templates, deren Ausführungen inklusive detaillierter Infos der Schritte („ExecStepInfo“). Diese Informationen können helfen, langsame oder nicht benötigte Schritte eines Workflows zu identifizieren.

Weiter sollen auch Informationen zu den Revisionen eines Templates exportiert werden. Anhand dieser Informationen kann verfolgt werden, ob Anpassungen eines Workflows einen Erfolg gebracht haben. Dazu müssen zu den Revisionen einige Metriken erfasst werden. Die zu erfassenden Metriken sind:

- Anzahl der Ausführungen
- Prozentzahl der erfolgreich abgeschlossenen Ausführungen
- Durchschnittliche Dauer einer Ausführung

Mit der Auswahl der Daten kann auch das Format des Exports ausgewählt werden. Alle zu exportierenden Daten können sehr gut in Tabellen exportiert werden. Dies führt zu hauptsächlich zwei Exportformaten: Comma Separated Values (CSV) und Excel.

CSV ist ein weit verbreitetes und simples Format, da es hauptsächlich mit Kommas (',') und Zeilenumbrüchen arbeitet. Dadurch ist es sehr einfach zu erstellen und einzulesen. Die meisten Tools, die mit solchen Daten arbeiten, haben die Möglichkeit CSV-Dateien einzulesen. Ein großer Nachteil von CSV ist jedoch, dass es nicht standardisiert ist und mehrere Arten mit kleinen Unterschieden existieren [9]. Im RFC-4180 werden lediglich scheinbar übereinstimmende Teile der verschiedenen Implementierungen dargestellt. Nach diesen gibt es einen weiteren Nachteil: Pro CSV-Datei kann nur eine Tabelle gespeichert werden. Dies würde für den Export des Paper-Trackers bedeuten, dass entweder mehrere Dateien oder eine gebündelte Datei angeboten werden muss.

Excel oder genauer das von Excel verwendete „xlsx“-Format ist ein offenes und standardisiertes Format für Kalkulationstabellen [10]. Da es ein offenes Format ist, kann es auch von anderen Programmen, wie Libre Office, geöffnet und bearbeitet werden. Es muss nicht, wie bei CSV erst importiert werden. Auch unterstützt es mehrere Tabellen in einer

Datei, welches den Download des Exports vereinfacht. Aus diesen Gründen wird Excel als Exportformat für den Paper-Tracker verwendet.

### 4.2.3 App für Mobilgeräte

Die App für Mobilgeräte soll dem Endbenutzer die Möglichkeit geben, alles um den Paper-Tracker und die damit verbundenen Workflows einzusehen und zu bearbeiten. Sie soll dafür die oben erwähnte REST-Schnittstelle nutzen und deren Operationen dem Endbenutzer zur Verfügung stellen.

Die Oberfläche selbst soll möglichst intuitiv sein. Um dies zu erreichen, wurden als Entwurf für die App einige Design-Prototypen erstellt, welche die grundlegenden Elemente der App darstellen. Diese Prototypen wurden zum Teil per Hand auf Papier gezeichnet und zum Teil digital entworfen.

Alle Designs wurden größtenteils mit Designelementen entworfen, die im zu verwendenden User Interface (UI)-Framework vorhanden sind. Daher wurden auf Bausteine des Flutter-Frameworks<sup>1</sup> zurückgegriffen, welche sich an die von Google entworfene Material-Designsprache halten. (vgl. [11]) Im Folgenden werden einige dieser Prototypen vorgestellt.

#### Hauptseite

In Abbildung 4.4 ist der Prototyp der Hauptseite der App zu sehen. Die Hauptseite ist der zentrale Einstiegspunkt der App und bietet eine Übersicht über grundlegende Informationen.

Allgemein ist die Hauptseite in einen Kopfbereich und einen Inhaltsbereich aufgeteilt. Der Kopfbereich zeigt lediglich den Titel der App „Paper-Tracker“ an und einen Button, mit dem das Tutorial aufgerufen werden kann. Das Tutorial ist weiter in Abschnitt 4.2.3 beschrieben. Im Inhaltsbereich gibt es weiter eine Aufteilung in verschiedene Tabs, die jeweils für eine Klasse im Datenmodell stehen. Es soll einen Tab für Workflows in der Ausführung, Workflow-Templates, Räume und Tracker geben. Jeder Tab und damit jede Klasse soll ein eindeutiges Icon zugewiesen bekommen, mit dem die Klassen auf verschiedenen Seiten der App wiedererkannt werden können. Die Tabs sollen über ein Wischen

---

<sup>1</sup><https://flutter.dev/docs/development/ui/widgets>

nach links und rechts oder über ein Auswählen in der Tableiste gewechselt werden können.

In den Tabs wird eine Liste aller Objekte der entsprechenden Klasse mit den wichtigsten dazugehörigen Informationen angezeigt. Dies ist zum Beispiel für einen Tracker der aktuelle Raum oder der Batteriestatus.

Über ein Pull-Down<sup>2</sup> der jeweiligen Liste, kann diese aktualisiert werden. Wenn der Nutzer einen Eintrag der Liste auswählt, soll sich eine Detailseite zu diesem Objekt öffnen. Ein Design-Prototyp dazu ist in Abschnitt 4.2.3 dargestellt.

Unten rechts in der Ecke ist ein „Floating Action Button“ untergebracht. Über diesen soll ein Dialog geöffnet werden, in dem ein neues Objekt der entsprechenden Klasse erzeugt werden kann. Dazu ist weiter unten in Abschnitt 4.2.3 auch ein Design-Prototyp beschrieben.

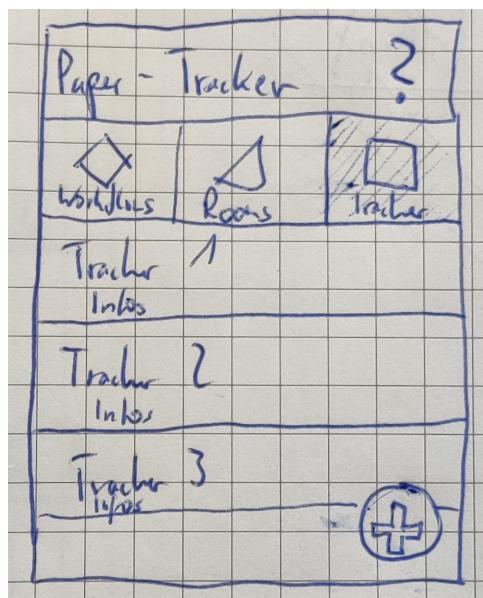


Abbildung 4.4: Design-Prototyp der Hauptseite

## Darstellung von Workflows

Abbildung 4.5 zeigt den Prototyp für die Darstellung von Workflows. Dies können sowohl Workflows in der Ausführung als auch Templates sein.

<sup>2</sup>Herunterziehen der Liste über den eigentlichen Inhalt hinaus.

Die einzelnen Schritte eines Workflows sollen als Boxen in einer Liste dargestellt werden. Der Nutzer muss in dieser Liste scrollen können, falls es in einem Workflow zu viele Einzelschritte gibt, um diese auf einem Bildschirm anzeigen zu können. Als letzte Box in der Liste wird ein Button angezeigt, mit dem der Nutzer einen neuen Schritt an die Liste anhängen kann. Die Eingabe der Informationen des neuen Schritts soll in einem Dialog erfolgen. Diese Box wird nur angezeigt, falls der anzugezeigende Workflow bearbeitbar ist.

In den Boxen zu den Schritten sollen einige Informationen angezeigt werden. So werden beispielsweise der Name des Schrittes, der dem Schritt zugewiesenen Raum und die verfügbaren Optionen dargestellt. Die Optionen sind jeweils einzelne Buttons. Durch ein Auswählen eines Options-Buttons, werden die zu diesem Button gehörenden Schritte unter dem Schritt eingerückt angezeigt. Diese Schritte werden wie oben beschrieben in einer identischen Liste angezeigt und unterstützen auch das Einfügen neuer Schritte.

Ist schon irgendwo erläutert, wann ein Workflow nicht mehr bearbeitbar ist?

## **Detailseite für Räume**

In Abbildung 4.6 ist der Design-Prototyp für eine Detailseite dargestellt. Beispielhaft wird hier die Tracker-Detailseite behandelt.

Im Kopfbereich der Seite ist neben einem Button für die Navigation das Icon für die Klasse Tracker und der Name des Trackers dargestellt. Der Inhaltsbereich der Seite beinhaltet eine Liste mit allen wichtigen Attributen des Trackers. Jeder Eintrag besteht aus dem Namen des Attributs links und dem Attribut selbst rechts. Wenn möglich soll das Attribut selbst in einem modifizierbaren Feld dargestellt werden, das vorerst gesperrt ist.

Der Fußbereich einer Detailseite besteht aus zwei Buttons. Mit dem linken Button können die editierbaren Attribute freigeschaltet werden. Dabei wird aus dem „Stift“-Icon ein „Speichern“-Icon und über den Button können die bearbeiteten Attribute gespeichert werden. Der rechte Button dient zum Löschen des dargestellten Objekts.

Je nach Klasse des dargestellten Objekts ändert sich das Icon im Kopfbereich und die angezeigten Attribute. In folgender Tabelle 4.3 ist gelistet, für welche Klassen welche Attribute angezeigt werden und, ob diese editierbar sind.

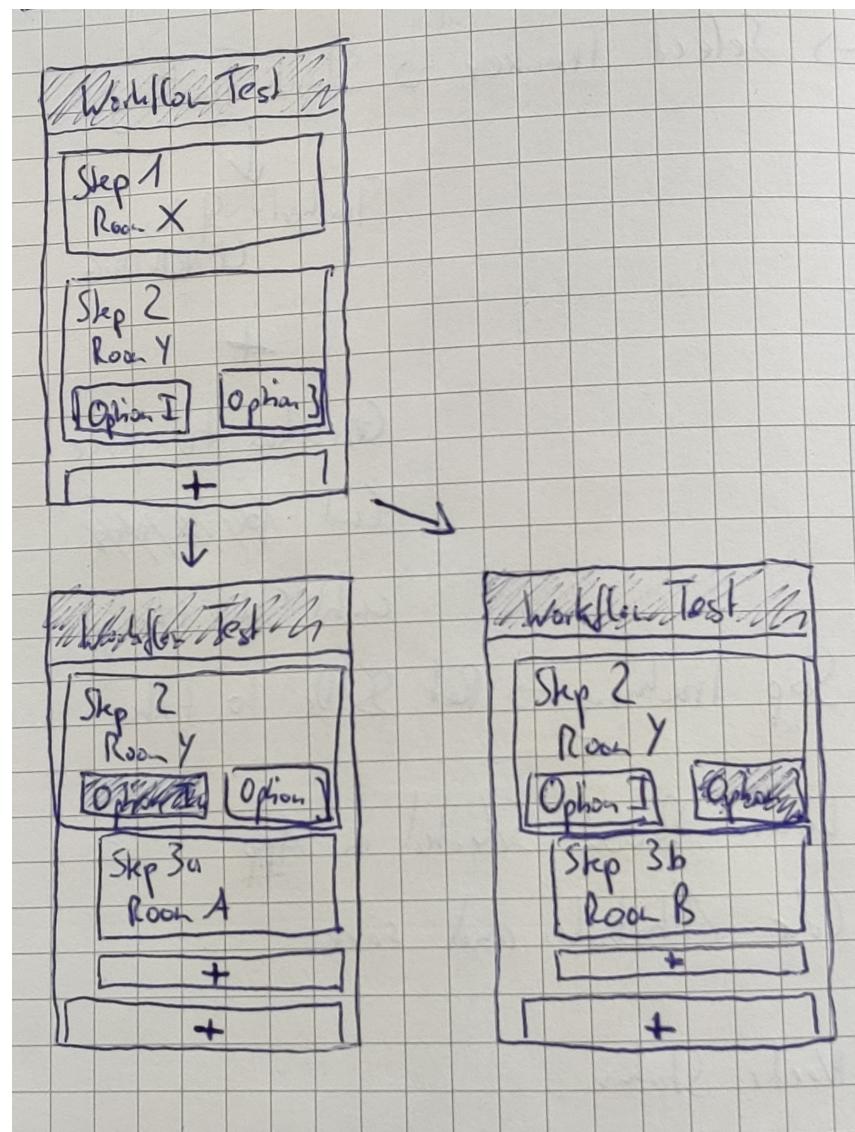


Abbildung 4.5: Design-Prototyp der Darstellung von Workflows

Klasse	Attribut	Editierbar
Tracker	Label	Ja
	Raum	Nein
	Batterie	Nein
	Status	Nein
Raum	Label	Ja
	Eingelernt	Über Button
WorkflowTemplate	Label	Ja
	Initiale Revision	Nein
	Schritte	siehe Workflow-Darstellung
WorkflowExec	Status	Nein
	Schritte	Nein

Tabelle 4.3: Dargestellte Attribute der Klassen

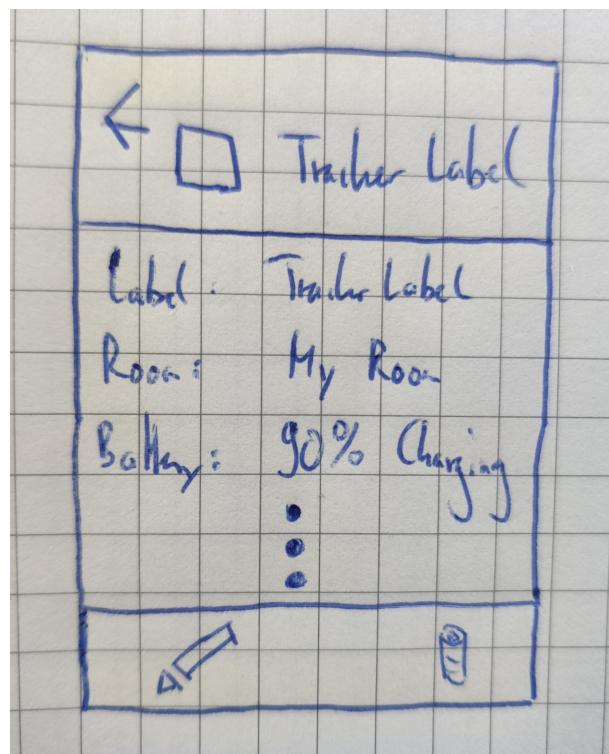


Abbildung 4.6: Design-Prototyp der Detailseite eines Trackers

### Seite zum Einlernen eines Raumes

Weiter ist in Abbildung 4.7 der Prototyp der Einlern-Seite dargestellt. Auf dieser Seite kann der Endbenutzer einen Raum einlernen. Dazu muss in zwei Dropdowns der zu lernende Raum und der zum Lernen verwendete Tracker ausgewählt werden. Wird von einer Detailseite eines Raumes oder Trackers auf die Lernseite navigiert, so wird dieses Objekt entsprechend in einem der Dropdowns vorausgewählt. Sind Raum und Tracker ausgewählt, kann der Einlernvorgang gestartet werden.

In diesem Fall wird statt dem Button zum Starten des Lernens eine Liste der gefundenen SSIDs angezeigt. Der Nutzer kann in dieser Liste auswählen, welche SSIDs zum Einlernen des Raumes verwendet werden sollen. Die Liste wird während des Vorgangs stetig mit neu gefundenen SSIDs erweitert. Sobald der Vorgang beendet wird, kann am Ende der Seite das Einlernen über einen Button abgeschlossen werden. Wird dieser betätigt, wird von der Seite automatisch auf die vorherige zurück navigiert.

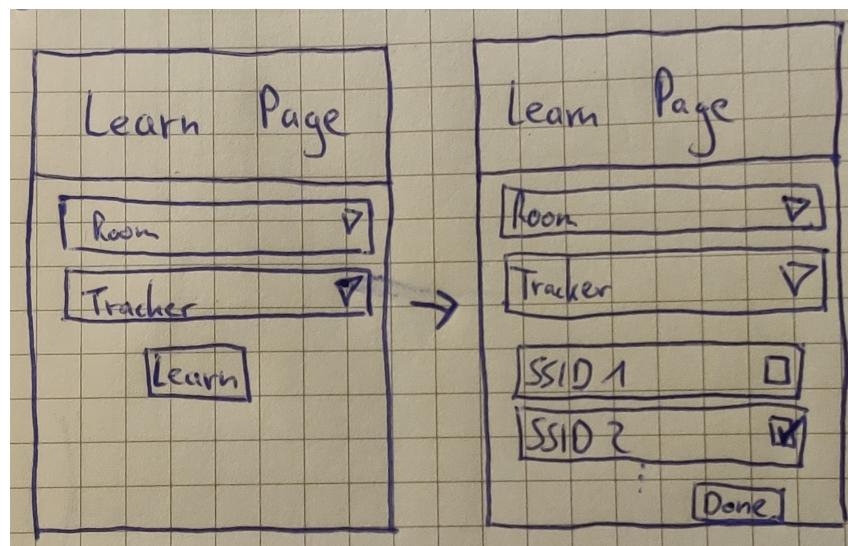


Abbildung 4.7: Design-Prototyp der Einlern-Seite

### Dialoge zum Erstellen von Objekten

Das Hinzufügen eines neuen Objektes einer Klasse kann über den „Floating Action Button“ auf der Hauptseite angestoßen werden. Durch den Button soll ein Dialog geöffnet werden, deren Design-Prototyp in Abbildung 4.8 dargestellt ist.

Der Dialog wird durch ein Pop-up realisiert. Dieses hat eine Überschrift, die angibt welches Objekt erzeugt wird. Unter der Überschrift können in einer Liste alle benötigten Attribute des Objekts eingegeben werden. Wenn möglich, wie zum Beispiel bei der Auswahl eines Raumes, soll eine Dropdown-Auswahl verwendet werden. Zu dem Dropdown soll auch wieder das Icon der Klasse, aus welcher ein Objekt ausgewählt werden soll, dargestellt werden.

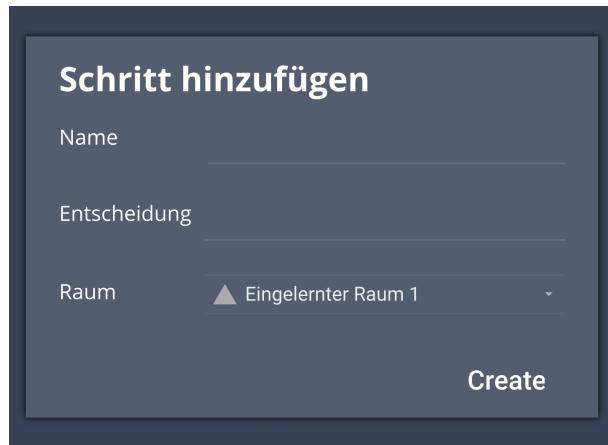


Abbildung 4.8: Design-Prototyp eines Pop-ups zum Erstellen eines Raumes

## Tutorial-Seite

Beim ersten Start der App soll automatisch ein Tutorial gestartet werden, welches die Bedienung der App erklärt. Der Nutzer soll dieses Tutorial auch von der Hauptseite aus aufrufen können.

Das Layout des Tutorials ist schlicht gehalten. Im Zentrum ist eine Box dargestellt, in der es ein Titel, ein Bild und eine Beschreibung gibt. Durch eine Liste an solchen Boxen kann durch ein Wischen nach links und rechts geblättert werden. Jede Box soll mit ihrem Inhalt einen Aspekt der App näher erläutern und dem Nutzer erklären. Unter der Box wird dem Nutzer signalisiert, wie weit dieser durch die Liste fortgeschritten ist, indem der Index der aktuellen Box und die Gesamtzahl der Boxen angezeigt werden. Am unteren Teil der Seite gibt es einen Button, mit dem das Tutorial beendet wird.

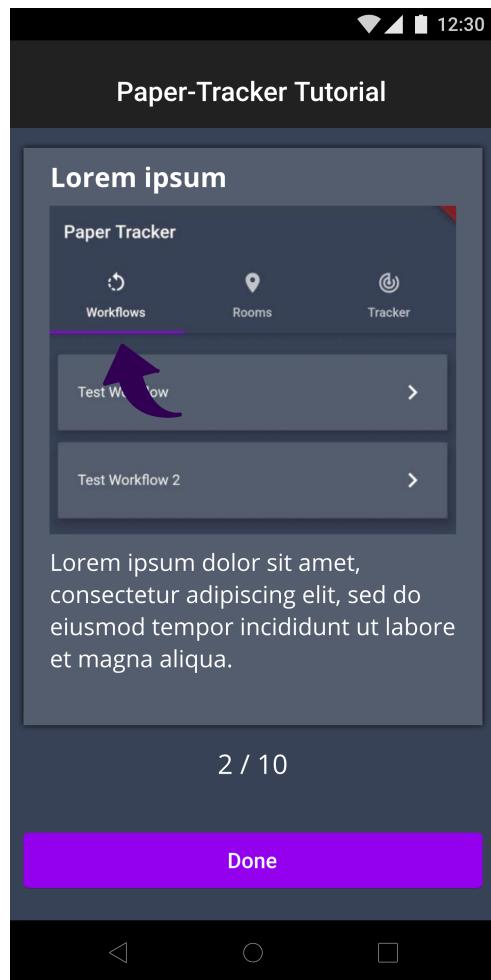


Abbildung 4.9: Design-Prototyp des Tutorials

Der Inhalt des Tutorials soll alle wichtigen Funktionen der App und auch die Konfiguration des Trackers selbst abdecken. Dafür sind die folgenden Inhalte angedacht:

1. Konfiguration der App mit der Backend-Server URL mit Protokoll
2. Hauptseite mit Unterteilung in vier Tabs und des Buttons für das Tutorial
3. Tracker-Tab mit leerer Liste an Trackern und Hinweis, dass über den Flasher ein Tracker konfiguriert werden kann
4. Pull-Down der Liste zum Aktualisieren. Der zuvor konfigurierte Tracker erscheint
5. Tracker-Detailseite und das Bearbeiten eines Trackers
6. Raum-Liste und das Erstellen eines neuen Raumes
7. Funktionsweise des Einlernen eines Raumes
8. Template-Liste und das Erstellen eines Templates
9. Editieren eines Templates in der Detailseite des Templates
10. Erstellen einer neuen Revision eines Workflow Templates
11. Workflow-Ausführungs-Liste und das Starten einer neuen Ausführung
12. Menü mit Aktionen zu einem Schritt in einer Workflow-Ausführung

### Icons für zentrale Klassen

Wie schon zu der Hauptseite und Detailseite angemerkt, gibt es für jede zentrale Klasse ein zugehöriges Icon. Die soll ermöglichen, dass der Endbenutzer in der App über die Icons die dazugehörige Klasse wiedererkennen kann. So kann direkt erkannt werden, welcher Objekttyp auf der Detailseite angezeigt wird, oder welche Objekte in einem Dropdown ausgewählt werden. Zusätzlich kann das Icon abgeändert werden, wenn sich zum Beispiel der Zustand eines Objektes ändert. Dies wird bei der Workflow-Ausführung verwendet, die ein anderes Icon besitzt, falls die Ausführung abgeschlossen ist. Eine Übersicht der zu verwendenden Icons ist in Abbildung 4.10 gegeben.

Studie über Nutzen von Icons zur Wiedererkennung von Schlüsselwörtern einbringen

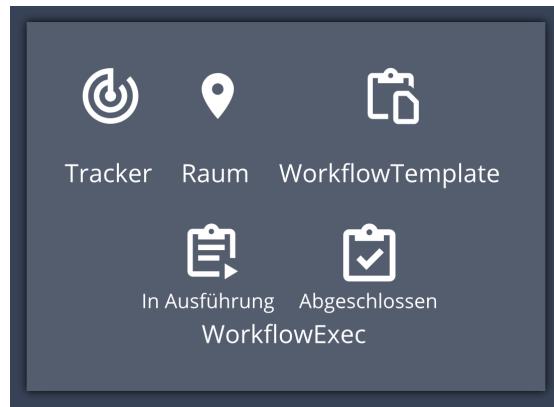


Abbildung 4.10: Icons für Klassen

#### 4.2.4 Flasher-Tool

Da das Flasher-Tool nicht zu den Hauptkomponenten des Paper-Trackers zählt, gibt es keinen konkreten Entwurf zum UI-Design. Trotzdem muss konkret definiert werden, welche Funktionalitäten das Tool beinhaltet.

Das Tool muss plattformübergreifen mit möglichst wenigen externen Abhängigkeiten gestartet werden können. Mit dem Start der Anwendung soll der Nutzer in einem Graphical User Interface (GUI) die Konfigurationsdaten für die Firmware eingeben können. Die benötigten Konfigurationswerte sind in der folgenden Liste dargestellt:

##### **WLAN SSID**

Name des WLAN Netzwerkes, mit dem sich der Tracker verbinden soll

##### **WLAN Nutzername**

Nutzername, mit dem sich der Tracker am WLAN anmelden soll. Diese Konfiguration kann freigelassen werden, falls nur ein Passwort für die Anmeldung ausreicht.

##### **WLAN Passwort**

Passwort, mit dem sich der Tracker am WLAN anmelden soll.

##### **Backend-Server Internet Protocol (IP)**

Die IP-Adresse, unter welcher der Backend-Server aus dem angegebenen WLAN-Netz erreichbar ist.

Neben den Konfigurationsdaten müssen noch der Ort der Firmware auf dem ausführenden

Computer und der Port, an welchem die Hardware verbunden ist, angegeben werden.

Nachdem alle Konfigurationsdaten angegeben sind, soll das Tool die Hardware mit den Werten programmieren. Dabei soll dem Nutzer der aktuelle Prozess in Form eines Fortschrittsbalken oder durch Anzeige von Protokollen visualisiert werden.

# 5 Implementierung

## 5.1 Backend-Server

### 5.1.1 Verwendete Technologien

Die grundlegend verwendete Technologie für den Backend-Server ist die Programmiersprache „Go“. Go wurde ausgewählt, da es eine einfach zu verstehende Programmiersprache ist, die optimiert für Server-Anwendungen ist (vgl. [12]). Zudem sind alle am Projekt beteiligten Personen schon mit der Sprache vertraut.

Go stellt eine gute Standardbibliothek zur Verfügung, die an vielen Stellen verwendet wird. Trotzdem werden zusätzlich einige externe Bibliotheken verwendet. Diese dienen hauptsächlich dazu, die externen Schnittstellen (HTTP und CoAP) mit den dazugehörigen Datenformaten zur Verfügung zu stellen. Auch werden Bibliotheken für ein einfacheres Testing verwendet.

Im Folgenden sind die verwendeten Bibliotheken aufgelistet:

#### **pflag (<https://github.com/spf13/pflag>)**

„pflag“ wird zum Angeben und Parsen von Kommandozeilenargumenten verwendet.

Es ist kompatibel mit „viper“.

#### **viper (<https://github.com/spf13/viper>)**

Für die Konfiguration des Servers wird „viper“ verwendet. Es kann Konfigurationen aus „pflag“, Dateien und mehr einlesen.

#### **logrus (<https://github.com/sirupsen/logrus>)**

„logrus“ ist strukturiertes Logging-Tool, dass für jegliche Logs innerhalb des Servers verwendet wird.

#### **gorm (<https://github.com/jinzhu/gorm>)**

„gorm“ ist ein Object-relational mapping (ORM) Framework, dass die Datenbankanbindung für den Server zur Verfügung stellt.

### **Gin (<https://github.com/gin-gonic/gin>)**

„Gin“ ist ein HTTP Framework, dass Routing, Parameter-Matching, etc. übernimmt. Zudem kann es standardmäßig JSON verstehen.

### **go-coap (<https://github.com/go-ocf/go-coap>)**

„go-coap“ ist ein CoAP Framework, dass hauptsächlich das Routing übernimmt. Einige weitere Aufgaben, die z.B. „Gin“ übernimmt, müssen selbst programmiert werden.

### **go-codec (<https://github.com/ugorji/go>)**

„go-codec“ stellt die Codierung für CBOR zur Verfügung.

### **now (<https://github.com/jinzhu/now>)**

„now“ stellt Hilfsfunktionen zur Verfügung für Zeitberechnungen.

### **stats (<https://github.com/montanaflynn/stats>)**

„stats“ stellt einige mathematische Funktionalitäten für statistische Berechnungen zur Verfügung.

### **xlsx (<https://github.com/tealeg/xlsx>)**

Mit „xlsx“ können Dateien im Microsoft Excel Open XML Spreadsheet (XLSX) Dateiformat erstellt, bearbeitet und gelesen werden. Im Server wird sie für das Erstellen des Excel-Export genutzt.

### **gomail (<https://github.com/go-gomail/gomail>)**

„gomail“ wird verwendet, um E-Mails verschicken zu können.

### **ginkgo (<https://github.com/onsi/ginkgo>)**

Mit „ginkgo“ können Tests in Behavior-driven development (BDD) Technik geschrieben werden.

### **gomega (<https://github.com/onsi/gomega>)**

„gomega“ ist eine Matching Library, die zu „ginkgo“ gehört. Sie stellt Vergleiche zwischen erwarteten Werten und tatsächlichen Werten zur Verfügung.

### **GoMock (<https://github.com/golang/mock>)**

„GoMock“ ist ein Mocking Framework, dass für verschiedene Tests verwendet wird.

Als Datenbank wird in Kombination mit „gorm“ eine „SQLite“ verwendet. Der große Vorteil ist, dass eine SQLite Datenbank eine einfache Datei ist und kein externer Server

aufgesetzt werden muss. Dies vereinfacht vor allem die Entwicklung. Zum Beispiel ist ein einfaches Zurücksetzen der Daten durch ein Löschen der Datei möglich.

In einem späteren Einsatz des Paper-Trackers kann dank der „gorm“ Bibliothek auch andere Datenbanken verwendet werden, die eher für einen dauerhaften und sichereren Einsatz vorgesehen sind. Möglich sind dafür „MySQL“, „PostgreSQL“, „Microsoft SQL Server“ und wie erwähnt SQLite. Im Code des Servers muss hierfür lediglich eine erweiterte Konfiguration ermöglicht werden. (vgl. [13])

### 5.1.2 REST-Schnittstelle

### 5.1.3 CoAP-Schnittstelle

### 5.1.4 Tracker-Lokalisierung

Wie in Unterabschnitt 4.2.1 beschrieben wird die Lokalisierung der Tracker mit einem Score-basierten Sysytem durchgeführt.

Dazu werden zunächst Scanergebnisse vom jeweiligen Tracker gesammelt und dann ausgewertet. Da die Datengröße der Scanergebnisse je nach Anzahl der vom Tracker sichtbaren WLAN-Netzwerken zu groß für ein einzelnes User Datagram Protocol (UDP)-Paket ist<sup>1</sup>, wurde das Application Programming Interface (API) so gestaltet, dass die Ergebnisse in mehreren Paketen an den Server übermittelt werden können. Dazu berechnet der Tracker im Vorraus, in wie viele Pakete die Scanergebnisse geteilt werden müssen. Diese Zahl, sowie ein für den Ergebnissatz einheitlicher, zufällig erzeugter Identifier (ID) sind jedem Teipaket angehängt. So kann der Server erkennen, ob alle Ergebnisse empfangen wurden und diese dann verwenden. Dass nicht alle Pakete eines Ergebnissatzes beim Server angekommen sind ist dadurch erkennbar, dass ein anderer ID empfangen wird. In diesem Fall werden alle vom vorherigen Paket empfangenen Scanergebnisse verworfen und die neuen Pakete gespeichert.

Sobald ein vollständiger Ergebnissatz empfangen wurde, wir aus diesem ein Score für jeden Raum ermittelt. Dieser Score ist eine positive Ganzzahl

Das hier  
kann viel-  
leicht zu  
CoAP-  
Schnittstelle  
verschoben  
werden

---

<sup>1</sup>Im Arduino-Framework gibt es eine Limitierung von 1446 Bytes für die Größe eines UDP-Paketes. (vgl. [14])

### 5.1.5 Daten-Export

In diesem Kapitel wird die Implementierung des Datenexports in eine Excel-Datei näher erläutert. Dies besteht aus größtenteils zwei Unterpunkten: Dem Sammeln und Aufbereiten der Daten für den Export und das Schreiben der Excel-Datei selbst. Beides wird zusammengefasst im „ExportManager“ durchgeführt.

Um die Daten für den Export zu sammeln, nutzt der „ExportManager“ einige andere Manager: „WorkflowTemplateManager“, „WorkflowExecManager“ und „RoomManager“. Über diese können alle benötigten Daten abgefragt werden. Zuerst werden alle Templates abgefragt. Für jedes Template werden alle verfügbaren Ausführungen des Templates angefordert. Weiter wird für jeden Schritt in einem Template/einer Ausführung Informationen über die Schritte und die den Schritten zugewiesenen Räumen abgefragt. Während die Daten gesammelt werden, werden auch einige Metriken erfasst: Die Anzahl Ausführungen pro Template, wie viele Ausführungen beendet wurden und wie lange eine durchschnittliche Ausführung gebraucht hat.

Nachdem alle grundlegenden Daten gesammelt wurden, wird nochmals durch alle Templates iteriert und diese nach Revisionen sortiert. Dies bedeutet, dass alle Templates zusammen gruppiert werden, die die gleiche ursprüngliche Revision haben. Die ursprüngliche Revision selbst wird auch mit gruppiert. Damit sind alle Daten entsprechend gesammelt und aufbereitet, um exportiert zu werden.

Für den Export selbst wird, wie schon in Unterabschnitt 5.1.1 angemerkt, die Bibliothek „xlsx“ verwendet. Die Entscheidung wurde für diese Bibliothek getroffen, da sie eine simple Schnittstelle hat und regelmäßig aktualisiert wird (Stand 20. April 2020: Letzte Aktualisierung vor 24 Tagen [15]).

„xlsx“ arbeitet mit Pointern auf Objekte verschiedener Typen, die jeweils ein Objekt in Excel wiederspiegeln. So gibt es zum Beispiel Objekte für eine Excel-Datei, ein Sheet in einer Datei und eine Zelle eines Sheets. Eine Datei kann über eine Funktion der Bibliothek erstellt werden, ein Sheet kann über das Datei-Objekt erstellt werden und ein Zell-Objekt über das Sheet-Objekt.

Für den Excel-Export des Paper-Trackers wird zunächst eine neue Datei erstellt. Diese ist nur im Hauptspeicher vorhanden und wird nicht auf die Festplatte geschrieben. Weiter wird für jedes Workflow Template ein Sheet erstellt. In dieses Sheet werden als Tabelle

alle Ausführungen dieses Templates mit Informationen zu den Schritten geschrieben. Dies geschieht schon während des Sammeln der Daten. Ein Beispiel für ein solches Sheet ist in Abbildung 5.1 zu sehen.

Label of execution	Status	Start Time	End Time	Step1 (Room1) Part of Exec	Step1Start Time	Step1 End Time	Step1 Decision	Step1 Skipped
Exec1	StatusCompleted	14/04/2020 12:15	14/04/2020 12:15	TRUE	14/04/2020 12:15			TRUE
Exec2	StatusCompleted	14/04/2020 13:09	14/04/2020 13:09	TRUE		14/04/2020 13:09		TRUE

Abbildung 5.1: Template-Sheet eines Excel-Exports

Für die Übersicht der Revisionen wird zusätzlich ein separates Sheet erstellt. In dieses werden nach dem aufbereiten der Daten, in einer Tabelle die verschiedenen ursprünglichen Revisionen mit ihren folgenden Revisionen gelistet. Hier werden auch die erfassten Metriken dargestellt. Dies resultiert in dem in Abbildung 5.2 dargestellten Sheet.

Original Revision Label	Template Label	# Executions	% Completed	Mean Completion Time in Hours
OrigRev	OrigRev	2	100	0.001760731
	OrigRev 1	1	100	0.002000755
	OrigRev 2	2	50	0.000777738

Abbildung 5.2: Ursprüngliche Revisionen-Sheet eines Excel-Exports

Die daraus entstandene Excel-Datei ist immernoch nur im Hauptspeicher vorhanden und muss auch nicht auf die Festplatte geschrieben werden. Das Objekt der Datei kann auf ein „io.Writer“-Interface, dass in der Standardbibliothek von Go enthalten ist, schreiben. Dadurch kann das Objekt direkt in die Antwort der REST-Schnittstelle die Datei schreiben.

### 5.1.6 Tests

Wie in allen Software-Projekten ist es wichtig, den Paper-Tracker zu testen. Dabei spielt vor allem der Backend-Server eine große Rolle, da dieser die meiste Geschäftslogik implementiert.

Alle für den Server existierenden Tests sind automatisierte Tests, die einzelne Einheiten des Codes testen. Vor allem werden die Manager getestet, da diese letztendlich die Geschäftslogik beinhalten. Dafür werden verschiedene Bibliotheken verwendet. Die „ginkgo“-Bibliothek stellt Funktionen zur Verfügung, um die Tests zu definieren und Code vor/nach den Tests auszuführen. Mit Hilfe von „gomega“ können Ergebnisse der Tests

mit zu erwartenden Ergebnissen verglichen werden. Sie stellt neben den einfachen Vergleichen auch Vergleiche für Fehlertypen, komplexe verschachtelte Datenstrukturen, Listen und mehr.

Damit die einzelnen Einheiten des Servers getrennt voneinander getestet werden können, wird zusätzlich die „gomock“ Bibliothek verwendet. Diese ermöglicht es, Stellvertreterobjekte für in diesem Fall Manager und Repositories zu erstellen. Die Stellvertreterobjekte können dann für jeden einzelnen Testfall mit zu erwarteten Parametern und einem Rückgabewert konfiguriert werden. Da „gomock“ nur aus Interfaces Stellvertreterobjekte erstellen kann, müssen für alle Manager und Repositories eigene Interfaces erstellt werden. Deshalb gibt es, im Gegensatz zum Entwurf, für jeden Manager ein Interface und eine Implementierungsklasse. Für die Tests wird dann die Instanz des Singletons auf ein Stellvertreterobjekt gesetzt.

Mit diesen Techniken wurden einige Tests für die Manager-, Modell- und Utilities-Klassen geschrieben. Es sind somit ca. 20% der Manager, ca. 27% der Modelle und ca. 68% der Utilities getestet.

MEHR!

## 5.2 Hardware und Firmware

Im Folgenden wird erläutert, wie der Tracker hinsichtlich der verwendeten Hardware und der entsprechenden Firmware implementiert wurde.

### 5.2.1 Verwendete Technologien

Als Mikrokontrollerplattform wurde die sogenannte „TinyPICO“-Plattform gewählt. Nach eigener Aussage handelt es sich dabei um die kleinste vorgefertigte Entwicklungsplattform basierend auf Mikrocontrollern vom Typ ESP32. Dieser Aspekt ist für die Erfüllung von **NF-2** wichtig. Dieser Mikrocontroller wurde gewählt, da er alle benötigten Funktionen wie beispielsweise WLAN bereits über ein API bereitstellt.

Die TinyPICO-Entwicklungsplattform stellt neben dem Mikrocontroller weitere Hardware bereit, so sind unter anderem die Stromversorgung über Universal Serial Bus (USB) sowie Lithium-Polymer-Akkumulator (Akku), ein Übersetzer von USB zu Universal Asynchron-

nous Receiver Transmitter (UART) zur Programmierung des Mikrocontroller und eine Vollspektrum-Light Emitting Diode (LED) verbaut. Ein weiterer Vorteil ist das bereits integrierte Ladesystem, welches einen angeschlossen Lithium-Polymer-Akku automatisch auflädt, sobald der Tracker über USB mit einer Stromquelle verbunden wird. Außerdem können über das Ladesystem der aktuelle Ladezustand und die Spannung des Akku abgefragt werden, was für die Erfüllung von **F-6** unabdingbar ist.

Die Firmware wurde in der Programmiersprache C++ auf Basis des Arduino-Frameworks erstellt. Dieses Framework wurde gewählt, da es auf vielen Plattformen, wie auch dem ESP32 lauffähig ist, weit verbreitet ist und somit viele Bibliotheken für das Framework existieren.

Das „Flasher“-Tool zum Programmieren und Konfigurieren der Firmware wurde in Python entwickelt. Der Grund dafür liegt in Möglichkeit sehr schnell und simpel ein Programm und auch ein GUI zu programmieren. Dies wird durch die Bibliothek „PySimpleGUI“ ermöglicht, die lediglich mit einer groben Layout-Beschreibung ein Oberfläche zusammenstellt. Alle weiteren benötigten Bibliotheken sind in der Python-Standardbibliothek enthalten. Vor allem ist hierbei die „multiprocessing“-Bibliothek wichtig, mit welcher andere Programme aufgerufen werden können. Um aus dem entstandenen Python-Skript eine ausführbare Datei für die verbreitesten Betriebssysteme zu erstellen, wird „PyInstaller“ verwendet. Die daraus entstehende Datei beinhaltet bis auf das PlatformIO-Tooling alle benötigten Ressourcen.

Kann man  
das mit  
bündeln?!

### 5.2.2 Flasher

Aufgrund der angestrebten Einfachheit dieses Programmes, besteht es aus insgesamt nur drei Dateien. Der Hauptbestandteil des Flashers ist das eigentliche Python-Skript. Zusätzlich wird die Datei „requirements.txt“ benötigt, welche alle Bibliotheken für den Bibliotheksmanager „pip“ auflistet. In dieser Datei sind lediglich, wie oben erwähnt, „pysimplegui“ und „pyinstaller“ aufgeführt. Die dritte und letzte Datei ist ein „Makefile“, welches einige Befehle zum Installieren der Bibliotheken, Ausführen des Skripts oder Erstellen der ausführbaren Datei bereitstellt.

Das Skript selbst beginnt nach dem Importieren der benötigten Bibliotheken mit dem Parsen der Kommandozeilenargumente. Möglich ist nur das Argument **-keep-credentials**.

Das Skript erzeugt eine Datei, in welcher die Zugangsdaten für das vom Tracker zu verwendende WLAN-Netzwerk, sowie weitere potentiell sensitiven Informationen enthalten sind. Im Standardfall wird diese Datei nach dem Erstellen der Firmware und der Programmierung des Trackers gelöscht. Ist das Kommandozeilenargument `-keep-credentials` gegeben, wird die Datei nach Erfolgreichem Abschluss des Programmiervorganges nicht gelöscht, was vor allem in der Entwicklungsphase der Software vorteilhaft ist.

Weiter werden einige Konstanten definiert. Diese sind zum einen der Titel des Programm-Fensters und einige sogenannte „Keys“, die eindeutig bestimmte Elemente der GUI identifizieren.

Nach den Konstanten werden drei Funktionen definiert, die die eigentliche Logik des Flashers beinhalten. Zuerst eine Funktion, die alle möglichen Ports, über die ein Tracker programmiert werden kann, auflistet. Dies wird über ein Aufrufen von PlatformIO ermöglicht: „`pio device list`“.

Die nächste Funktion generiert die für die Firmware benötigte Datei mit der Konfiguration des Trackers. Sie liest ein Template mit Platzhalter-Werten der Datei ein und ersetzt die Platzhalter mit den eingegebenen Werten, die als Parameter übergeben werden. Nach dem Einsetzen der Werte, wird die Datei geschrieben. Der Pfad, an dem sich die Firmware befindet, ist auch Teil der Werte, die als Parameter übergeben werden. Zugriff auf einen einzelnen Wert wird über die oben erwähnten konstanten „Keys“ ermöglicht.

In der letzten Funktion wird das Programmieren an sich ausgeführt. Auch diese Funktion bekommt die eingegebene Werte als Parameter. Aus diesen wird der ausgewählte Port benötigt. Mit diesem wird wieder PlatformIO aufgerufen: „`pio run -e tinypico -t upload -upload-port port`“.

Vor dem Hauptteil des Skripts werden noch die Layouts für die GUI benötigt. Es werden zwei verschiedene Layouts definiert: Ein Eingabe-Layout und ein Programmier-Layout. Das Eingabe-Layout ist in Abbildung 5.3 dargestellt. Es ist eine Tabelle aus Eingabe-Feldern mit Erklärungen, in denen alle benötigten Konfigurationen eingetragen werden können. Die Eingabe-Felder selbst sind mit den Keys verknüpft. Am unteren Teil des Layouts befindet sich ein Button, der das Programmieren startet.

Das Programmier-Layout besteht lediglich aus einem Titel, einem Textfeld und einem Button. Es ist in Abbildung 5.4 dargestellt. Im Textfeld soll der aktuelle Status angezeigt

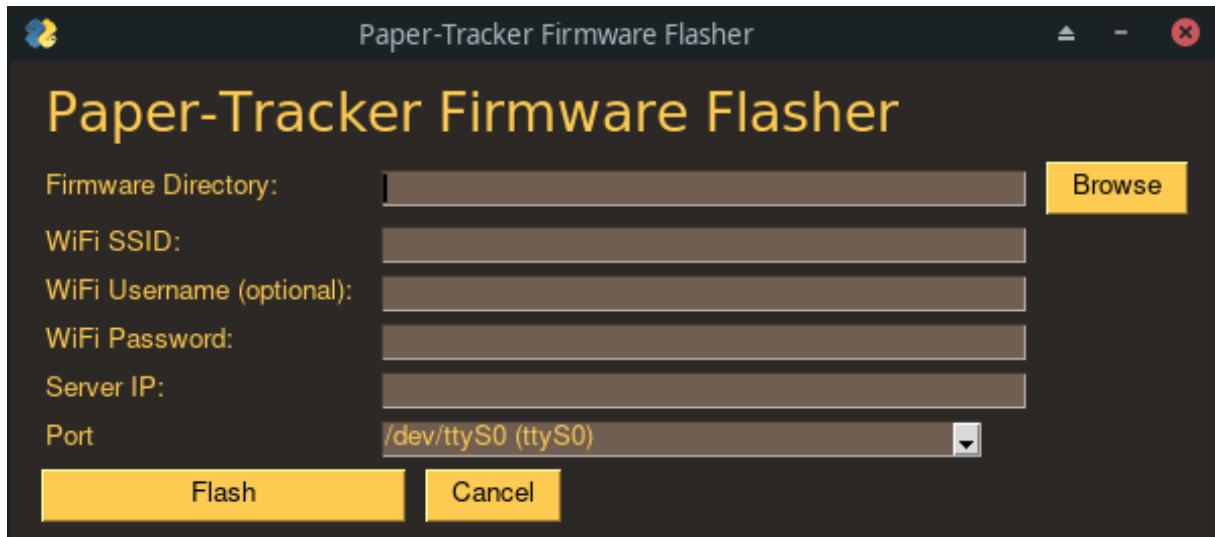


Abbildung 5.3: Layout des Eingabe-Fenster

werden. Der Button ist zum Schließen des Programms, sobald der Vorgang abgeschlossen ist.

Im Hauptteil des Skripts werden nun zuerst das Fenster mit dem Eingabe-Layout geöffnet. Sobald der Button zum Start gedrückt wurde, werden die eingegebenen Werte ausgelesen. Das Eingabe-Fenster wird geschlossen und das Programmier-Fenster mit den entsprechendem Layout geöffnet. Ist dieses offen, wird die Funktion zum Generieren der Konfigurationsdatei und die Funktion zum Programmieren des Trackers aufgerufen. Im Textfeld des Layouts werden diese Schritte entsprechend dokumentiert. Auch die Ausgabe des Programmierens wird in das Textfeld ausgegeben. Nach dem Programmieren wird, je nach Kommandozeilenargument, die generierte Datei gelöscht. Nun kann das Fenster und damit das gesamte Programm über den Schließen-Button geschlossen werden.

Tritt während diesem gesamten Vorgang ein Fehler auf, wird dieser entsprechend aufgefangen und die Fehlermeldung im Textfeld angezeigt. Das Löschen der generierten Datei wird auf im Fehlerfall ausgeführt.

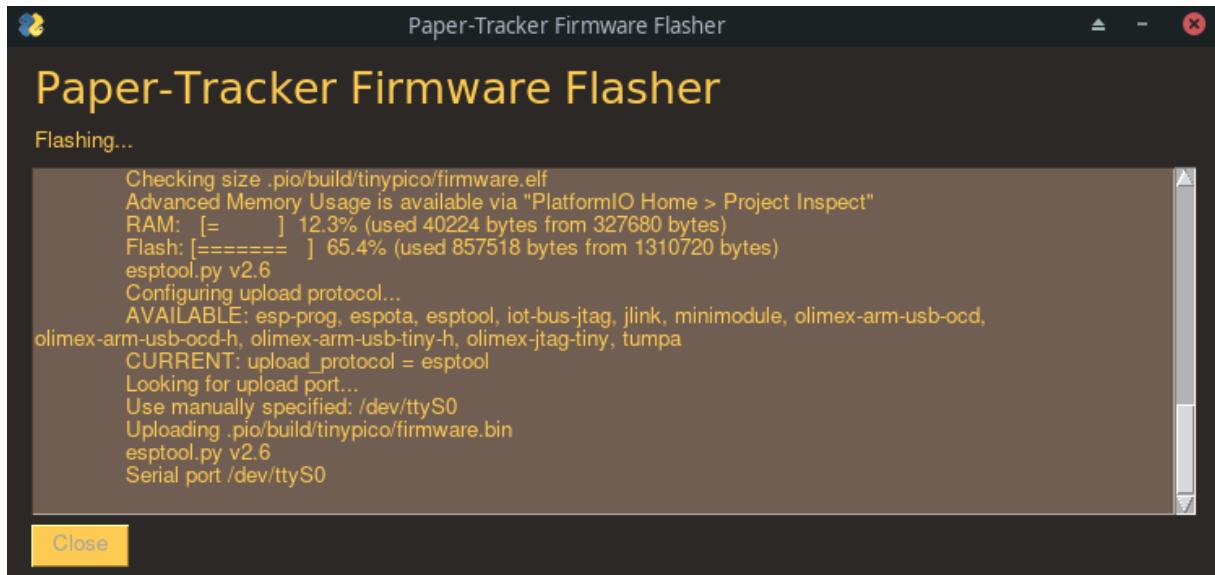


Abbildung 5.4: Layout des Programmier-Fenster

## 5.3 App

Im diesem Kapitel wird die technische Umsetzung der App für Mobilgeräte beschrieben.

### 5.3.1 Verwendete Technologien

Zur Programmierung der App wurden die Programmiersprache Dart und das Framework Flutter verwendet. Ein großer Vorteil dieser Kombination ist, dass Anwendungen sowohl für Android, als auch für iOS kompiliert werden können, ohne den Quellcode anpassen zu müssen. Auch eine Laufzeitumgebung für Desktopcomputer unter Windows, macOS und GNU/Linux befindet sich aktuell im Teststadium, sodass die Paper-Tracker App gegebenenfalls auch auf Arbeitsplatzrechnern verwendet werden kann.

Auch für das Erstellen der App wurden einige Bibliotheken verwendet, die die Entwicklung vereinfachen. Vor allem werden zum Beispiel Bibliotheken verwendet, um die Kommunikation mit dem Backend-Server zu ermöglichen. Die verwendeten Bibliotheken sind im Folgenden aufgelistet:

#### **http (<https://pub.dev/packages/http>)**

„http“ ist eine Bibliothek, die es ermöglicht asynchron HTTP-Anfragen durchzu-

führen. Die Bibliothek wird dafür verwendet, um mit dem Backend-Server zu kommunizieren.

#### **json\_annotation ([https://pub.dev/packages/json\\_annotation](https://pub.dev/packages/json_annotation))**

Mit „json\_annotation“ kann über Annotationen im Programmcode eine Serialisierung zu JSON und Deserialisierung von JSON für Datenklassen generiert werden.

#### **shared\_preferences ([https://pub.dev/packages/shared\\_preferences](https://pub.dev/packages/shared_preferences))**

Durch die „shared\_preferences“ Bibliothek können einfache Konfigurationen als Schlüssel-Wert-Paare abgespeichert werden. Dies wird für die Uniform Resource Locator (URL) des Backend-Servers verwendet.

#### **material\_design\_icons ([https://pub.dev/packages/material\\_design\\_icons\\_flutter](https://pub.dev/packages/material_design_icons_flutter))**

„material\_design\_icons“ stellt zusätzliche Material<sup>2</sup>-Icons zur Verfügung. Dies ist notwendig, da die standardmäßig verfügbaren Icons in Flutter nicht ausreichend sind.

#### **fluttertoast (<https://pub.dev/packages/fluttertoast>)**

Mit „fluttertoast“ können kleine Pop-ups dargestellt werden, die dem Nutzer der App einfach und schnell Feedback zu einer Aktion geben können.

#### **intl (<https://pub.dev/packages/intl>)**

Die „intl“-Bibliothek wird für das Formatieren von Daten und Uhrzeiten verwendet. Zudem kann sie für die Übersetzung der App in andere Sprachen verwendet werden.

## **5.4 Hardware-Gehäuse**

Neben der Hardware des spezifischen Trackers und der Software wurde auch ein Hardware-Gehäuse entwickelt. Dieses Gehäuse soll den Tracker mit der Batterie beinhalten und eine Möglichkeit bieten, dieses an einem Stapel Papier zu befestigen. Die Herstellung des Gehäuses soll in einem 3D-Drucker erfolgen. Dazu muss das Gehäuse digital modelliert werden.

Bevor die Modelle selbst erstellt werden können, muss ein grobes Layout vorliegen. Für den Paper-Tracker wurden zwei verschiedene Layouts in Betracht gezogen. Diese werden in

---

<sup>2</sup>Eine Designssprache von Google, die in fast allen Google-Produkten eingesetzt wird und auf Minimalismus setzt. [11]

den folgenden Sektionen genauer erläutert und die aus den Layouts entstandenen Modelle gezeigt.

### 5.4.1 Layout Controller auf der Batterie

Das erste Layout ist in Abbildung 5.5 abgebildet. Bei diesem Layout befindet sich der Mikrocontroller auf der Batterie.

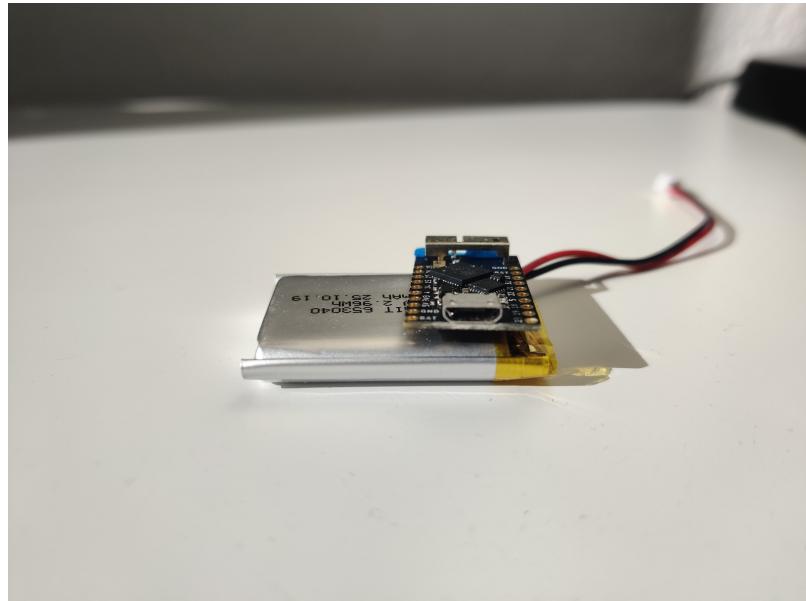


Abbildung 5.5: Gehäuse-Layout mit Mikrocontroller auf der Batterie

Dies ergibt den Vorteil, dass die Grundfläche des Trackers mit ca. 45x35mm sehr gering gehalten wird. Daraus ergibt sich dementsprechend der Nachteil, dass mit ca. 15mm die Konstruktion relativ hoch wird.

Das aus dem Layout entstandene Modell besteht aus zwei verschiedenen Teilen: Dem Körper des Gehäuse und dem Deckel.

Der Körper ist in Abbildung 5.6 dargestellt. Er biete eine Art Schublade, in welche die Batterie hineinpasst. Oberhalb der Schublade wird der Mikrocontroller durch Wände im Platz gehalten. An der Seite gibt es eine Aussparung, um den USB-Port zum Laden benutzen zu können.

Geschlossen wird der Körper mit dem Deckel. Dies ist in Abbildung 5.7 dargestellt. Die

beiden Teile sollen nur über das Zusammenstecken miteinander verbunden werden. Am Deckel ist auch der Mechanismus zum Anklemmen des Gehäuses an Papier angebracht. Er besteht aus einem Steg, der schräg zum Gehäusekörper zeigt. Zudem sind Noppen angebracht, die die Grifffestigkeit erhöhen sollen.

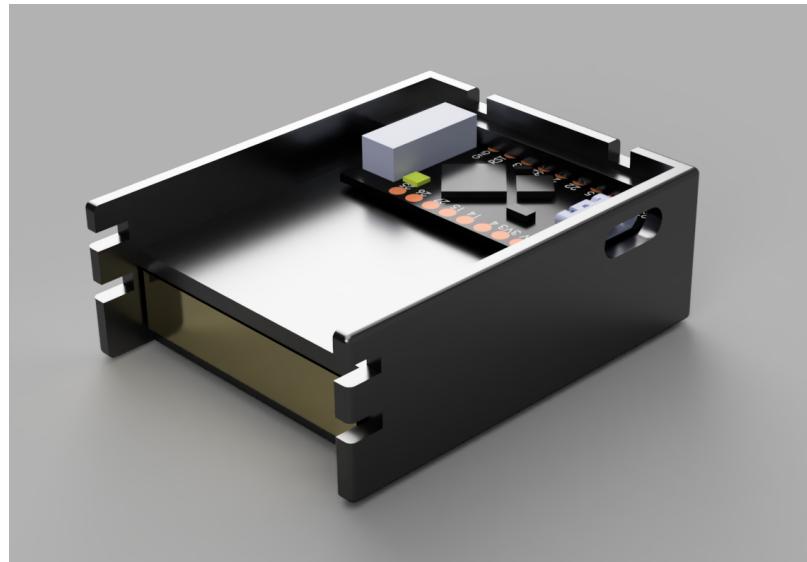


Abbildung 5.6: Körper des Gehäuse-Modell mit Controller auf der Batterie

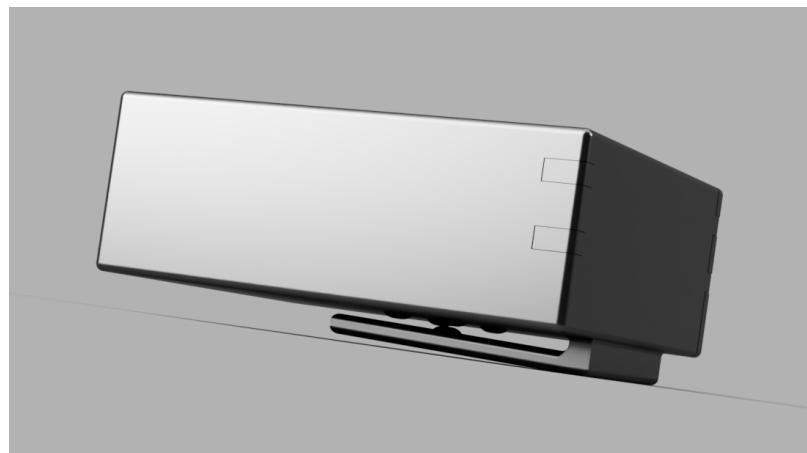


Abbildung 5.7: Gehäuse-Modell mit Controller auf der Batterie

### 5.4.2 Layout Controller neben der Batterie

Das zweite Layout ist in Abbildung 5.8 abgebildet. Der Unterschied zum ersten Layout ist, dass der Mikrocontroller neben der Batterie platziert wird.

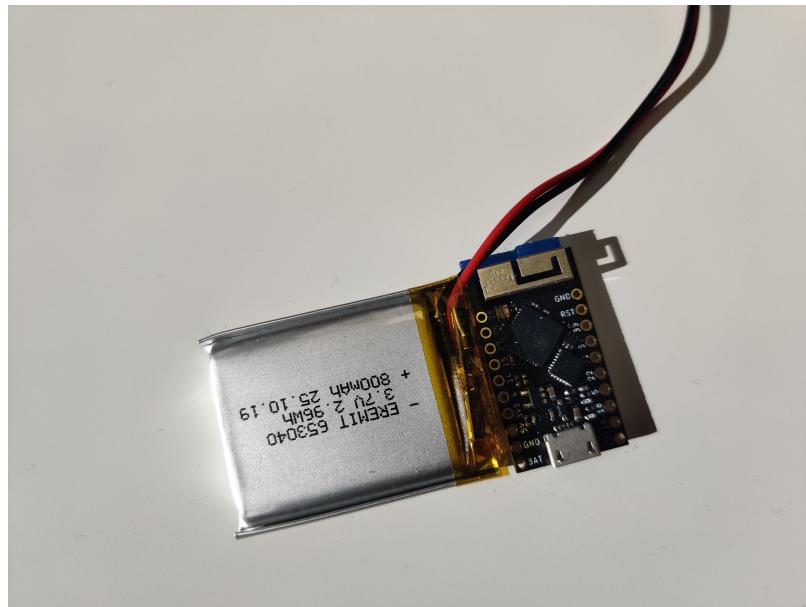


Abbildung 5.8: Gehäuse-Layout mit Mikrocontroller neben der Batterie

Dies resultiert in einer erhöhten Grundfläche zum ersten Layout von 70x35mm aber in einer Reduktion der Höhe auf nur 12mm.

Das Modell nach diesem Layout ist in drei verschiedene Teile aufgeteilt. Es gibt, wie zum ersten Layout, auch einen Körper und einen Deckel. Zusätzlich gibt es jedoch noch den Clip zum Befestigen an Papier. Dieser ist demnach nicht mehr im Deckel mit enthalten.

Der Körper ist in Abbildung 5.9 dargestellt. Er bietet einen Platz für die Batterie und den Mikrocontroller nebeneinander. Für den USB-Port ist ebenfalls in der Außenwand eine Ausparung vorhanden. Kleine Erhebungen halten die Teile am Platz und verhindern ein umherschieben. In diesen Erhebungen sind auch Löcher für Schrauben inkludiert und auf der Unterseite Platz für eine Mutter. Der größere Einschnitt in der Unterseite ist für den Clip bestimmt.

Der Deckel des Modells ist ähnlich dem Körper, besitzt aber keine Außenwände. Er ist in Abbildung 5.10 abgebildet. Die kleinen Erhebungen des Deckels fixieren die Teile innerhalb

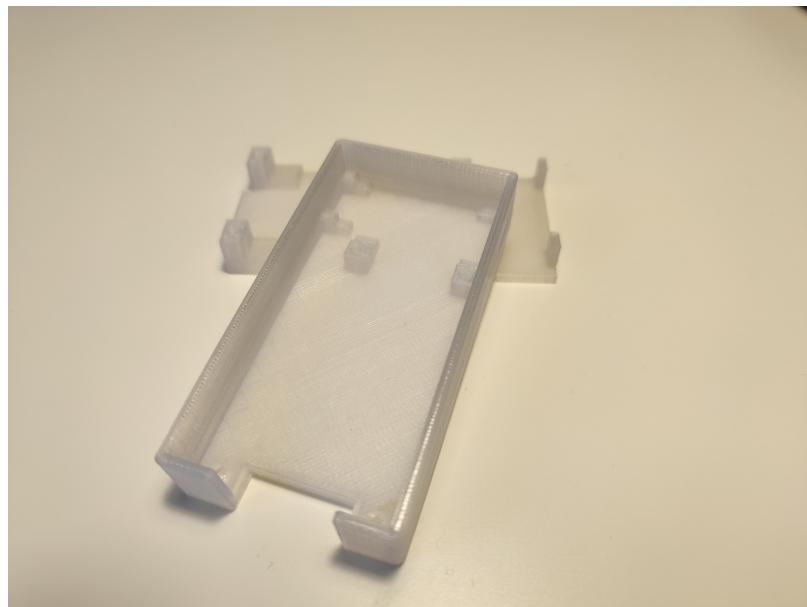


Abbildung 5.9: Körper des Gehäuse-Modell mit Controller neben der Batterie

des Gehäuses auf der vertikalen Achse. Statt Platz für eine Mutter auf der Aussenseite der Erhebungen mit Schraubenlöchern, ist eine Einsenkung für den Schraubkopf vorgesehen.

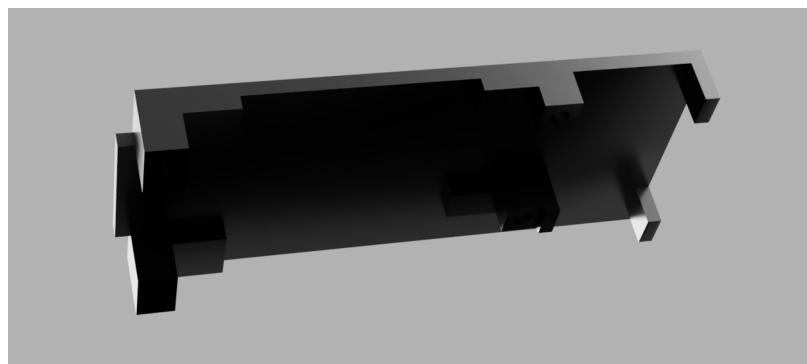


Abbildung 5.10: Deckel des Gehäuse-Modell mit Controller neben der Batterie

Angebracht ist der Clip an dem Einschnitt im Körper und fixiert durch die zwei danebenliegenden Schraubenlöcher. Die Schraube wird dabei zwischen den Erhebungen des Körpers und des Deckels, sowie durch die in Abbildung 5.11 dargestellten Flügel des Clips geführt. Für die Fixierung an einem Papierstapel wurde die gleiche Technik, wie für das erste Layout verwendet. Um den 3D-Druck zu simplifizieren wurden bei diesem Entwurf jedoch keine Noppen an den Clip und an den Körper des Gehäuses angebracht.

Diese werden nach dem Druck aufgeklebt.

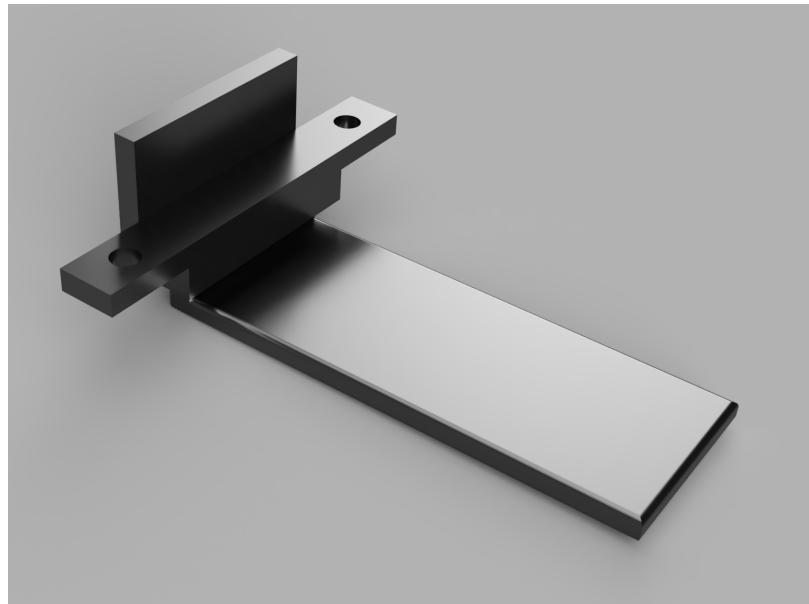


Abbildung 5.11: Clip des Gehäuse-Modell mit Controller neben der Batterie

## **5.5 Bestehende Herausforderungen**

### **5.5.1 Ungenauigkeit des Tracking**

### **5.5.2 UI-Details in der App**

# **6 Validierung**

## **6.1 Genauigkeitsmessung des Tracking**

### **6.1.1 Beschreibung**

### **6.1.2 Durchführung**

### **6.1.3 Ergebnisse**

## **6.2 Nutzerumfrage zur UX der App**

### **6.2.1 Beschreibung**

### **6.2.2 Durchführung**

### **6.2.3 Ergebnisse**

### **6.2.4 Gesamtergebniss der Validierung**

# **7 Ausblick und Weiterentwicklungen**

## **7.1 Audiovisuelle Signale**

## **7.2 Verbessertes Gehäuse**

## **7.3 Tracking von Smartphones**

Es ist wünschenswert, dass eine Person, die einen Raum betritt, in welchem ein Dokument auf deren Bearbeitung wartet, darüber informiert wird. Zu diesem Zweck können Smartphones in das Tracking-System mit aufgenommen werden.

## **8 Zusammenfassung**

# Glossar

**GPS** Das Global Positioning System (GPS) ist ein von den Vereinigten Staaten von Amerika entwickeltes globales Navigationssystem, welches im zivilen, kommerziellen, wissenschaftlichen und militärischen Bereich Anwendung findet, um die Position von GPS-Empfängern zu bestimmen. (vgl. [16]). 10

**Overhead** Also Overhead werden Metadaten bezeichnet, welche zusätzlich zu den eigentlichen Nutzdaten anfallen. Beispiele für Overhead sind Prüfsummen, Netzwerkadressen, oder Codierungszeichen . 15

Quelle für  
Overhead  
suchen

# Literatur

- [1] Schnabel, P. *IEEE 802.11 / WLAN-Grundlagen*. 03/2020. URL: <https://www.elektronik-kompendium.de/sites/net/0610051.htm> (besucht am 27.03.2020) (siehe S. 2).
- [2] Luber, S. *Was ist eine MAC-Adresse?* 08/2018. URL: <https://www.ip-insider.de/was-ist-eine-mac-adresse-a-665074/> (besucht am 27.03.2020) (siehe S. 2).
- [3] Haider, Z. *802.11 Topologies AKA Service Sets*. 03/2019. URL: <https://www.wifi-professionals.com/2019/03/802-11-topologies-aka-service-sets> (besucht am 27.03.2020) (siehe S. 3).
- [4] „IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications“. In: *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)* (03/2012) (siehe S. 3).
- [5] Patwari, N. u. a. „Relative location estimation in wireless sensor networks“. In: *IEEE Transactions on Signal Processing* 51.8 (08/2003), S. 2137–2148 (siehe S. 10).
- [6] Dizdarević, J. u. a. „A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration“. In: *ACM Computing Surveys* 51.6 (01/2019), S. 1–29 (siehe S. 15).
- [7] Naik, N. „Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP“. In: *2017 IEEE International Systems Engineering Symposium (ISSE)*. IEEE, 10/2017 (siehe S. 15).
- [8] Fielding, R. T. „Architectural Styles and the Design of Network-based Software Architectures“. Diss. University of California, 2000 (siehe S. 16).
- [9] Shafranovich, Y. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. RFC 4180. RFC Editor, 10/2005. URL: <https://tools.ietf.org/html/rfc4180> (siehe S. 21).

- [10] *Information technology — Document description and processing languages — Office Open XML File Formats — Part 1: Fundamentals and Markup Language Reference.* Standard. International Organization for Standardization, 11/2016 (siehe S. 21).
- [11] LLC, G. *Introduction - Material Design.* URL: <https://material.io/design/introduction/#> (besucht am 06.04.2020) (siehe S. 22, 44).
- [12] Weigend, J. *Microservices mit Go / Informatik Aktuell.* 06/2019. URL: <https://www.informatik-aktuell.de/entwicklung/methoden/microservices-mit-go.html> (besucht am 01.04.2020) (siehe S. 34).
- [13] Jinzhu. *Connecting to database / GORM.* 03/2020. URL: [https://gorm.io/docs/connecting\\_to\\_the\\_database.html](https://gorm.io/docs/connecting_to_the_database.html) (besucht am 01.04.2020) (siehe S. 36).
- [14] AG, A. *WiFiUDP.* URL: <https://www.arduino.cc/en/Reference/WiFiUDPConstructor> (besucht am 15.04.2020) (siehe S. 36).
- [15] Inc., G. *tealeg/xlsx: Go (golang) library for reading and writing XLSX files.* URL: <https://github.com/tealeg/xlsx> (besucht am 20.04.2020) (siehe S. 37).
- [16] Department Of Transportation, F. A. A. *Global Positioning System Wide Area Augmentation System (WAAS) Performance Standard.* 10/2008. URL: <https://web.archive.org/web/20170427033332/http://www.gps.gov/technical/ps/2008-WAAS-performance-standard.pdf> (besucht am 30.10.2019) (siehe S. 53).

# **Todo list**

■ . . . . .	2
■ Primärquelle finden . . . . .	2
■ Primärquelle finden (IEEE RFC) . . . . .	2
■ Primärquelle finden . . . . .	3
■ Evtl. noch eine Anforderungsanalyse mit anderen Leuten durchführen . . . . .	6
■ Es wäre schön, Glossar-Referenzen zu kennzeichnen (oft mit Glos, oder Gls) . . .	10
■ Quelle für die Genauigkeit von GPS in Gebäuden . . . . .	10
■ Ist das hier eher Analyse? . . . . .	10
■ Glossar: Mesh-Netzwerk . . . . .	10
■ Time-of-Arrival erläutern . . . . .	10
■ Papers zu RSSI-Basiertem Tracking einfügen (Microsoft) . . . . .	10
■ Lifecycle des Paper-Trackers erläutern . . . . .	11
■ Quellen verwenden, die in diesem Paper zitiert werden . . . . .	15
■ Hier das Polling-Prinzip oder unter Hardware?! . . . . .	15
■ Quelle für aktuelle Protokolle . . . . .	16
■ Soll REST erläutert werden? . . . . .	16
■ Zwischen "überschriften" hinzufügen für Room, Tracker, etc. . . . .	16
■ Erstellung der DB Verbindung reinpacken . . . . .	19
■ Spezielle Fkt einzelner Manager erklären? . . . . .	19
■ Ist schon irgendwo erläutert, wann ein Workflow nicht mehr bearbeitbar ist? . .	24

■ Studie über Nutzen von Icons zur Wiedererkennung von Schlüsselwörtern ein-bringen . . . . .	31
■ Das hier kann vielleicht zu CoAP-Schnittstelle verschoben werden . . . . .	36
■ MEHR! . . . . .	39
■ Kann man das mit bundeln?! . . . . .	40
■ Quelle für Overhead suchen . . . . .	53