hochschule mannheim

Fakultät Informationstechnik
Studiengang Technische Informatik

Bachelorarbeit

Smartphones als Sensorbox

Marius Cerwenetz

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

Vorgelegt von	Marius Cerwenetz
am	08. Juli 2022
Referent	Prof. Dr. Peter Barth
Korreferent	Prof. Dr. Jens-Matthias Bohli

Schriftliche Versicherung laut Studien- und Prüfungsordnung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mannheim, 08. Juli 2022

Marius Cerwenetz

Zusammenfassung

In dieser Arbeit wurde ein Framework erstellt um Smartphonesensoren über eine Programmierumgebung auszulesen und Ausgaben auf dem Smartphone auszuführen. Hierfür wurde eine Android-Anwendung, eine Kontrollanwendung und drei Softwarebibliotheken in dem Sprachen C, Java und Python implementiert. Als Verbindungstechnologien kommen UDP und MQTT zum Einsatz.

Inhaltsverzeichnis

Kapitel 1

Einführung

Ziel dieser Arbeit ist es ein Framework zu entwickeln, dass Smartphones von Anwendern in eine Programmierumgebung einbindet um Sie beim Programmierenlernen zu unterstützen. Hierbei stehen insbesondere die Interaktivität in Form von Sensordatenübermittlung und Ausgaben im Vordergrund.
Viele frische Entwicklerinneren und Entwickler oder Personen die das Programmieren gerade erst entdecken mühen sich zu Anfang mit grundlegenden softwaretechnischen Konzepten, syntaktischen Herausforderungen oder finden sich wieder in semantischen Stromschnellen. Rein virtuelle, akademische Übungsaufgaben senken meist die Lernmotivation, abstrahieren gelerntes und lassen Softwareentwicklung fade und dumpf erscheinen. Projekte mit Microcontrollern bieten eine praktikablere, einfache, realitätsbezogene Einstiegsmöglichkeit. Spielerisch können kleine Projekte realisiert werden die durch physische Äußerungen Entwickler einladen sich an Problemen auszuporbieren. Diese Eigenschaften sind insbesondere bei Projekten mit wenig Vorwissen für Lernwillige wie Schüler oder Erstemester-Studierende sinnvoll. Gelerntes kann direkt angewand werden und Änderungen an Algorithmen sind schnell in der Verhaltensweise der Hardware erkennbar. Praktische Programmieraufgaben bieten für Programmieranfänger den höchsten Lerneffekt bei höchster Motivation.[2]
Die in Microcontroller integrierte Sensoren bilden hierbei den Schlüsselstein zwischen Änderungen in der realen Welt und Auswirkungen im virtuellen Programm. Für Nutzer erscheint ein Gerät bedienbar. Änderungen der realen Umgebung wirken sich unmittelbar auf den geschriebenen Algorithmus aus. Bei einem Fehlverhalten kann dieser angepasst werden, bis das gewünschte Verhalten vorliegt. Die ständige Berührung und Formung des Algorithmus mindert Ängste vor Änderungen, schafft Routine und damit ein tieferes Verständnis, Context und Hintergrundwissen für Probleme.

Projekte mit Microcontrollern sind jedoch auch mit Nachteilen und Einstiegsvor-

aussetzungen behaftet. Ein Arduino-Einstiegskit kostet im internen Arduino-Shop im Juni 2022 87,90 € [3]. Ein Großteil der Kosten entfällt zwar auf den eigentlichen Microcontroller, ein nicht unmittelbarer Teil jedoch auch auf Peripherie wie wie Breadboards, Verbindungskabel und Erweiterungsboards. Neben zusätzlich verursachten Kosten setzen Sie außerdem gewisses Hintergrundwissen voraus. Für die erstmalige Verwendung von Breadboards muss beispielsweise bekannt sein welche Ports wie verbunden sind, welche Konventionen es für Plus- und Minuspol gibt und welche Bauteile für die Benutzung überhaupt geeignet sind. Dies stellt eine Einstiegshürde dar, die die eigentliche interaktive Lernerfahrung herauszögert und Ziele entfernt. Einstiegshürden wie dies führen zu einem Motivationsverlust.

Durch Smartphones lässt sich dieses Problem jedoch umgehen. Moderne Fertigungsverfahren der Halbleitertechnik reduzieren die durchschnittliche Chip-Größe enorm. Dies betrifft auch Sensoren. Bauartbedingte Einschränkungen sind heute nicht mehr so wesentlich wie zu Anfang der Smartphone-Entwicklung. Aktuelle Smartphones sind daher bereits mit zahlreichen Sensoren wie Lagesensoren, Gyroskop oder Näherungssensoren ausgestattet. Diese sind zwar für die komfortable Alltagsnutzung des Geräts konzipiert, können jedoch ganz regulär in Anwendungen dediziert angesteuert und ausgelesen werden.
Die elektrische Komponente konventioneller Microcontroller-Sets erlaubt außerdem einen Fehlgebrauch der im schlimmsten Fall in der Zerstörung von Komponenten enden kann und zusätzliche Kosten verursacht. Projekte mit Smartphones reduzieren diese Komplexitätsebene. Sämtliche Schaltkreise sind bereits intern verschaltet und durch die Qualitätssicherung der Smartphone-Hersteller von möglichen Hardwarefehlern weitgehendst geschützt.
Ein weiterer Vorteil Smartphones statt Microcontroller zu verwenden liegt in der Verfügbarkeit. Weltweit besaßen 2022 5,2 Mrd. Menschen ein Smartphone. [5] Vor allem jüngere Menschen nutzen Smartphones. Viele Kinder besitzen bereits mit 10 Jahren [8] ein Smartphone. Da ein Smartphone üblicherweise nicht nur für Programmieraufgaben, sondern auch für den Alltag verwendet wird, sind die Anschaffungskosten hier auch nicht auf einen einzigen Zweck gebunden.

Zusammenfassend lässt sich konstatieren, dass Smartphones eine Alternative zu herkömmlichen Microcontroller-Sets darstellen. Sie verursachen weniger Kosten und sind meistens schon in Gebrauch. Durch Ihre eingesetzten Sensoren können Sie zuverlässig Umgebungseigenschaften quantisieren. Sie haben eine höhere Prozessorenleistung, so dass auch rechenintensivere Aufgaben ausgelagert werden können. Durch die Entwicklungsmöglichkeiten von Mobilen Anwendungen sind der Kreativität der Ausgabemöglichkeiten keine Grenzen gesetzt. Neben üblichen Übertragungsschnittstellen wie USB bieten Sie häufig auch eine W-LAN und Bluetooth-Funktion die eine

Ansteuerung über unterschiedliche Technologien einfach macht und autonomere, den Umgebungsansprüchen angepasste Nutzungsszenarien ermöglicht.
Eine Einbindung von Smartphones mit Bordmitteln ist in den meisten Prgrammierumgebungen jedoch nicht vorgesehen. Smartphones und Entwicklungsumgebungen bieten meist keine direkte Möglichkeit der Kopplung. Auch gibt es Seitens der Smartphone-Betriebssystemhersteller keine standardisierten Ausgabemöglichkeiten. Moderne Smartphone-Betriebssysteme bauen sich modular aus mobilen Anwendungen auf, so dass das Betriebssystem keine direkten Ausgabemöglichkeiten besitzt. Visuelle und haptische Ausgaben sind nur über mobile Anwendungen möglich.
Das zentrale Thema besteht darin eine Smartphone-Anwendung für visuelle Ausgaben und Sensorwerte zu entwickeln und Schnittstellen in Programmierumgebungen zu schaffen, die die Nutzungsmöglichkeiten des Smartphones in Programmierprozesse einbinden. Möglich wird dies durch Softwarebibliotheken die Funktionsaufrufe bereitstellen, welche Aktionen auf dem Smartphone auslösen. Das Smartphone reagiert auf die empfangen Anfragen und führt die entsprechenden Kommandos aus. Das Framework besteht aus einer programmiersprachenunabhängigen Bibliothek, einer Server-Anwendung und einer mobilen Anwendung für Android Smartphones. Neben den technisch Details und Funktionsweisen werden in dieser Arbeit außerdem Beispielaufgaben gereich mit den angehende Programmiererinnen oder Programmierern gereicht. Diese sind in Kapitel 2 zu finden. In Kapitel 3 werden die verwendeten Komponenten vorgestellt und Designentscheidungen innerhalb des Entwicklungsprozesses erläutert. Die einzelnen Komponenten werden anschließend in Kapitel 4, Kapitel 5 und Kapitel 6 im Detail erklärt. Zum Schluss werden die Ergebnisse in Kapitel 7 vorgestellt und in Kapitel 8 diskutiert.

Kapitel 2

Smartphones als Aktor und Sensor

Smartphones sind in sich geschlossene technische Geräte, die neben vordefinierten Verbindungsschnittstellen wie einem USB-Port, WLAN und Bluetooth meistens keine weiteren Schnittstellen bieten um externe Hardware und Schaltungen anzuschließen und Fernzusteuern. Microcontroller-Schaltungen zum Programmierenlernen bieten meistens mehrere Ausgabemöglichkeiten wie LEDs, Lautsprecher oder Piepser. Smartphones bestechen hier durch den Vorteil der virtuellen Darstellung auf dem Display. Gewohnte Ausgabeelmente können dadurch virtualisiert werden. Die Funktionen sind hier außerdem nicht nur von Mehrzweck-Ausgaben wie LED-Grids begrenzt die zum Beispiel für die Textanzeige oder Bildanzeige verwendet werden können. Spezifische Zweckgebundene Elemente wie Textfelder, Textausgaben oder Bildausgaben in der App können beliebig kombiniert werden. Darüber hinaus ist die Anordnung der jeweiligen Elemente frei, so dass das Layout anders als bei Microcontrollern auch im Nachhinein noch geändert werden kann, was den Entwicklungsprozess deutlich beschleunigt und flexibler macht.
Moderne Smartphones bieten außerdem eine manigfaltige Menge an unterschiedlichen Sensoren. Android bietet für die Verwendung Lage-, Temperatur-, Schwerkraft-, Licht-, Magnetfeld, Luftfeuchtigkeitssensoren und noch einige Weitere [7]. Moderne Flaggschiff-Smartphones bieten viele dieser Sensortypen auch an. Die Vielfalt übersteigt meistens die
Ein Nachteil liegt in der Ansteuerung von mechanischen Bauteilen wie Motoren. Diese werden meistens durch eine elektrische Verbindung zwischen Bauteil und Controller realisiert. Smartphones bieten diese Art der Ansteuerung nicht, da Hardwareschnittstellen wie GPIO-Pins nicht zur Verfügung stehen.
Nichtsdestotrotz bieten Smartpones eine Menge Einsatzmöglichkeiten um einfache Programmierbeispiele auszuprobieren. In diesem Kapitel werden in Abschnitt 2.1 Beispielaufgaben für Programmiererinnen und Programmierer vorgestellt die Sie mit dem Framework lösbar sind. Durchführbar ist dies durch verschiedene Ansteuerun-

gen aus der Programmierumgebung der Entwickler. Die angebotenen Funktionalitäten stellen Anforderungen and das Projekt auf die in Abschnitt 2.2 eingegangen wird.

2.1 Beispielprogrammieraufgaben

Praxisnahe Programmieraufgaben sollen angehende Softwareentwickler an Probleme heranführen. Sie sollen nicht zu kompliziert, jedoch auch nicht zu trivial sein. Unterschiedliche Schwierigkeitsklassen helfen ihnen sich selbst an Aufgaben heranzutasten. Da die Aufgaben außerdem interaktiv gestaltet sein müssen sind Sie immer mit einem Äußeren Einfluss verbunden, auf den reagiert werden muss.
Eine Liste der Aufgaben ist Tabelle 2.1 zu entnehmen. Beschrieben wird der Name

Name der Aufgabe	**Sensoren**	**Ausgaben**	**Schwierigkeit**
Disco	-	Led	+
Würfeln	Lagesensor	Textfeld	+
Diebstahl-Alarm	Näherungssensor	Textfeld, Led, Vibration	++
Klatsch-Zähler	Mikrofon	Textfeld	++
Dreh-Zähler	lagesensor	Textfeld	+++

Tabelle 2.1: Aufgaben

der Aufgabe, die benötigten Sensoren, die benötigten Ausgaben sowie der geschätzte Schwierigkeitsgrad. Letzerer ist in drei größen gegliedert: Einfach (+), Mittel(++) und Schwer (+++).

In der Aufgabe *Disco* soll eine virtuelle LED alle 500 ms die von der Farbe grün auf die Farbe rot wechseln. Die Aufgabe benötigt keine Sensoren, da die LED nicht auf eine Eingabe seitens des Programmierers warten soll. Sie soll immer blinken. Die Ausgabe der farbig blinkenden LED wird über eine virtuelle LED auf dem Display des Smartphones realisiert. Der Schwierigkeitsgrad ist als einfach einzustufen, da hier weder Sensoren benötigt werden, noch sonstige Abhängigkeiten innerhalb des Codes zu berücksichtigen sind.

In der Aufgabe *Würfeln* soll gemessen werden, ob ein Smartphone geschüttelt wurde. Ist dies der Fall soll eine Zufallszahl generiert werden. Anschließend soll diese auf dem Smartphone ausgegeben werden. Verwendet wird dabei der Lagesensor, der Beschleunigungen in drei Achsenrichtungen misst. Als Ausgabe muss für die Zufallszahl das Textfeld verwendet werden. Die Schwierigkeit wird hier auch auf Einfach eingeschätzt, da lediglich ein Sensor und eine Ausgabe verwendet wird. Auch der Algorithmus ist nicht von vielen Abhängigkeiten geprägt.

In der Aufgabe *Diebstahl-Alarm* soll überprüft werden, ob der Näherungssensor eine Annäherung gemessen hat. Wenn dies der Fall ist soll der Text „ALARM“ auf dem Smartphone ausgegeben werden. Das Smartphone soll fünf mal für die Zeit von 1000ms vibrieren. Die Farbe der virtuellen LED soll wie in Aufgabe *Disco* alle 500ms die Farbe wechseln. Dies soll einen Diebstahl-Alarm simulieren bei der sich jemand mit der Hand dem Smartphone nähert.
Hier wird lediglich der Näherungssensor verwendet um eine Näherung zu detektieren. Die Ausgabe des Alarm-Textes soll auf einem Textfeld dargestellt werden. Außerdem sollen die Vibrationsmotoren des Smartphones, die sonst beispielsweise für Benachrichtigungen verwendet werden, benutzt werden um ein haptisches Feedback zu liefern. Der Schwierigkeitsgrad der Aufgabe ist als *Mittel* eingestuft, weil die Aufgabe zu unterschiedlichen Zeit, unterschiedlich lang auf mehreren Ausgabemöglichkeiten erfolgen muss. Außerdem ist die Ausgabe an die Bedingung einer Näherung an das Smartphone gebunden.

Bei der Aufgabenstellung *Klatsch-Zähler* muss für einen definierten Zeitraum die Anzahl der Klatscher gemessen werden. Programmierer starten also ein Programm und klatschen vor dem Smartphone in die Hände. Die Anzahl der Klatscher soll dann gemessen und auf dem Display des Smartphones ausgegeben werden. Verwendet wird dafür lediglich das Mikrofon und das Textfeld des Smartphones. die Schwierigkeit ist hier ebenfalls auf *Mittel* gesetzt, da die Klatscher auf den Sensor nicht immer die gleiche intensivität haben. Programmierer müssen also erproben, wann es sich wirklich um ein Händeklatschen handelt und wann einfach nur Hintergrundgeräusche zu hören sind.

Bei der Letzen Aufgabe *Drehzähler* soll der Programmierer das Smartphone innerhalb eines eigens definierten Zeitraums drehen und anhand der zurückgegeben Werte die Anzahl der Drehungen ermitteln. Diese soll anschließend im Textfeld auf dem Smartphone ausgegeben werden. Zur Verwenung kommen hier ebenfalls der Lagesensor als Sensoreingabe und das Textfeld als Ausgabe. Die Aufgabe ist als *Schwer* bewertet, da hier Wiederholungen in einer Werteabfolge erkannt werden müssen. Diese sind jedoch nicht immer exakt gleich. So muss auch erprobt werden wie sich eine Umdrehung überhaupt in den Werten äußert und wie Drehungen optimal erkannt werden können.

Die Aufgaben können natürlich nach Belieben kombiniert werden. *Disco* ist beispielsweise schon in *Diebstahl-Alarm* integriert. Denkbar wäre auch beim *Würfeln* den *Klatschzähler* für die generierung der Zufallszahl zu verwenden. Der Diebstahlalarm kann auch bei einer ungeraden Anzahl an Drehungen des Drehzählers gestartet

werden.
Dadurch werden Programmierern Konzepte wie Schleifen oder Bedingungen nah gebracht. Zum Beispiel könnte die LED blinken wenn eine gewisse Umdrehungsanzahl erreicht wurde um das Konzept einer While-Schleifen zu erläutern.

2.2 Softwareanforderungen

Die im vorherigen Kapitel erläuterten Beispielprogrammieraufgaben stellen Anforderungen an das Framework, die die weiteren Design- und Architekturentscheidung in der Konzeption bestimmen. Ziel ist es die Programmierumgebung möglichst nahtlos an das Smartphone heranzubringen. Nutzern soll nach Möglichkeit nicht gewahr sein, dass die Entscheidungsstrukturen von Algorithmen auf dem Smartphone in der lokalen Programmierumgebung ablaufen. Die zugrundeliegende Komplexität in der Benutzung soll reduziert werden um Eintrittsbarrieren zu beschränken. Funktionsaufrufe sollen möglichst einfach und verständlich gehalten werden. Asnychrone Konzepte wie Futures oder Tasks werden nicht umgesetzt. Dise stellten weitere Voraussetzungen an Programmierer und brächten eine zusätzliche Komplexitätsebene der Ablaufgestaltung hinzu. Rückgabewerte müssen simpel gestaltet sein. Eine Ausgabe in serialisierten Formaten wie XML oder JSON erfordert zusätzliches Hintergrundwissen und weitere Bibliotheken sowie zusätzlichen Code der die Umsetzung von Aufgaben eher behindert. Dies fördert zudem ein rebungsloses Zusammenspiel von Sensordaten-Abfragen und Kommandos die auf dem Smartphone ausgeführt werden. Da die Ausgaben auf dem Smartphone jedoch häufig von einer Änderung der Umgebungsbedingungen abhängen sind die Anforderungen an Sensordatenabfragende Methoden wichtig.

Sensordaten müssen sofort angefragt werden können. Sie müssen zum Zeitpunkt der Abfrage bereits vorliegen. Sensor-Mess-Prozesse dürfen nicht auf eine Anfrage hin gestartet, sondern vorher begonnen werden. Je rascher die Werte vorliegen desto schneller kann im Programm drauf reagiert werden. Für Anwender äußert sich das in einem reaktionsfähigerem Verhalten der programmierten Anwendung. So können Experimente eher begriffen werden, da Auswirkungen in der Realität instantan Auswirkungen auf den vom Schüler oder Studenten entwickelten Programmablauf haben.
Neben einer allgemeinen Vereinbarung wann Messprozesse gestartet werden muss natürlich auch berücksichtigt werden, dass die Sensorwerte einen Transportweg durchlaufen. Berücksichtigt werden müssen also auch Latenzen zwischen Smartphone und Programmierumgebung. Diese können je nach Mobilfunk- bzw WLAN Standard, Umwelteinflüssen und Geräteanzahl variieren. Auch Round-Trip-Time

(RTT) spielt eine Rolle. Während die Sie bei Standards wie WLAN 802.11b ca. 10 ms beträgt kann Sie bei UMTS auf 300 ms bis 400 ms ansteigen.[1] Um diesen Effekt zu mitigieren müssen Sensorwerte zwischengespeichert werden um einen Puffer aufzubauen auf den in Latenzfällen zugegriffen werden kann. Dies bedeutet jedoch auch, dass das Smartphone kontinuierlich in periodischen Abständen Sensorwerte senden muss, damit der Puffer gefüllt ist. Liegen die Daten vor, kann der vom Programmierer entwickelte Algorithmus Ausgaben auf dem Smartphone auslösen.

Ausgabe-Kommandos sind zwar ebenfalls von Latenzen betroffen, haben allerdings keine direkte Auswirkung auf die Entscheidungsstruktur eines Programms. Sollten Rückgabewerte trotzdem nötig sein und anfallen, sind Sie nicht von häufigen Änderungen betroffen auf die im Zweifelsfall reagiert werden muss, sondern dienen vielmehr der Ausführungs-Verlässlichkeit. So kann überprüft werden, ob die Änderungen an den Ausgabemöglichkeiten wirklich stattgefunden haben. Die Smartphone-Anwendung muss diese jedoch in erster Line einmal bereitstellen. Die Ausgabemöglichkeiten sind in Tabelle 2.1 erwähnt. Sie umfassen die Ausgabe über eine LED, ein Texteld und der Vibration des Gerätes.
Die LED muss muss zwischen zwei Farben wechseln können um ein Änderungen zu signalisieren.
Das Textfeld muss alphanumerische Zeichen ausgeben können um sowohl Werte. Sowohl die Ausgabe von Zeichen als auch Werten ist optimal. Sonderzeichen sollen ebenfalls möglich sein. UTF-8 ist als unterstützer Zeichensatz nötig. Die Länge soll jedoch beschränkt sein und nur der Ausgabe von einzelnen Werten oder Wörtern dienen, nicht der Ausgabe von ganzen Absätzen.
Die Vibrationsausgabe ist eine haptische Ausgabe, keine Optische. Für diese muss eine Zeitdauer konfigurierbar sein um Vibrationsmuster konfigurieren zu können.

Die erwähnten Anforderungen werden in der Architektur berücksichtigt. Sie bilden gewissermaßen den Rahmen für den Aufbau des Frameworks. Rahmenbedingungen die die Einzelheten der Komponenten allgemein definieren, die jedoch auch das Zusammenspiel und die Koordination festsetzen. Diese sind insbesondere für über die Arbeit hinausgehende Erweiterungs-, Skalierungs-, Orchestrierungs- und Automatisierungsmöglichkeiten hilfreich. Eine längerfristige, aufbauende Entwicklung des Frameworks soll möglichst bequem und kompfortabel möglich sein.

Kapitel 3

Architektur

Die im vorigen Kapitel behandelten Softwareanforderungen und Einschränkungen bestimmen den Aufbau des Frameworks um den es in diesem Kapitel geht. Insgesamt besteht das Framework aus einer mobilen Anwendung für Android-Smartphones, einer programmiersprachenunabhängigen Bibliothek für Sensoranfragen und Ausgabe-Kommandos und einem Kontrollprogramm das den Nachrichtenaustausch koordiniert. Der Aufbau ist in Abbildung 3.1 dargestellt. Das Framework arbeitet in drei

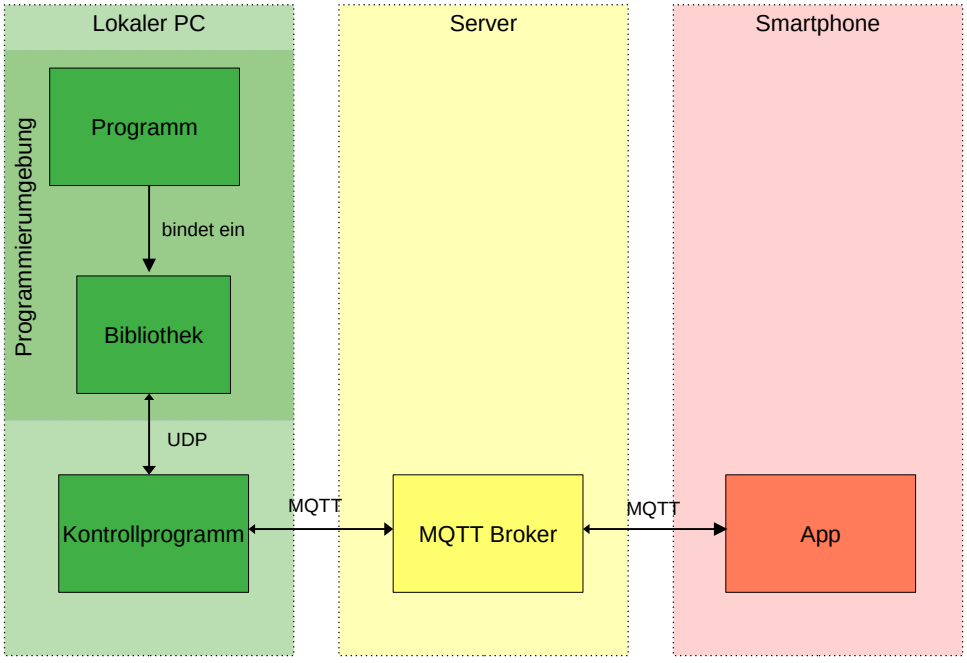

Abbildung 3.1: System-Aufbau

Bereichen: Auf dem lokalen PC, auf einem Server und auf dem Smartphone. Auf dem lokalen PC platziert sind sowohl das Kontrollprogramm, die Bibliothek welche die Aufrufe beinhaltet und das vom Entwickler geschriebene Programm. Programm und Bibliothek bilden zusammen die Programmierumgebung von der aus der Entwickler die volle Funktionalität des Frameworks nutzen kann, indem er in seinem Programm die Bibliothek einbindet. Diese bildet ein standardisiertes Interface für Funktionsaufrufe. Startet ein Entwickler einen Aufruf der Bibliothek, kommuniziert diese ihn per UDP dem Kontrollprogramm.
Das Kontrollprogramm erfüllt drei Aufgaben: Die Beantwortung von Sensoranfragen der Bibliothek, dem Zwischenspeichern von Sensorwerten als Puffer und dem Weiterleiten von Ausgabe-Kommandos auf das Smartphone. Bibliotheksaufrufe werden also per UDP an das Programm gestellt. Bei Sensoranfragen nimmt dieses Sie an und liefert ebenfalls über UDP den zuletzt zwischengespeicherten Wert zurück. Damit dieser auch vorliegt werden von der Smartphone App kontinuierlich, periodisch Sensordaten-Aktualisierungen per MQTT an das Kontrollprogramm gesendet. Das Kontrollprogramm wiederum sendet Ausgabe-Kommandos per MQTT an die Smartphone-App.
Die Smartphone-Anwendung beginnt zum Start mit der Sensor-Messdatenerhebung welche Sie wie geschildert an die Kontrollanwendung sendet. Daneben erwartet Sie Ausgabekommandos der Bibliothek, welche Sie bei Eingang ausführt.
Als Vermittlung zwischen Kontrollprogramm und Smartphone-App dient ein MQTT-Broker. Dieser nimmt Nachrichten beider Kommunikationspartner an und kontrolliert die Zustellung.
Neben der korrekten Funktiosweise der einzelnen Entitäten hat der Austausch von Nachrichten im Framework große Relevanz. So müssen Inhalt und Aufbau definiert sein, damit die inneren Algorithmen der Komponenten korrekt arbeiten können.

3.1 Nachrichtenformate

Ein einheitliches Kommunikationsformat ist für den Nachrichtenaustausch unabdingbar. Der ausgearbeitete Standard definiert Nachrichten in einem Klartextformat. Als Darstellungsform wird JSON verwendet.
Die Nachrichtenformate werden als Vorlagen in einer zentralen Datei abgelegt. Diese werden in der Bibliothek, dem Kontrollprogramm und in der Smartphone-App abgespeichert und eingelesen. So ist zentral definiert welche Nachrichten es gibt, welche Felder und Datenformate Sie beeinhalten. Neben diesen Daten sind zusätzlich noch Bezeichnungen über die nutzbaren Sensoren, sowie den ausführbaren Ausgabe-Kommandos enthalten.
Jede Nachricht weist mindestens die Angabe eines Nachrichtenformat-Typs auf.

Diese sind der Tabelle 3.1 zu entnehmen. Die Spalte *Nachrichtentyp* beschreibt die

Nachrichtentyp	Quelle	Ziel	Netzwerkprotokoll
sensor_request	Bibliothek	Kontrollprogramm	UDP
sensor_response	Kontrollprogramm	Bibliothek	UDP
update_request	Smartphone	Kontrollprogramm	MQTT
rpc_request	Bibliothek, Kontrollprogramm	Smartphone	UDP/MQTT
rpc_response	Smartphone, Kontrollprogramm	Bibliothek	UDP/MQTT

Tabelle 3.1: Nachrichten-Typen

Bezeichnung des Nachrichtentypes. Zu Übersicht sind noch *Quelle* und *Ziel* der Nachrichten beschrieben. Außerdem aufgeführt ist das verwendete Netzwerkprotokoll zur Übermittlung.

*Sensor_request*s werden von der Bibliothek an das Kontrollprogramm gesendet. Ihr Verwendungszweck besteht darin, einen zwischengespeicherten Sensorwert abzufragen. Daher beeinhaltet Sie neben dem Typ-Bezeichner auch noch das Feld `sensor_type`. Nach Eingang ermittelt das Kontrollprogramm den gewünschten Sensorwert und gibt das Ergebnis zurück.
Eine solche Antwort erfolgt im Nachrichtenformat *sensor_response*. Diese beeinhaltet den vom Kontrollprogramm erschlossenen Sensordaten-Wert, `sensor_value`, als String. Sie wird von der Bibliothek entgegengenommen und als float an das aufrufende Programm des Entwicklers zurückgegeben. Da Aufrufe von der Bibliothek immer blockierend im Call-Response-Schema aufgerufen werden kann es nicht passieren, dass sich responses überholen und die Sensorwerte eines anderen Sensors für den angefragten Sensorwert eingetragen werden. Deshalb ist die Angabe des gültigen Sensor-Typs für sensor_responses nicht nötig.
Damit Sensorwerte vorhanden sind muss die Smartphone-App der Kontrollanwendung kontinuierlich gemessene Sensorwerte übermitteln, damit diese in den internen Puffer eingespeichert werden können um bei erhöhten Latenzen Zwischenwerte liefern zu können. Dies geschieht durch den Nachrichtentyp *update_request*. Geltungsgegenstand ist diese Nachricht für alle Art von Sensoren. Übermittelt wird daher sowohl der Sensor-Typ, als auch der gemessene Sensor-Wert um ihn für die Kontrollanwendung schnell zuordenbar zu machen und Sie in den internen Datenpuffer eintragen zu können.
Der gesamte Ablauf für Sensoranfragen ist in Abbildung 3.2 zusammengefasst dargestellt. Zu sehen sind die drei Komponenten Bibliothek, Kontrollprogramm und App auf dem Smartphone. Die Bibliothek sendet sensor_requests per UDP an das Kontrollprogramm. Dieses antwortet über UDP mit einer sensor_response. Während-

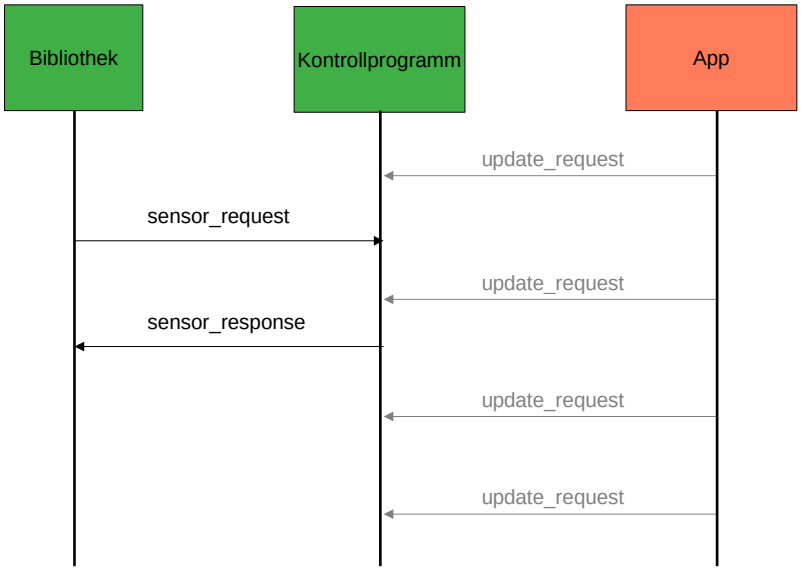

Abbildung 3.2: Nachrichtenablauf der Sensordatenübermittlung

dessen sendet die App auf dem Smartphone kontinuierlich update_requests um neue Messergebnisse der Sensorwerte einzuspeichern.

Neben den Sensordaten betreffenden Nachrichten existieren auch Ausgabe-Kommandos um Ausgaben in der Smartphone-App umzusetzen. Unterscheidbar sind solche Anfragen die lediglich ausgeführt werden sollen und solche , die einen Rückgabewert erwarten. Für erstere gibt es den Nachreichtentyp *rpc_request*. RPC steht für Remote Procedure Call und bezeichnet Funktionsaufrufe die auf einem Client aufgerufen, jedoch auf einem Server ausgeführt werden. Die Bezeichnung entspricht so nicht dem Konzept, da die die Smartphone-App in diesem Fall die Rolle des Servers einnähme. Die Voraussetzung einer Client-Server-Anwendung ist nicht gegeben. Vielmehr tauschen hier gleichwertige Kommunikationspartner Daten aus. Die Bezeichnung wurde eher unter dem Fokus auf die entfernte Ausführung gewählt.
Der Nachrichtentyp enthält die Felder `command` und `value`. Ersters beeinhaltet eines der spezifizierten Ausgabe-Kommandos. Das Zweite beschreibt die Größe des Parameters für das Ausgabekommando. Es ist fast immer befüllt. Vereinzelt gibt es jedoch auch Kommandos die keinen Parameter benötigen. Dann bleibt dieses Feld leer. Die Nachricht wird von der Bibliothek, aufgrund eines Aufrufs des Programmes des Entwicklers erst per UDP an die Kontrollanwendung und von dort aus per MQTT an das Smartphone gesendet. Die Smartphone-App überprüft bei Eingang den Aufruf und setzt die gewünschte Ausgabe entsprechend um.

Manche Kommandos führen nicht nur Ausgaben aus, sondern erheben zusätzlich noch einen Rückgabewert. Damit dieser vom Smartphone zurück an die Bibliothek gesendet werden kann gibt es das Nachrichtenformat *rpc_response*. Dieses wird erst per MQTT an das Kontrollprogramm und von dort aus per UDP an die Bibliothek gesendet. Wie bei sensor_responses können sich die Antworten nicht gegenseitig überholen, was die Übertragung des zugrundeliegenden Ausgabe-Kommandos überflüssig macht. Nur der ermittelte Wert des Kommandos ist relevant und wird in der Nachricht übermittelt.
Der Nachrichtenablauf wird in Abbildung 3.3 nochmal übersichtlich zusammengefasst. Zu sehen sind die drei Komponenten Bibliothek, Kontrollprogramm und App

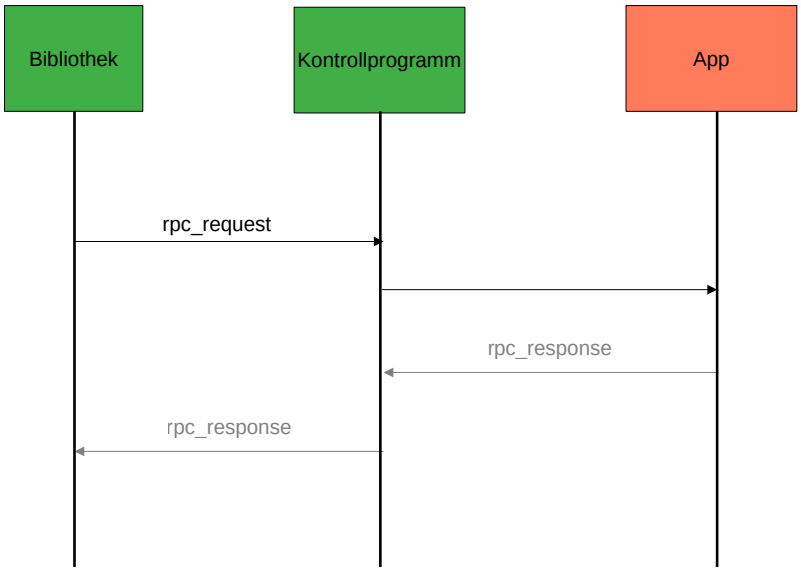

Abbildung 3.3: Nachrichtenablauf der RPC-Anfragen

auf dem Smartphone. Die Bibliothek sendet rpc_requests per UDP an das Kontrollprogramm. Dieses leitet die Nachricht per MQTT direkt weiter an die App. Dort wird das gewünschte Kommando ausgeführt. Fällt ein Rückgabewert an, wird eine rpc_response generiert und per MQTT zurück gesendet. Die Kontrollanwendung leitet die Nachricht dann per UDP weiter an die Bibliothek.

Die erstellten Nachrichtenformate decken alle Nutzungszenarien vollständig ab. Sie bilden ein solides, effizientes Grundgerüst mit Möglichkeit zur Erweiterung. Diese werden auch vom Serialisierungsformat JSON unterstützt. Neue Kommandos mit größeren Parameterwerten sind simpel umzusetzen. Neben Text könnten beispielsweise auch Binärdaten wie Bilder oder Sound-Dateien übertragen werden. Limitiert

werden Sie nur von den darunterliegenden Protokollen UDP und MQTT.

3.2 MQTT

MQTT steht für Message Queuing Telemetry Transport und ist ein Client-Server Protokoll, das auf TCP basiert. Eingesetzt wird es häufig bei Machine-to-Machine Interaktionen, also Lösungen im IoT-, Smarthome und Automatisierungstechnik-Bereich. Durch den platzsparenden Header von 2 Bytes und einer maximalen Nachrichtengröße von ca. 260 MB ist es gleichzeitig ein leichtgewichtiges jedoch auch flexibles Protokoll zum Nachrichtenaustausch. Nachrichten können verschlüsselt übertragen werden. TLS fungiert dabei auf dem Transportweg als Verschlüsslungsprotokoll.

Der Nachrichtenaustausch erfolgt nach dem Observer-Pattern. Als Publizierungs- und Empfangsplattform dienen hier sogenannte Topics. Sie funktionieren wie ein Posteingang, dienen allerdings auch der Kategorisierung und Trennung von unterschiedlichen Nachrichten-Kontexten. Topics können hierarchisch in Topics und Sub-Topics geschichtet werden. Clients können beim Senden einer Nachricht angeben, auf welchen Topics Sie die Nachricht publizieren möchten. Clients die die Nachricht empfangen möchten müssen das gewünschte Topic vorab abbonieren. Neben dem Abbonieren von einzelnen Topics, gegebenenfalls Subtopics, können über Eltern-Topics auch deren Subtopics aggregiert abboniert werden.
Als Vermittlungsstelle dient ein MQTT-Broker. Dieser bildet die Server-Seite im Client-Server-Konzept. Senden Clients Nachrichten an ein Topic werden diese vom Broker entgegengenommen und an die abbonierten Clients zugestellt. Clients können sowohl Sender als auch Empfänger sein. Das Empfangen der eigenen gesendeten Nachrichten ist möglich.
Alle Clients besitzen darüber hinaus einen Namen, so, dass empfangene Nachrichten auf den Sender zurückgeführt werden können. Topic-Interne Filter sind ebenfalls möglich, um beispielsweise nur Nachrichten von bestimmten Clients zu empfangen.
Nachrichten können mit einer Quality of Service (QoS), einer Übertragungsgüte, übertragen werden. Sie gelten sowohl auf dem Weg vom Broker zu einem abbonierendem Client als auch von einem sendenden Client zum Broker. Topics können also mit einer Güte abboniert und Nachrichten mit einer Güte gesendet werden. Diese sind in Tabelle 3.2 dargestellt. Aufgeführt ist die Übertragungsgüteklasse und das dazugehörige Zustellprinzip. Letzteres beschreibt die Übertragungsgarantie für Empfänger.
Insgesamt gibt es drei Güteklassen: 0, 1 und 2. Je höher die QoS-Stufe desto höher die Übertragungsgüte. Im Gegenzug steigt mit der Übertragungsgüte allerdings auch die Latenz der Übertragung.

QOS-Level	Zustellprinzip
0	At most once
1	At least once
2	Exactly once

Tabelle 3.2: Aufgaben

QoS-Stufe 0 ist die geringste Gütestufe. Hier werden Nachrichtern einmal übertragen. Es wird nicht sichergestellt, dass die Nachricht schlussendlich zugestellt wird. Der Sender überträgt die Nachricht nur einmal an den Broker und dieser Sie auch nur einmal an den Empfänger. Es gibt keine Übertragungsbestätigung an keiner Zwischenstation. Deshalb kann ist die Garantie lediglich dass die Nachricht im besten Falle einmal zugestellt wird.
Bei der QoS Stufe 1 muss zwischen Sender und Empfänger unterschiden werden. Sender senden die Nachricht immer wieder, bis Sie eine Bestätigung vom Broker bekommen. Dieser sendet Sie immer wieder an den Empfänger bis dieser den Eingang einmal bestätigt. Es kann durch die wiederholte Sendung jedoch vorkommen, dass eine Nachricht mehrmals gesendet wird. Deshalb ist die Garantie hier lediglich *mindestens einmal*, da nicht sichergestellt werden kann ob eine Nachricht mehrmals eingegangen ist.
QoS 2 ist die höchste Gütestufe. Sender übertragen die Nachricht einmal an den Broker und warten dann auf die Bestätigung, senden also nicht fortwährend die gleiche Nachricht bis Sie einmal bestätigt wird. Der Broker sendet Sie dann einmal an alle abbonierten Clients und wartet auf die Bestätigung. Nachteilig ist hier vor allem die hohe Wartezeit sowie das Fehlschlagen einer Übertragung, sollte das Timeout auslaufen. Dafür kann der einmalige Empfang der Nachricht garantiert bestätigt werden.

Wie eingangs erwähnt ist die Übertragungsgüte sowohl von sendenden Clients, als auch von abbonierenden Client individuell einstellbar. Damit die verlangte Güte sichergestellt werden kann müssen Sender und Empfänger mit der gleichen Güte senden bzw. ein Topic mit der gleichen Güte-Stufe abbonieren. Wird ein Topic mit einer geringeren Güte abboniert, als die Nachricht versendet wird, wird die Nachricht mit der geringeren Güte des Abbonenten übertragen. Wenn eine Nachricht mit geringerer Güte auf einem Topic publiziert wird, als Clients das Topic abboniert haben, wird die Übertragung mit der geringeren Güte-Stufe des publizierenden Clients übertragen. Zusammenfassend gilt: Unterscheiden sich Sender- und Abbonenten-Güteklassen geschieht die Übertragung mit der geringeren Güteklasse.

Kapitel 4

Android Anwendung

Die Android-Anwendung ist eine der drei zentralen Bestandteile des Frameworks. Sie dient dazu Sensormessprozesse zu starten, Sensordaten zu übermitteln und Ausgabe-Kommandos auszuführen. Für diese bietet Sie unterschiedliche UI-Elemente in einer Activity, der RootActivity ein. Sie ist zentrale UI Schnittstelle für sämtliche Formen der Ausgaben. Diese sind: eine Signal-Led und ein Textfeld. Neben UI Elementen gibt es zusätzlich noch die haptische Ausgabe der Vibration.
Für die Grundfunktionalität benötigt die Kontrollanwendung des Frameworks Sensordaten welche Anschließend versendet werden müssen. Versandt werden die Nachrichten über einen Service, der im Hintergrund ausgeführt wird, dem MQTT-Service. Die Sensordatenerhebung funktioniert über sogenannte SensorEventListener. Sie starten Sensormessprozesse und überwachen diese auf neue Werte. In einem solchen Fall senden Sie über den MQTT-Service update_requests an das Kontrollprogramm. Die Anwendung versendet neben dieser Nachrichtenart auch noch rpc_response-Nachrichten. Erstellt werden diese von der Klasse JsonMessageWrapper.

4.1 Funktionsablauf

Die Erhebung der Messwerte erfolgt zum Start der Anwendung. Ein Ablaufplan ist in Abbildung 4.1 zu sehen. Asynchrone Schritte sind mit einem *A* markiert.

In der Root-Activity werden zuerst alle UI-Elemente eingebunden um Sie über Kommandos zu manipulieren. Anschließend werden Konfigurationsdaten eingelesen. Insgesamt gibt es zwei Konfigurationsdateien: config.json und protocol.json. In Ersterem ist zum Beispiel der Hostname des MQTT-Brokers, der Port oder das Topic definiert. Diese Daten sind für die Übermittlung der Nachrichten per MQTT wichtig. In protocol.json wird die Form der für das Smartphone relevanten Nachrichtenforma-

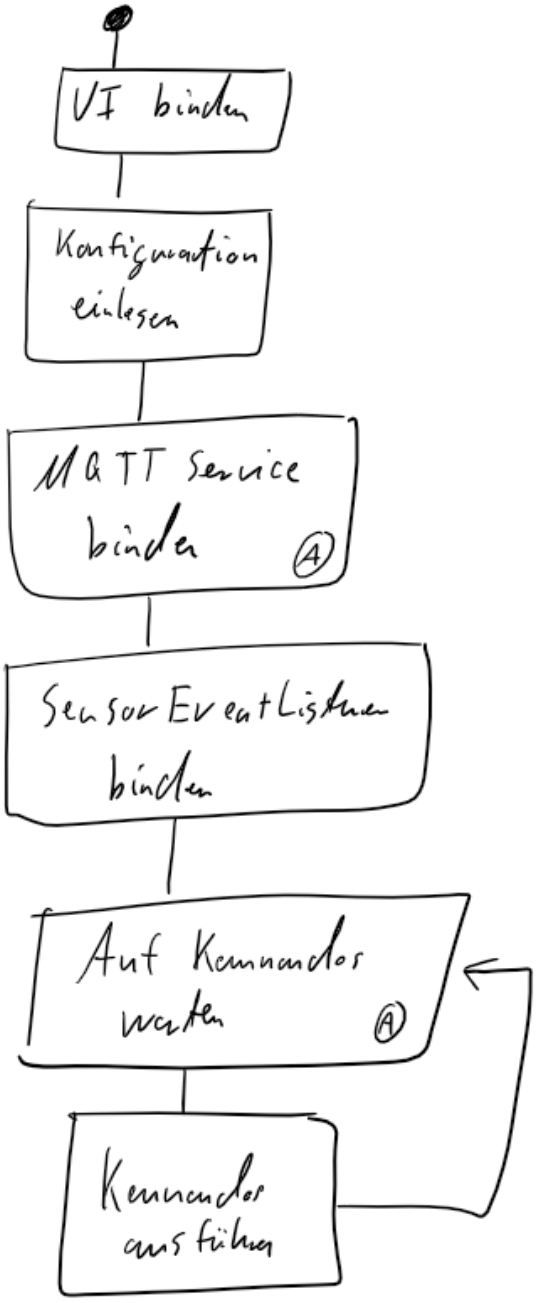

Abbildung 4.1: Ablaufdiagramm Android Anwendung

te update_request und rpc_response definiert. Außerden sind dort die unterstützten Sensoren und Ausgabekommandos beschrieben. Relevant werden diese bei der Entscheidungsfindung bei Eingang eines rpc_requests, wenn determiniert werden muss welche Aktion die Nachricht beeinhaltet.
Nach dem Einlesen der Konfigurationen wird der zur Kommunikation verwendete MQTT Service eingebunden. Dies geschieht asynchron. Über eine ServiceConnection wird beim erfolgreichen einbinden über eine Callback-Methode der weitere Verlauf definiert. Die Root-Activity speichert dann die Referenz auf den Service und der Service die Referenz auf die Root-Activity. Grund für dieses gegenseitige Einbinden ist, dass Nachrichten im MQTT Service in einem seperaten Thread behandelt werden. Bei RPC-Requests müssen jedoch UI-Elemente verändert werden können. Dies ist ohne weiteres nicht aus dem Service heraus möglich. Mit einer Referenz auf die Activity kann der Service UI-Ändernde Funtionen auf der Activity ausrufen. Android unterbindet jedoch UI-Manipulationen durch Threads die nicht der UI-Thread sind. Dieses Problem wird durch die Methode *runOnUiThread* umgangen, welche die Änderung in der Ausführungswarteschlange des UIThreads einreiht. Der Service baut eine Verbindung zu einem MQTT Server auf.
Beim Einbinden des MQTT Servers stellt dieser eine Verbindung zu einem in config.json definierten MQTT-Broker und Topic her.
Ist der Service final eingebunden können die Sensormessprozesse gestartet werden, da Messdaten nun zuverlässig gesendet werden können. Verschiedene SensorEventListener werden nun gestartet und zentral in einem SensorEventListenerContainer gesammelt gespeichert und die Messprozesse jeweils angestoßen. Somit ist die Startroutine der Mobilen Anwendung abgeschlossen. Auf Nachrichten wird nun nur noch im MQTT-Service in einem MessageListener mit entsprechendem Callback reagiert.

Die Funktionsweise der Sensordatenübertragung wird in Abbildung 4.2 nocheinmal zusammenfassend dargestellt. Die Klasse *SmartBitEventListenerContainer* beeinhaltet SensorEventListener für alle Arten von untertstützten Sensoren. Der Container dient lediglich der Datenhaltung. Aufgabe der SensorEventListener ist es auf Sensorwert-Änderungen zu reagieren und eine entsprechende Callback-Funktion aufzurufen. In dieser werden dann über statische Methoden der Klasse *JSONMessageWrapper* update_requests generiert und der gemessene Wert eingesetzt. Die so generierte Nachricht wird anschließend über den gebundenen MQTT-Service an das vorher definierte Topic versendet.

Die Anwendung ist nun betriebsbereit und beginnt bereits erste Nachrichten an die Kontrollanwendung zu senden. Übermittelt werden die Sensordaten an den Broker mit einer QOS-Stufe von 0. Verluste von update_requests sind unproblematisch, da es je nach Taktung sehr schnell neue Sensorwerte gibt die übertragen werden

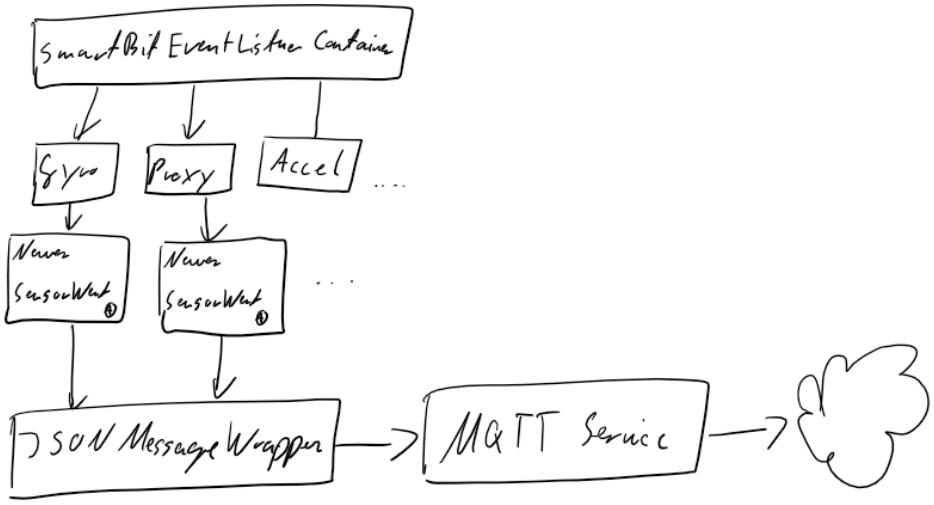

Abbildung 4.2: Ablaufdiagramm SensorEventListener

können. Eine exakte Zustellung ist hier nicht notwendig und verlangsamt eher den Übertragungsprozess.

4.2 Sensoren

Smartphones beeinhalten verschiedene Sensoren die Daten über die Umgebung erfassen können. In der Android Anwendung werden folgende Sensortypen verwendet:

- Lineare Beschleunigungssensoren
- Mikrofon
- Annäherungssensor
- Gyroskop

Beschleunigungs- bzw. Lagesensoren messen die Beschleunigung in m/s^2 für die drei Bewegungsrichtungen: X-, Y- und Z-Achse in einem festgelegten Zeitraum. Eine Übersicht über die Anordungen der drei Axen ist in Abbildung 4.3 zu sehen. Die X-Achse verläuft horizontal durch das Display des Smartphones hindruch, die Y-Achse vertikal und die Z-Achse durchschneidet das Smartphone in die Tiefe.

Die Frequenz mit der Messwerte erstellt werden kann manuell angegeben werden. Hierfür stehen vier Schnelligkeitsstufen bereit.:

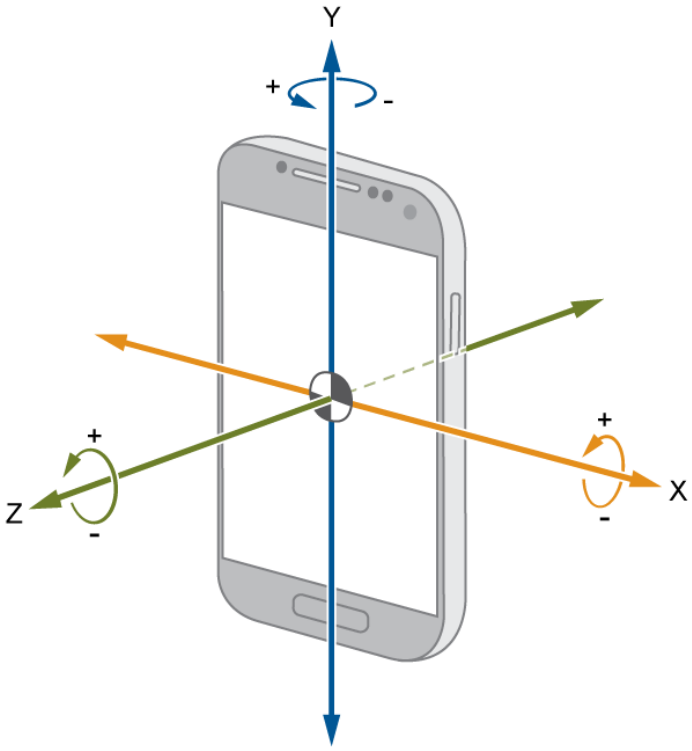

Abbildung 4.3: Android-Koordinatensystem

- SENSOR_DELAY_FASTEST : Kein Verzögerung. Verwendet die Frequenz des Sensors.
- SENSOR_DELAY_GAME : Verzögerung um 1ms
- SENSOR_DELAY_UI : Verzögerung um 2ms
- SENSOR_DELAY_NORMAL : Verzögerung um 3ms

Die mit der jeweiligen Frequenz aufgenommenen Beschleunigungssensordaten beinhalten jedoch auch die Ergbeschleunigung. Diese muss für die bereinigten, realen Werte zuerst noch von den aufgenommenen Werten subtrahiert werden[6].

4.3 Angaben für die Nachrichtenformate

Alle Sensordaten besitzen ein festgelegtes Kürzel zur Standardisierung des Nachrichtenverkehrs. Sie dienen vor allem der Adressierung der jeweiligen Daten in der Middleware und in der Library.
Die Sensortyp Kürzel sind in Tabelle 4.1 zu finden.

TYPE-Kürzel	Beschreibung
accell_x	Lagesensor für die X-Richtung
accell_y	Lagesensor für die Y-Richtung
accell_z	Lagesensor für die Z-Richtung
gyro_x	Gyroskopsensor für die X-Richtung
gyro_y	Gyroskopsensor für die Y-Richtung
gyro_z	Gyroskopsensor für die Z-Richtung
prox	Näherungssensor

Tabelle 4.1: Sensor-Kürzel mit Beschreibung

4.4 Kommandos und Ausgaben

Für die Ausgabe auf dem Smartphone sind verschiedene Kommandos definiert. Diese sind der Tabelle **??** zu entnehmen. `CMD-Kürzel` beschreibt die Notation des Kürzels mit dem eine Aktion ausgeführt werden kann. Diese wird unter `Beschreibung` kurz zusammengefasst. `return` gibt an, ob der Aufruf des Requests eine Antwort rücksendet und somit auch, ob ein Aufruf der Funktion in der Library blockiert oder nur sendet.

Kapitel 5

Kontrollanwendung

Für Vermittlung zwischen den Komponenten fungiert eine Server-Anwendung. Sie ist in Python geschrieben und vermittelt zwischen UDP-Anfragen auf der einen Seite vom Client aus und MQTT-Anfragen vom Smartphone auf der anderen Seite. User können über die Library RPC-Anfragen oder Sensor-Anfragen an den Server stellen. Dies geschieht in form von JSON-Anfragen die per UDP übermittelt werden. Der Server ist unter der localhost-Adresse 127.0.0.1 auf dem Port 5006 erreichbar.
Für die MQTT-Verbindung kommt dabei die unter OpenSource-Liznenz stehende MQTT-Library Paho der Eclipse-Foundation zum Einsatz. [4]

5.1 Anforderungen

Aus den Anforderungen ergeben sich folgende Aufgaben die der Server umsetzen muss:

- Sensoranfragen schnell beantworten
- Anfragen die Funktionen auf dem Smartphone starte, müssen schnell an das Smartphone weitergereicht und gegebenfalls wieder beantwortet werden.

Damit Sensoranfragen schnell beantwortet werden können, werden die aktuellen Sensorworte fortlaufend vom Smartphone an den Server übermittelt. Dieser speichert Sie dann, sobald ein Sensorwert gemessen und übermittelt wurde in eine interne, threadsichere Datenstruktur. Hierdurch wird insbesondere der Latenzunterschied zwischen Smartphone und Localhost berücksichtigt. UDP Anfragen per Localhost haben eine wesentlich geringe Latenz als MQTT-Anfragen, auf die das Smartphone reagieren muss. Durch das Cachen auf dem Server liegen bei Anfragen per UDP, also von der Library aus, immer Sensordaten vor.

Um Funktionsanfragen zeitnah auszuführen müssen unterschiedliche Netzwerklatenzen berücksichtigt werden. Die Übermittlung erfolgt asynchron zwischen Library und Smartphone, wobei zu erwarten ist dass Übermittlungen an das Smartphone mit einer höheren Latenz übertragen werden, als Anfragen zwischen Library und Server, da sich beide auf dem gleichen Host befinden. Die Verbindung zwischen Library und Server erfolgt darüber hinaus über localhost und somit über ein Loopback-Device. Loopback-Devices reichen Netzwerkpakete nicht herkömmlich über ein physisches Netzwerkinterface weiter sondern stellen ein virtuelles Gerät dar, dessen Übertragungsrate an die CPU gekoppelt ist. Die Bandbreiten sind dadurch sehr hoch und die Latenzen gering.
Um die Latenzen und die damit verbundenen unterschiedlichen Auftrittszeitpunkte von Anfragen zu berücksichten, sowie eine möglichst effiziente Abarbeitung der selbiger zu ermöglichen wird eien paralelle Algorithmusstruktur verwendet.

5.2 Interner Aufbau

Die Serveranwendung mit dem Namen server.py ist aufgeteilt in einen Datenverwaltungsteil, DataHandler, und eine MQTT Anbindung, MQTTHandlerThread. Da die Sensorwerte des Smartphoens vorrätig gehalten werden, wird außerdem eine Datenklasse für diese, SensorDB, intern gehalten. Eine Übersicht über die Komponenten ist Abbildung 5.1 zu entnehmen.

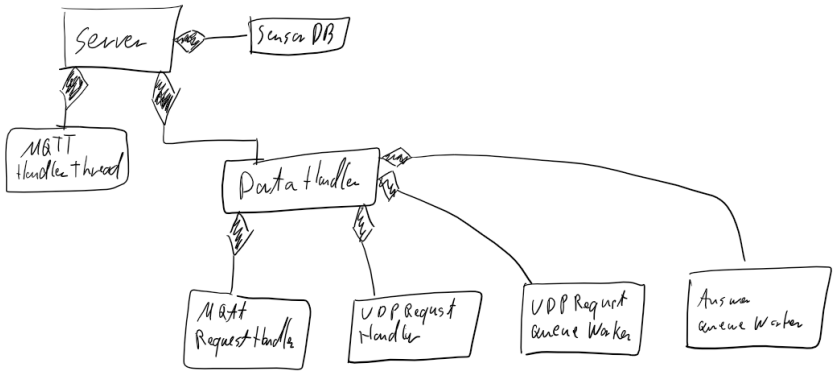

Abbildung 5.1: UML Digaramm Server

DataHandler wiederrum teilt sich nochmal auf in vier seperate Funktionen, die als Threads nebenläufig laufen: MqttRequestHandler, UDPRequestHandler, UDPRequestQueueWorker und AnswerQueueWorker.
Die Funktionsweise und Zwecke dieser Threads wird im Folgenden an zwei Beispielen erläutert. Für das erste Beispiel wird Abbildung 5.2 betrachtet. Zu sehen ist ein

MQTT-Request, also eine Anfragen des Smartphones, dass über MQTT an den Server gesendet wird. Schnittstellen zu MQTT sind in der Abbildung grün, Schnittstellen zur Library per UDP, sind blau markiert.

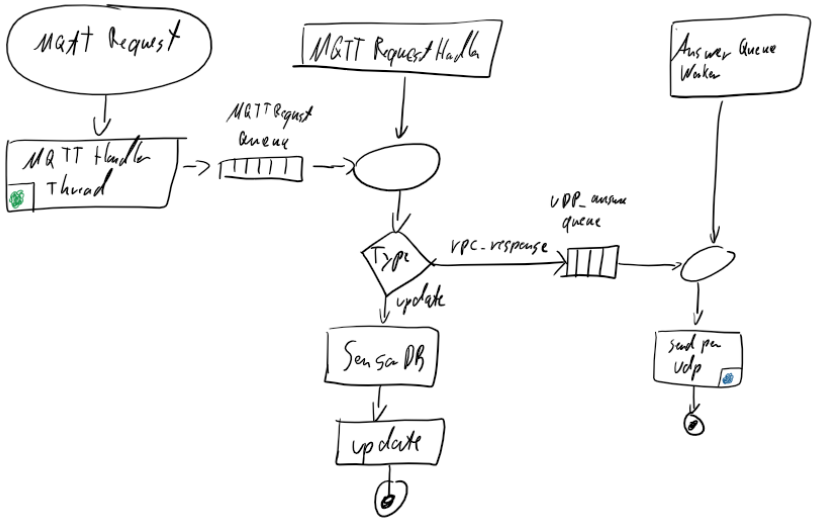

Abbildung 5.2: Ablaufdiagramm MQTT Request

Erreicht ein MQTT Request den Server wird es im MQTTHandlerThread entgegengenommen. Dieser setzt die Nachricht in eine MQTTRequestQueue ein. Der MQTTRequestHandler des DataHandlers wartet bis ein Eintrag in der Queue vorhanden ist und nimmt gegebenenfalls eine Nachricht. Daraufhin wird derTyp des Requests bestimmt. Handelt sich um ein Sensorupdate muss nur der Sensorwert in der Datenbank aktualisiert werden. Handelt es sich um eine rpc_response, also um eine Antwort auf eine vorausgegangenes rpc_request, dass einen Rückgabewert fordert, wird das request in eine udp_answer_queue eingefügt. Der AnswerQueue Worker wartet, ähnlich wie der MQTT Request Handler, bis eine neue Nachricht vorhanden ist die per UDP an den Client gesendet werden soll und sendet diese dann gegebenfalls ab.

Das zweite Beispiel befasst sich mit dem Ablauf eines UDP-Requests, also einer Anfrage die mithilfe der Library gesendet wurde. Der Ablauf ist in Abbildung 5.3 dargestellt. Wie auch schon in der letzten Abbildung sind alle Schnittstellen zum Smartphone grün und alle Schnittstellen zur Library blau markiert. Erreicht ein UDP Request den Server wird es vom UDPRequestQueue-Worker in eine UDP Request Queue gelegt. Der UDPRequestHandler-Thread entnimmt die Nachricht und bestimmt den Anfragentyp. Handelt es sich um eine Anfrage des Types RPC_Request soll sie Aktionen auf dem Smartphone auslösen. Sie muss an das Smartphone gesen-

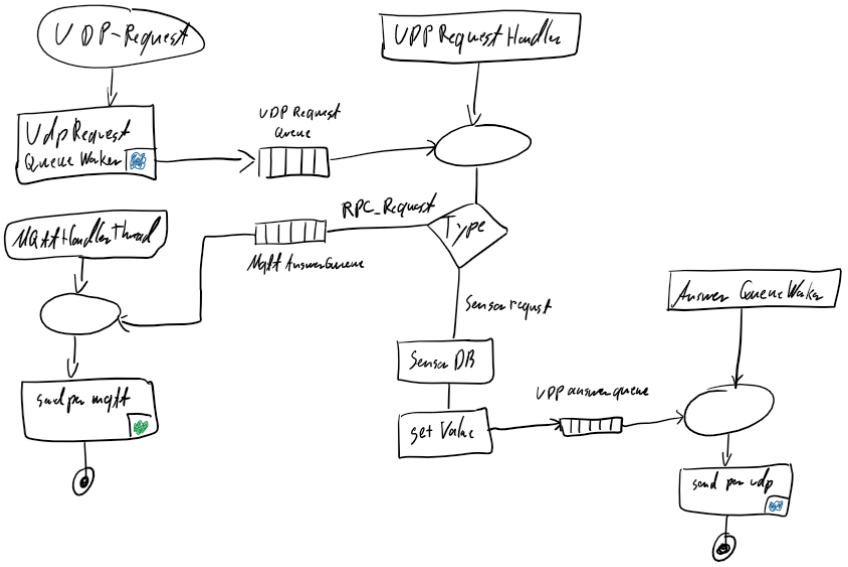

Abbildung 5.3: Ablaufdiagramm UDP Request

det werden, was über MQTT möglich ist. Dafür wird Sie ineine MqttAnswerQueue eingesetzt. Der MQTTHandlerThread entnimmt Sie und sendet Sie per MQTT ab.
Ist Request hingegen ein SensorRequest, also eine Anfrage auf die ein Sensorwert geantwortet werden soll wird der nachgefragte Sensorwert über die Klasse SensorDB entnommen und in die Udp_answer_queue eingetragen. Der AnswerQueueWorker entnimmt die Anfrage und sendet Sie per udp an die Library zurück.

Zusammenfassend erfüllen die Komponenten folgende Aufgaben. Der MQTTHandlerThread nimmt Nachrichten direkt per MQTT an und gibt die Anfrage weiter. Außerdem sendet er Nachrichten per MQTT, falls welche anfallen.
Der MQTTRequestHandler kümmert sich um das Verfahren von MQTT Requests.
Der UDPRequestQueue Worker nimmt wie der MQTTHandlerThread Anfragen die per UDP übermittelt wurden an und gibt Sie zur Behandlung entsprechend weiter. Er sendet jedoch im Gegensatz keine Responses zurück.
Hierfür gibt es den AnswerQueueWorker, dessen einzige Aufgabe es ist Antworten per UDP zurück zu übermitteln.

Die Kommunikation zwischen den Threads funktioniert über synchronisierte Queues des queue-Moduls[9] der cpython Implementierung. Es handelt sich um eine threadsichere Monitorklasse, die einen gleichzeitigen Zugriff zweier unterschiedlicher Threads durch Locks verhindert.

Kapitel 6

Programmierumgebung

Die Bibliotheken bilden die Einstiegsstelle für Nutzer um die Anwendung fernzusteuern oder Sensorwerte abzufragen. Funktionsaufrufe liegen jeweils in den Sprachen C, Java und Python vor. Daten werden wie in Kapitel 3 in Abbildung 3.1 gezeigt per UDP an eine Serveranwendung gesendet. Diese sendet die Daten dann nach gegebenenfalls per MQTT an das Smartphone weiter, oder wieder per UDP zurück. Die Bibliotheken senden und empfangen alle jeweils Daten per UDP. Hierfür werden Sockets benötigt. Für's Empfangen muss der Socket binded sein. Die Serveranwendung ist auf dem Port 5006 erreichbar. Antworten erwarten die Bibliotheken auf dem Port 5005.

Kapitel 7

Evaluation

7.1 Android Anwendung

Für Ausgaben auf dem Smartphone existiert die Android Anwendung Smartbit, die verschiedene Interaktionsmöglichkeiten bereitstellt. Das Userinterface ist in Abbildung 7.1 dargestellt..
Die Anwendung besteht aus zwei Leds, einer Signal- und einer Funktions-LED Ein Textfeld in der Mitte kann zur Ausgabe von Text verwendet werden. Die beiden Buttons können gedrückt werden. Neben den sichtbaren Elementen kann die Anwendung das Smartphone außerdem vibrieren lassen.
Die Signal-LED wird für die Ausgabe verwendet. Aufgabe der Funktions-Led ist die Anzeige eines aktuell ausgeführten Kommandos aufgrund eines rpc_request. Leuchtet die LED rot wird gerade ein Vorgang abgearbeitet. Anschließend verblasst Sie wieder um einen Leerlauf anzuzeigen. Dieses Designelement wurde zusätzlich implementiert, da gerade sich wiederholende Vorgänge, sowie haptische Vorgänge wie das Vibrieren kein optisches Signal aussenden. Hält der Anwender das Smartphone beispielsweise nicht in der Hand oder liegt es nicht auf einer harten Oberfläche auf der die Vibration hörbar ist, ist das Vibrieren nicht wahrnehmbar.

7.2 Kontrollanwendung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig ob ich schreibe: »Dies ist ein Blindtext« oder »Huardest gefburn«? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander

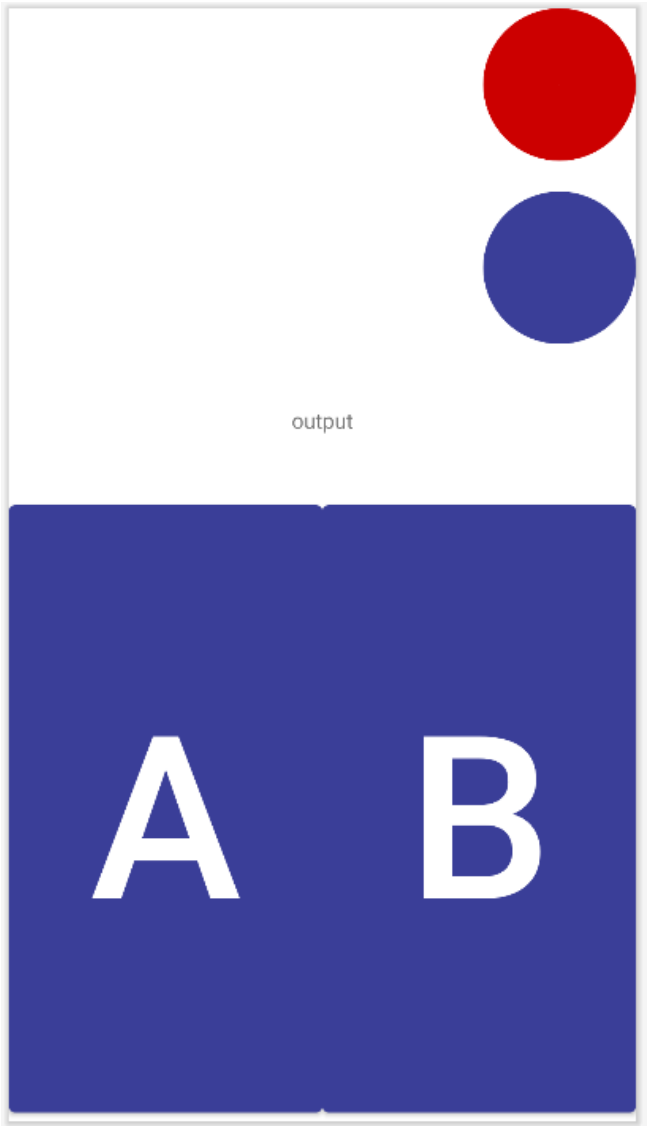

Abbildung 7.1: Userinterface Android Anwendung

stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muß keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie »Lorem ipsum« dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

7.3 Programmierumgebung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig ob ich schreibe: »Dies ist ein Blindtext« oder »Huardest gefburn«? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muß keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie »Lorem ipsum« dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Kapitel 8

Fazit

8.1 Android Anwendung

8.2 Kontrollanwendung

MQTT QOS 2 zu langsam. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig ob ich schreibe: »Dies ist ein Blindtext« oder »Huardest gefburn«? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muß keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie »Lorem ipsum« dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

8.3 Programmierumgebung

Nachteil: JSON Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig ob ich schreibe: »Dies ist ein Blindtext« oder »Huardest gefburn«? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muß keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie

»Lorem ipsum« dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

8.4 Ausblick

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig ob ich schreibe: »Dies ist ein Blindtext« oder »Huardest gefburn«? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muß keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie »Lorem ipsum« dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Kapitel 9

Quellen

Literaturverzeichnis

[1] Martin Winkler Christoph Lüders. c't 23 - pingpong, 2006.

[2] Martinha Piteira and Carlos Costa. Learning computer programming: Study of difficulties in learning programming. In *Proceedings of the 2013 International Conference on Information Systems and Design of Communication*, ISDOC '13, page 75–80, New York, NY, USA, 2013. Association for Computing Machinery.

Online-Quellen

[3] Arduino. Kits-arduini official store. https://store.arduino.cc/collections/kits. [letzter Zugriff: 08. Juni. 2022].

[4] Eclipse. Paho. https://www.eclipse.org/paho/. [letzter Zugriff: 03. Juni. 2022].

[5] Financesonline. Number of smartphone users worldwide 2022. https://financesonline.com/number-of-smartphone-users-worldwide/. [letzter Zugriff: 08. Juni. 2022].

[6] Google. Android developers - accellerometer example. https://developer.android.com/guide/topics/sensors/sensors_motion#sensors-motion-accel. [letzter Zugriff: 04. Juni. 2022].

[7] Google. Sensors - overview. https://developer.android.com/guide/topics/sensors/sensors_overview#sensors-intro. [letzter Zugriff: 09. Juni. 2022].

[8] bitkom. https://www.bitkom.org/Presse/Presseinformation/Mit-10-Jahren-haben-die-meisten-Kinder-ein-eigenes-Smartphone. [letzter Zugriff: 07. April. 2022].

[9] python. queue. https://docs.python.org/3/library/queue.html. [letzter Zugriff: 03. Juni. 2022].