AUTOMATISIERTE LICHTMESSUNG MIT INDOOR-LOKALISATIONSSYSTEM

TECHNISCHER BERICHT

Bachelorarbeit

Autoren
Patrick Scherler
Esteban Luchsinger

Betreuer
Prof. Dr. Farhad Mehta

Industriepartner
Tobias Hofer
HSi Elektronik AG

HOCHSCHULE FÜR TECHNIK RAPPERSWIL
Oberseestrasse 10, 8640 Rapperswil, Switzerland

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Abstract

2 Management Summary

2.1 Ausgangslage

2.2 Vorgehen, Technologien

2.3 Ergebnisse

2.4 Ausblick

3 Rechtliche Hinweise

4 Ausgangslage und Zielsetzung

Dieses Kapitel befasst sich mit der beim Projektstart bestehenden Ausgangslage beim Auftraggeber und der daraus entstandenen Motivation für diese Arbeit.

4.1 Motivation

Die Firma HSi Elektronik AG (Auftraggeber) führt unter anderem Lichtinstallationen bei Kunden im öffentlichen Bereich durch. Um die geforderten Helligkeitswerte der geltenden Standards garantieren zu können, werden entsprechende Lichtmessungen durchgeführt. Dabei setzt der Auftraggeber auf innovative Technologien, wie die Simulation der Lichtverhältnisse vor und nach der Installation. Die effektiven Messungen werden allerdings noch immer manuell mit einem Luxmeter durchgeführt und dokumentiert.

Gemäss den Standards müssten für die Überprüfung eines einzelnen Raumes Messungen in vordefinierten Abständen und Höhen ausgeführt werden. Manuelle Messungen verursachen hohe Zeitaufwände und sind durch die menschliche Ungenauigkeit fehleranfällig. Aus diesem Grund werden in der Praxis meist stichprobenartig Messungen erstellt, um die Daten der Simulation zu bestätigen.

Eine automatisierte und zuverlässige Lösung für die Lichtmessung würde diesen Prozess nicht nur verkürzen, sondern auch die Qualität und Aussagekraft der Messungen verbessern. Dies könnte in Zukunft auch eine regelkonforme Prüfung der geltenden Standards ermöglichen.

4.2 Zweck und Ziel

Ziel dieser Bachelorarbeit ist das Entwickeln einer praxistauglichen Lösung zur lokationsbasierten Ausführung von Lichtmessungen innerhalb eines Raumes. Die Lösung soll weitestgehend automatisiert sein und zwar vor allem dort, wo menschliche Fehler geschehen können. Dem Vermesser, der die Luxmessungen durchführt sollen möglichst alle benötigten Informationen über eine einfache Benutzerschnittstelle angezeigt werden.

Eine detaillierte Beschreibung mit den einzuhaltenden Punkten und einer Abgrenzung ist in der Aufgabenstellung ausgeführt. Nachfolgend sind als Ergänzung dazu alle Abweichungen der Aufgabenstellung aufgelistet, die zu Beginn der Arbeit so definiert wurden:

Anstatt *Beim Erreichen der Punkte im vordefinierten Raster sollen automatisch Lux-Messungen ausgeführt werden.*

Folgendes *Es werden permanent Lux-Messungen ausgeführt und mit den entsprechenden Positionsdaten verknüpft. Die Messwerte können anschliessend im User Interface gefiltert oder kombiniert werden.*

Grund Während einer Messung sollen so viele Messwerte wie möglich gesammelt und erst anschliessend bei der Analyse im User Interface gefiltert werden. So können in Zukunft auch weitere Auswertungen auf bestehenden Daten durchgeführt werden.

4.3 Lieferumfang

Der Lieferumfang dieser Arbeit umfasst folgende Punkte:

- System zur lokationsbasierten Ausführung von Lichtmessungen innerhalb eines Raumes. (Siehe Aufgabenstellung)
- Technischer Bericht mit Architekturdokumentation.

- Source Code Repository und Code Dokumentation.
- Zwischen- und Abschlusspräsentation inkl. Live-Demonstration der Applikation.

4.4 Annahmen und Abgrenzungen

Zusätzlich zur Abgrenzung in der Aufgabenstellung wurden zu Beginn der Arbeit mit dem Auftraggeber und dem Betreuer folgende Punkte definiert:

Ansteuerung Die Ansteuerung des Sensors kann notfalls vom Auftraggeber erledigt oder simuliert werden. In dieser Arbeit soll hauptsächlich das Zusammenspiel der verschiedenen Komponenten, die Visualisierung der Sensordaten und die Positionsbestimmung behandelt werden.

Standards Die in der Motivation erwähnten Standards dienen lediglich als Referenz und müssen von der entwickelten Software nicht zwingend eingehalten werden. Eine entsprechende Lösung zur Einhaltung der Standards wäre in Zukunft allerdings denkbar und könnte vom Auftraggeber an seine Kunden als weiteren Service angeboten werden.

4.5 Einschränkungen

Für dieses Projekt gibt es vor allem Einschränkungen im Bereich der Grösse und Verfügbarkeit des Systems.

- Das komplette System soll mobil sein, damit es ein Vermesser jederzeit zum Kunden mitnehmen kann.
- Die Sensorkomponente muss möglichst portabel sein, damit diese später auf ein Hilfsmittel, wie beispielsweise eine Drohne, passt.
- Eine Messung muss komplett offline durchgeführt werden können und nicht auf Services aus dem Internet angewiesen sein.

4.6 Abgrenzung zu anderen bestehenden Lösungen

- Simulationssoftware, die von der HSi eingesetzt wird. (Relux (CH), Dialux (DE))
- Evtl. Abgrenzung zu IoT?

4.7 Stand der Technik

Eine vergleichbare Lösung zur automatisierten Lichtmessung wurde auf dem Markt nicht gefunden. Die Problemstellung scheint zu spezifisch zu sein oder die Anwendungsfälle dafür zu gering. Eine Zerlegung in Einzelprobleme bringt jedoch die nachfolgenden Lösungen hervor.

4.7.1 Lichtmessung

Für Lichtmessungen gibt es bereits diverse Lösungen auf dem Markt. Im professionellen Bereich werden Beleuchtungsstärkemessgeräte (Luxmeter) mit Genauigkeitsklassen und rückführbarer Kalibrierung eingesetzt. Zur Ansteuerung per Software, wie es auch in dieser Arbeit benötigt wird, können Sensoren eingesetzt werden.

Zusätzlich gibt es seit einiger Zeit diverse Luxmeter-Apps, mit denen ein Smartphone als Luxmeter verwendet werden kann. Eine weitere Recherche auf diesem Gebiet scheint allerdings zwecklos, da die Mobiltelefone bereits aufgrund der verbauten Hardware nicht an die Genauigkeit von professionellen Geräten herankommen[1].

[1] https://www.dial.de/de/article/luxmeter-app-vs-messgeraetsind-smartphones-zum-messen-geeignet/

4.7.2 Positionsbestimmung

Für die Positionsbestimmung in geschlossenen Räumen wird Pozyx verwendet. Dieses verspricht laut Hersteller zentimetergenaue Positionsbestimmung per UWB. Die Entscheidung für diese Technologie wurde bereits vor Beginn der Arbeit gefällt.

GPS kommt für die Positionsbestimmung nicht in Frage, da der Empfang in Gebäuden generell stark reduziert bis unmöglich ist.

4.7.3 Simulation

Relux, Dialux (siehe Abgrenzung)

4.7.4 Kartierung

Die Kartierung umfasst das Verknüpfen der Helligkeitswerte mit den Positionsdaten und der Darstellung im Raum. Dies ist sicherlich der Kern dieser Arbeit.

- Google Maps Indoor
- Pozyx Beta
- elastic.co und GeoHash grid Aggregation (Nachteil: Kosten für Hosting / grosser Ressourcen-Verbrauch)
- HeatmapJS (klein/schlank)

5 Anforderungsanalyse

5.1 Systemkontext

Das in Abbildung 1 gezeigte Systemkontextdiagramm[2] ist ein guter Ausgangspunkt, um das Softwaresystem als Gesamtbild darzustellen. Das zu entwickelnde System, der Flux-Coordinator[3], befindet sich als Blackbox in der Mitte, umgeben von seinen Benutzern und den Sensoren, mit denen es interagiert.

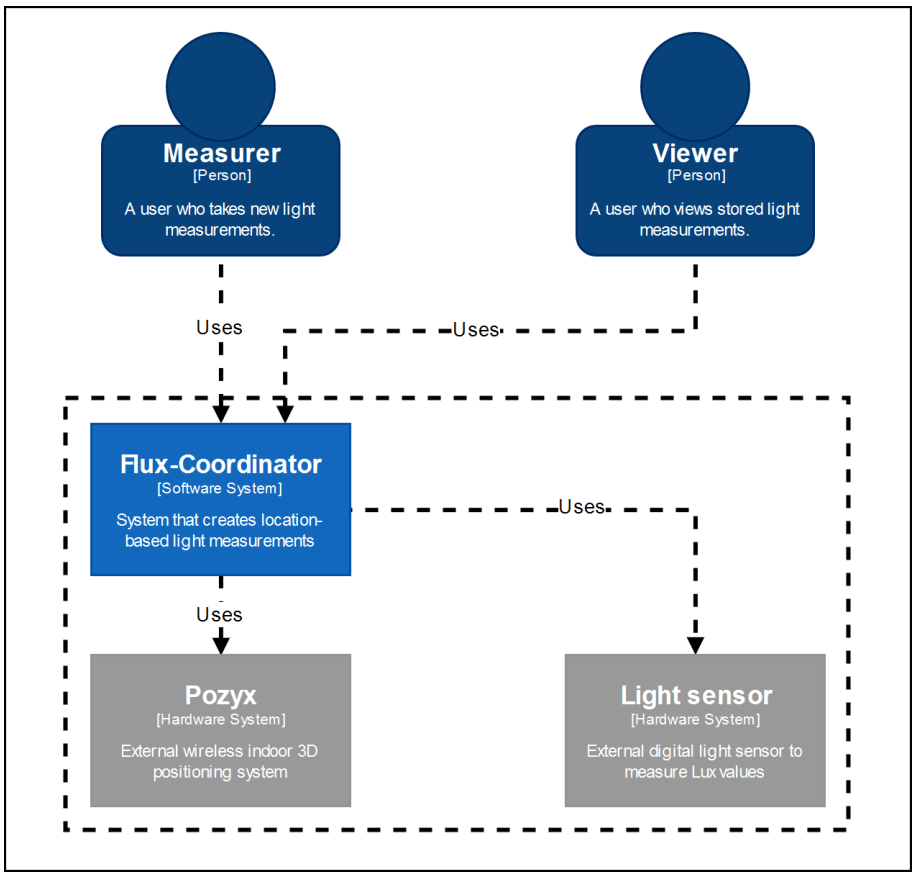


Abbildung 1 Systemkontextdiagramm des Flux-Coordinators

5.2 Rollen

Folgende Rollen wurden während der Anforderungsanalyse identifiziert:

- Betrachter: Greift auf die Anwendung zu, um bereits erstellte Aufzeichnungen anzuzeigen, zu importieren oder zu exportieren.
- Vermesser: Kalibriert die Sensoren, startet die Messungen und überwacht diese während der Durchführung. Der Vermesser ist ein Benutzer mit erweiterten Anforderungen.

5.3 Szenarios

Die folgenden Szenarios sollen helfen, eine High-Level-Perspektive über das System Flux-Coordinator und dessen Aufgaben zu erhalten. Das Entwickeln eines Szenarios garantiert, dass alle

[2] https://c4model.com/
[3] Der Präfix Flux ist eine Kombination aus Flux (lateinisch Fluss) und Lux (SI-Einheit der Beleuchtungsstärke) und soll den physikalischen Hintergrund der Arbeit symbolisieren. Der Name Flux-Coordinator ist zudem eine Anlehnung an den «Flux Capacitor» aus der Filmreihe «Back to the Future».

nötigen Interaktionsschritte einer Aufgabe aufgenommen und analysiert werden. Die einzelnen Schritte werden anschliessend als User Stories formuliert. Auf das modellieren von feingranularen Use Cases für die Szenarien wird verzichtet, da noch zu wenige Details über die zu verwendende Hardware oder über die zu erwartende Implementierung bekannt sind.

5.3.1 Messdurchführung

Die Messdurchführung ist sicherlich die Hauptaufgabe des Systems.

Zweck	Szenario, das die Durchführung einer Lichtmessung mit dem System Flux-Coordinator beschreibt.
Person	Mitarbeiter A, Bachelor of Science in Elektrotechnik

Ausrüstung	• Mobilgerät (Smartphone/Tablet/Laptop) mit unterstütztem Webbrowser und WLAN-Funktionalität
	• Flux-Coordinator Sensoreinheit, bestehend aus:
	  o Indoor Positionierungssystem «Pozyx», bestehend aus einem «Tag» und min. vier «Anchors»
	  o Lichtsensor
	  o Einplatinencomputer mit Akku

Szenario	• Mitarbeiter A startet die Sensoreinheit mit dem vorher installierten Flux-Coordinator System. • Das System startet automatisch und stellt einen WLAN-Hotspot zur Verfügung. • Mitarbeiter A verbindet sich mit dem WLAN und ruft per definierter URL im Webbrowser das System auf. • Mitarbeiter A erstellt ein neues Projekt mit Name und Beschreibung. • Mitarbeiter A erstellt zugehörigen Raum mit Name, Beschreibung, Gebäudeplan und Fläche. • Mitarbeiter A erstellt für den neuen Raum eine Messung mit Name, Beschreibung, Zielhöhe und Toleranz. • Mitarbeiter A verteilt die zur Verfügung stehenden «Anchors» des Positionierungssystems im Raum, gemäss den Richtlinien von Pozyx und vermisst diese anschliessend auf einen Millimeter genau, relativ zu einem vorher frei gewählten Nullpunkt. • Mitarbeiter A trägt alle verteilten «Anchors» mit ihrer Pozyx-ID und Position in die vorher erstellte Messung ein. • Mitarbeiter A definiert im User Interface den Skalierungsfaktor, sowie die Horizontal- und Vertikalverschiebung der «Anchors» auf dem hochgeladenen Gebäudeplan, sodass die Positionierung der Realität entspricht. • Mitarbeiter A kalibriert den Lichtsensor mit verschiedenen Lichtquellen und trägt die Messwerte im User Interface ein. • Mitarbeiter A startet die Messung über das User Interface. • Das System trägt den Vermesser und die Startzeit in die Messung ein. • Das System überträgt alle definierten Parameter der Messung an die Sensoreinheit und startet die Sensoren. • Das System zeichnet eine Heatmap mit allen gemessenen Lichtwerten, die in der Toleranz der festgelegten Zielhöhe liegen und aktualisiert die Visualisierung stetig.

	• Mitarbeiter A bewegt sich mit der Sensoreinheit im Raum umher, bis die Messabdeckung gemäss der Heatmap als ausreichend bewertet wird. • Mitarbeiter A beendet die Messung über das User Interface. • Das System trägt die Endzeit in die Messung ein. • Mitarbeiter A kann die generierte Heatmap der Messung betrachten und bei Bedarf als Datei exportieren.

5.3.2 Erstellte Messung betrachten

Szenario, um eine bereits erstellte Messung über das in der Cloud bereitgestellte Flux-Coordinator System zu betrachten.

5.3.3 Import / Export

5.4 Funktionale Anforderungen

Die Funktionen, die die Anwendung unterstützen soll:

#	Anforderung	Beschreibung	Rolle
1	Messung visualisieren	Als Benutzer möchte ich alle Messwerte einer Messung als Visualisierung anzeigen können, um deren Zusammenhänge zu verstehen.	Betrachter
2	Messwerte filtern	Als Benutzer möchte ich in der Visualisierung einen Filter setzen können, um die Messwerte nach ihren Abständen zu gruppieren.	Betrachter
3	Messung speichern	Als Benutzer möchte ich eine Messung speichern können, um die gesammelten Daten zu einem späteren Zeitpunkt zu betrachten.	Betrachter
4	Messung starten	Als Vermesser möchte ich eine Messung über die Benutzerschnittstelle starten können, damit die Messwerte aufgezeichnet werden.	Vermesser
5	Messung abschliessen	Als Vermesser möchte ich eine laufende Messung über die Benutzerschnittstelle abschliessen können, um die Aufzeichnung weiterer Messwerte zu beenden.	Vermesser
6	Messung fortfahren	Als Vermesser möchte ich eine bereits abgeschlossene Messung über die Benutzerschnittstelle fortfahren können, um sie durch weitere Messwerte zu erweitern.	Vermesser
7	Messung exportieren	Als Benutzer möchte ich im System vorhandene Messungen in einem textbasierten Format exportieren können, um sie extern zu sichern.	Betrachter
8	Messung importieren	Als Benutzer möchte ich eine nicht mehr im System vorhandene Messung importieren können, um sie wieder anzuzeigen.	Betrachter
9	Positionierungssystem konfigurieren	Als Vermesser möchte ich das Positionierungssystem über die Benutzerschnittstelle konfigurieren können, um die Messung schneller aufzusetzen.	Vermesser
10	Lichtsensor kalibrieren	Als Vermesser möchte ich den Lichtsensor über die Benutzerschnittstelle kalibrieren können, um ungenaue Messungen zu vermeiden und Korrekturen vorzunehmen.	Vermesser
11	Anmeldung durchführen	Als Benutzer möchte ich mich anmelden können, um auf die Daten meiner Installationen zugreifen zu können.	Betrachter
12	Abmeldung durchführen	Als Benutzer möchte ich mich nach der Anmeldung abmelden können, um die Daten meiner Installationen für unbefugte unzugänglich zu machen.	Betrachter
13	Instrumente testen	Als Vermesser möchte ich die Konfiguration und Kalibrierung der Sensoren vor Beginn der Messung testen können, um fehlerhafte Messungen zu vermeiden.	Vermesser

Tabelle 1 Funktionale Anforderungen des Flux-Coordinators

Das Formulieren der CRUD-Methoden (Create, Read, Update, Delete) wurde der Einfachheit halber weggelassen. Diese werden bei Bedarf implementiert.

Die Priorisierung der Anforderungen entspricht der Reihenfolge in der Tabelle. Es gibt keine optionalen Anforderungen, sondern lediglich die Priorisierung und eine zeitlich begrenzte Entwicklungszeit.

5.5 Nicht funktionale Anforderungen

Die folgenden nichtfunktionalen Anforderungen sind nach ISO 9126 [1] gruppiert.

#	Anforderung	Kategorie
1	Sensoren lassen sich austauschen, ohne dafür die Serverkomponente anpassen zu müssen.	Maintainability
2	Ein Benutzer muss auf das User Interface zugreifen können, ohne zusätzliche Software auf seinem Client Gerät installieren zu müssen.	Usability
3	Die Messwerte müssen bei einer Round-Trip-Time von unter 25 ms in mindestens 90% der Anfragen in weniger als 1 Sekunde angezeigt werden.	Efficiency
4	Die Server-Komponente muss auf einer von den Sensoren getrennten Plattform ausgeführt werden können.	Portability
5	Das User Interface soll auf den gängigen Plattformen (PC, Laptop, Tablet) verwendet werden können.	Portability
6	Die Funktionalität des Systems muss vor nicht-authentifizierten Zugriffen geschützt sein.	Security
7	Die aufgezeichneten Messwerte müssen unverändert persistiert werden. Eventuelle Faktoren oder Filter müssen zusätzlich zu den Rohdaten abgelegt werden.	Functionality

Tabelle 2 Nichtfunktionale Anforderungen des Flux-Coordinators

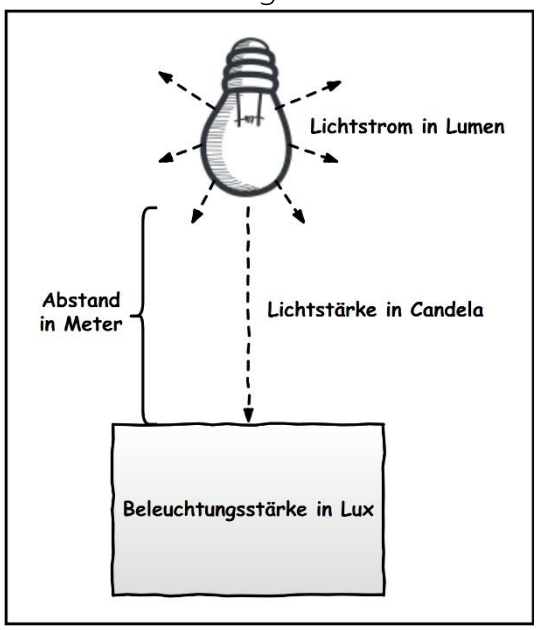
5.6 Mockups

Zur Visualisierung der Anforderungen und zur einfacheren Kommunikation mit dem Kunden und Betreuer wurden Mockups zu allen Ansichten erstellt. Nachfolgend sind zur Übersicht die wichtigsten Ansichten aufgeführt.

Mockups einfügen

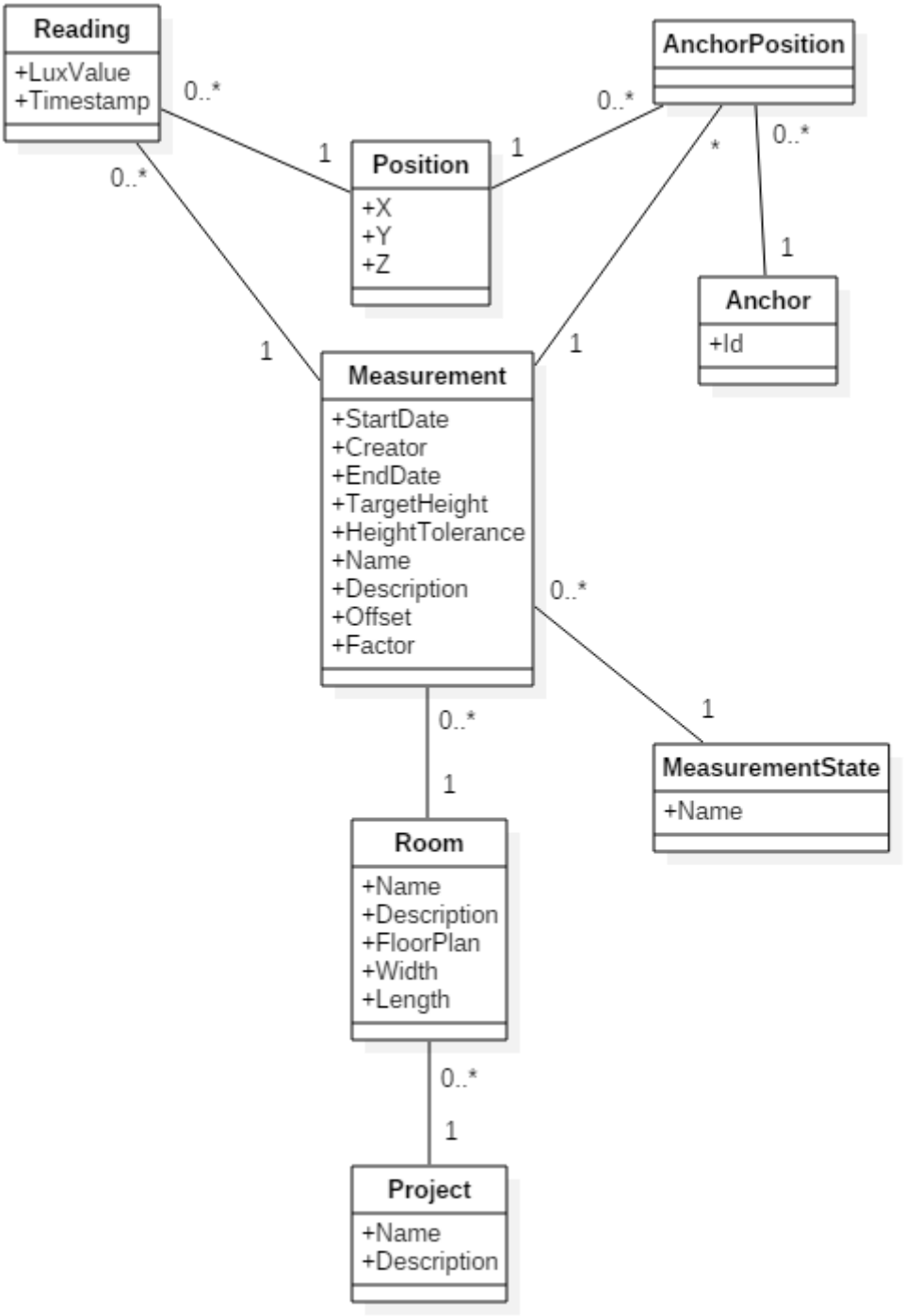
6 Domainanalyse

6.1 Lichtmessung



6.2 Domain-Modell

Das nachfolgende Domain-Modell zeigt die wesentlichen konzeptionellen Klassen und ihre Zusammenhänge. Es soll einen Überblick über die Problemdomäne schaffen.



Nachfolgend sind einige Klassen und Beziehungen genauer beschrieben:

Auf den ersten Blick ist das Attribut *TargetHeight* der Klasse *Measurement* identisch mit dem *Z*-Attribut der Klasse *Position*. Die *TargetHeight* muss jedoch zu Beginn einer Messung definiert werden und beschreibt somit die Soll-Höhe der auszuführenden Messung. Der *Z*-Wert beschreibt hingegen den effektiv von der Positionierungssoftware gemessenen Wert.

Das Attribut *HeightTolerance* definiert die Toleranz für Abweichungen von der gewünschten Messhöhe (*TargetHeight*). Dieser Wert muss bei unterschiedlichen Traversierungs-Hilfsmitteln gegebenenfalls angepasst werden, da nicht immer dieselbe Präzision erreicht werden kann. Eine zu kleine

Toleranz bedeutet das Wegwerfen vieler Messwerte und würde die Kartierung des Raumes verlangsamen. Eine zu grosse Toleranz ergibt Messwerte, die sich in der aufgenommenen Höhe stark unterscheiden. Dies könnte die Aussagekraft der Messung verschlechtern.

Die Beziehung zwischen *Measurement* und *AnchorPosition* definiert für eine Messung beliebig viele Antennen des Positionierungssystems. Laut dem Hersteller des Positionierungssystems Pozyx sind mit entsprechender Programmierung unbegrenzt viele Antennen möglich[4].

6.2.1 Ubiquitous Language

Nachfolgend sind die wichtigsten Begriffe der Problemdomäne für den Kontext dieser Arbeit definiert. Dies ist ein bewährtes Vorgehen im Domain-Driven Design (DDD) [2].

Besonders hervorzuheben sind die Begriffe *Measurement* und *Reading*, da sie schnell verwechselt werden und doch den Kern dieser Arbeit bilden.

Begriff	**Erklärung**
Project	Logische Gruppierung von Messungen (Measurement) und deren Räumen (Room) die zu einem gemeinsamen Auftrag gehören
Room	Zu vermessender Raum oder Bereich eines Raumes innerhalb eines Gebäudes
Measurement	Einzelne Messdurchführung innerhalb eines Raumes (Room) mit beliebig vielen Messwerten (Readings)
Reading	Einzelner Messwert als Kombination aus Helligkeitswert in Lux und relativer Position innerhalb des Raumes als kartesische Koordinaten
Anchor	Referenzpunkt des Positionierungssystems innerhalb des Raumes
Kartierung	Das Vermessen eines Raumes heisst Kartierung.
Floor Plan	Der Grundriss des zu vermessenden Raumes.

Tabelle 3 Definition der Ubiquitous Language

6.3 Human Error (Menschliche Fehler)

https://en.wikipedia.org/wiki/Human_error

[4] https://www.pozyx.io/Documentation/where_to_place_the_anchors (Kommentar von Pozyx Labs)

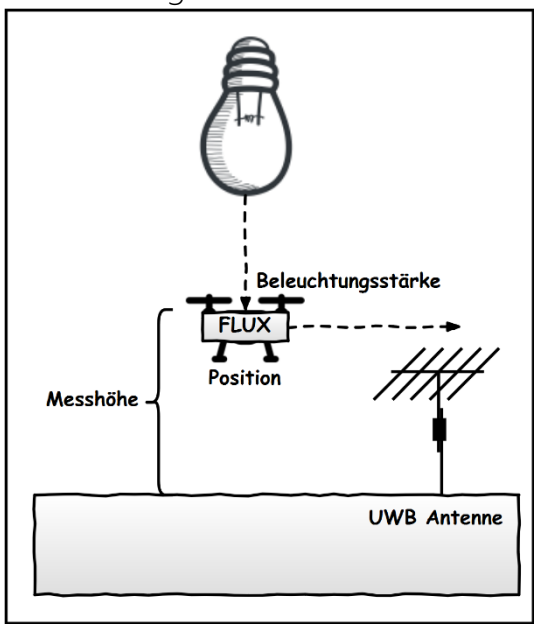
6.4 Positionserkennung

6.5 Integration der Sensordaten

6.6 User Experience

6.7 MCMC-Problem?

6.8 Lösungsentwurf



7 Architektur

Die meisten der nachfolgenden Architekturdiagramme sind dem C4 Model von Simon Brown [3] nachempfunden. Dieses soll es Softwareentwicklern vereinfachen, die Funktionsweise eines Softwaresystems zu beschreiben und minimiert die Lücke zwischen der Software Architektur und dem Programmcode. Das Systemkontextdiagramm aus Kapitel *5.1 Systemkontext* entspricht dabei dem ersten «C» (Context).

Die Anforderungen an die Architektur ergeben sich zum einen aus der Aufgabenstellung und zum anderen aus den Projekteinschränkungen aus Kapitel *4.5 Einschränkungen* und aus den nicht funktionalen Anforderungen aus Kapitel *5.5 Nicht funktionale Anforderungen*.

7.1 Ziele

Mit folgender Architektur sollen von zwei unabhängigen Sensoren aufgezeichnete Licht- und Positionswerte kombiniert und an ein Server-System übertragen werden. Anschliessend sollen die Messwerte für den Benutzer visualisiert und zum Export angeboten werden.

Die Kommunikation mit dem Server soll sowohl von den Sensoren (Messdurchführung) als auch von der Benutzerschnittstelle (Betrachtung der Visualisierung) initiiert werden können. Zudem sollen diese beiden Kommunikationsabläufe komplett unabhängig voneinander funktionieren.

7.2 Einschränkungen

Das Indoor-Lokalisierungssystem zur Bestimmung der Positionswerte wurde bereits im Vorfeld dieser Arbeit zusammen mit dem betreuenden Dozenten evaluiert und muss in die Architektur integriert werden. Als Lösung wurde folgende Technologie bestimmt: *Pozyx UWB - Indoor Positionierung System*[5].

Der zu verwendende Lichtsensor ist ebenfalls Vorgegeben. Vom Auftraggeber wurde folgender Sensor bestimmt: *TCS3430 Tristimulus Color Sensor*[6].

[5] Bestellt wurde das «Pozyx Ready to Localize» Set mit 4 Anchors: https://www.pozyx.io/store/detail/2
[6] https://ams.com/TCS3430

7.3 Container Aufteilung

In Abbildung 2 wird das System Flux-Coordinator in einzelne Container zerlegt. Container sind das zweite «C» des C4 Model und beischreiben logische Behältnisse, die Code oder Daten enthalten.

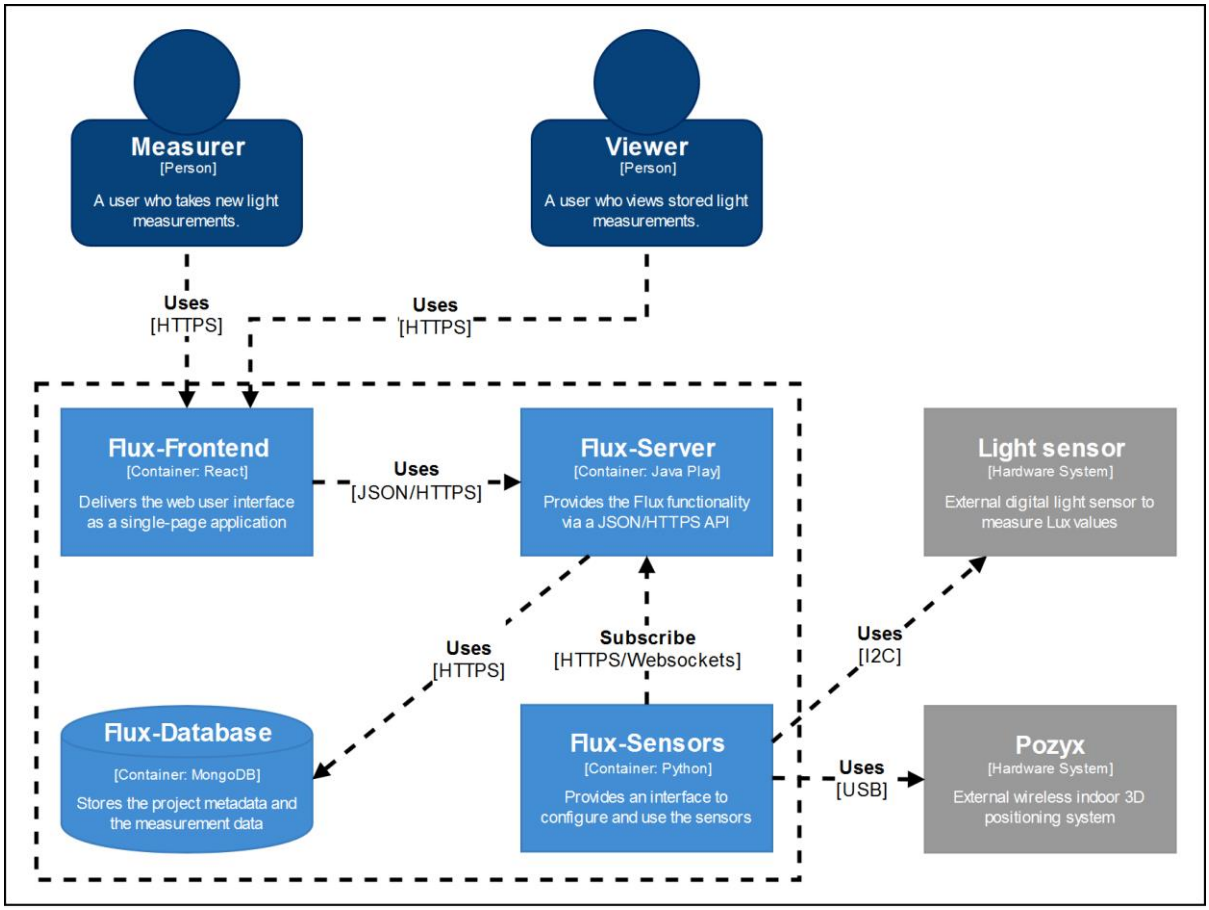


Abbildung 2 Container Diagramm des Flux-Coordinators

Die Architektur soll so aufgebaut werden, dass das System auch in verschiedenen Situationen optimal eingesetzt werden kann. Es sollte dafür jedoch nicht komplexer werden als nötig, damit die Bereitstellung, Administration und Bedienung des Systems nicht darunter leiden.

Die folgenden drei Situationen sind in den Anforderungen definiert oder aus Gesprächen mit dem Auftraggeber und in eigenen Versuchen entstanden:

- Eine Messdurchführung beim Kunden muss komplett offline durchgeführt werden können und darf nicht auf Services aus dem Internet angewiesen sein. Dabei Steuert der Vermesser die Sensoreinheit und sieht die gesammelten Messwerte in Echtzeit in einer Visualisierung.
- Bereits abgeschlossene Messungen können auch ohne eingeschaltete Sensoreinheit und ohne Installation von zusätzlicher Software unterwegs auf einem Mobilgerät betrachtet werden.
- Das System kann auch von mehreren Benutzern gleichzeitig auf Büro-PCs eingesetzt werden, um neuen Messungen zu erstellen oder bereits abgeschlossene Messungen zu betrachten. Die gespeicherten Daten können bei Bedarf von jedem PC im selben Netzwerk exportiert werden.

Um diesen Anforderungen gerecht zu werden, wird ein verteiltes Softwaresystem aus verschiedenen, unabhängigen Containern entworfen. Nachfolgend sind die vier Container und ihre Aufgaben genauer beschrieben. Dem Namen wird jeweils das Präfix «Flux» vorangestellt, um zu verdeutlichen, dass damit ein Container innerhalb des Systems Flux-Coordinator gemeint ist.

7.3.1 Flux-Frontend

Die Hauptaufgabe des Flux-Frontend ist das Visualisieren der Messwerte, um dem Benutzer eine optimale Auswertung zu erlauben.

Daraus ergeben sich folgende Anforderungen an den Container:

- Bereitstellung einer geeigneten Visualisierung der Messwerte
- Automatische Aktualisierung bei neuen Messwerten
- Bedienung der Applikation durch den Benutzer
    - Login
    - Starten und Stoppen von Messungen
    - Import und Export von Messungen
    - CRUD Funktionalitäten
    - Filtern von Messwerten und Feineinstellung der Visualisierung

Als Technologie wird ein modernes Frontend-Framework, wie beispielsweise Angular[7] oder React[8] in Betracht gezogen. Dadurch könnte von Lösungen der Frameworks profitiert werden und der Fokus bei der Entwicklung auf die Kernaufgabe des User Interfaces, der Visualisierung der Messwerte, gelegt werden. Bei der Evaluation des Frameworks sollte daher speziell auf die Möglichkeiten der Datenvisualisierung geachtet werden.

7.3.2 Flux-Server

Der Flux-Server stellt den Mittelpunkt des Systems dar, an dem die Daten zusammenkommen und auch wieder verteilt werden. Seine Hauptaufgabe ist somit die Vermittlung und Bereitstellung der Messdaten für die anderen Container.

Daraus ergeben sich folgende Anforderungen an den Container:

- Bietet eine Schnittstelle für Flux-Frontend
    - zur Abfrage von Messwerten
    - zur Benachrichtigung bei neuen Messwerten
    - zur Bedienung der Applikation (siehe Anforderungen von Flux-Frontend)
- Bietet eine Schnittstelle für Flux-Sensors
    - zum Starten und Stoppen der Sensoren
    - zur Übergabe von neuen Messwerten
- Sendet die zu persistierenden Daten an die Datenbank
- Gibt in der Datenbank gespeicherte Daten zurück

Für den Flux-Server ist als Technologie ein modernes Backend Web-Framework, wie Spring, Play Framework oder Express angedacht. Der Backend Server hat in diesem Projekt nebst der Vermitt-

[7] https://angular.io/
[8] https://reactjs.org/

lung und Bereitstellung der Daten keine speziellen Aufgaben oder eigene Business Logik zu implementieren. Es sollte also ein Framework evaluiert werden, dass im Umgang mit Daten stark ist und die Standardfunktionalität eines Web Backends bereits erfüllt.

7.3.3 Flux-Sensors

Dies ist die Sensoreinheit, die die beiden Sensoren für die Licht- und Positionsbestimmung verbindet. Seine Hauptaufgabe ist das Auslesen und Synchronisieren der Sensordaten, damit zu jeder Lichtmessung die jeweilige Position im Raum und die aktuelle Zeit hinzugefügt wird.

Daraus ergeben sich folgende Anforderungen an den Container:

- Registriert sich beim Flux-Server und wartet darauf, dass eine Messung gestartet wird
- Kommuniziert mit den beiden Sensoren und fragt deren Messwerte ab
- Synchronisiert die Messwerte und fügt die aktuelle Zeit hinzu
- Sendet die kombinierten Messwerte der Sensoren an den Flux-Server
- Beendet den Messvorgang, sobald der Flux-Server dies signalisiert

7.3.4 Flux-Database

Die Datenbank ist sicherlich der simpelste Container. Seine Hauptaufgabe ist das Persisitieren der Messdaten.

Daraus ergeben sich folgende Anforderungen an den Container:

- Persistiert Messwerte und dazugehörige Metadaten, wie Projekt-, Raum- und Messdetails
- Stellt die persistierten Daten auf Anfrage des Flux-Servers zur Verfügung

7.4 Kommunikations-Architektur

- http und Websockets
- Sensoren: I2C vs. USB / Push vs. Poll
- Swagger

7.5 Deployment

Bei der bisherigen Planung des Systems wurde stets auf eine tiefe Kopplung der einzelnen Container wertgelegt, um den Einschränkungen und Anforderungen des Projekts gerecht zu werden. Nachfolgend sind zwei Deployment-Szenarien abgebildet, in denen der Flux-Coordinator am ehesten zum Einsatz kommen wird. Durch die Unabhängigkeit der einzelnen Container sind jedoch diverse Variationen möglich. So lässt sich das System in Zukunft optimal erweitern.

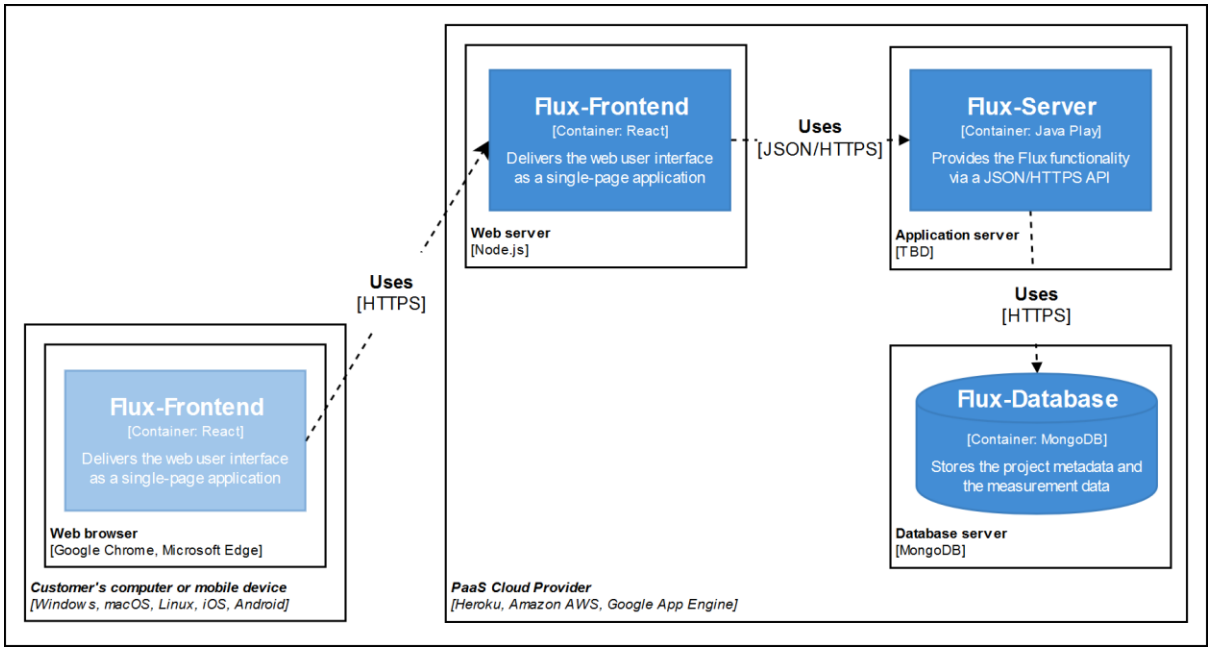


Abbildung 3 Deployment Diagramm mit Paas Cloud Provider

Die erste Variante in Abbildung 3 zeigt eine mögliche Anwendung für den Einsatz im Büro. Die einzelnen Container werden auf einem PaaS Cloud Provider ausgeführt und die Sensorkomponente wurde ganz weggelassen. So können bereits erstellte Messungen betrachtet und exportiert werden.

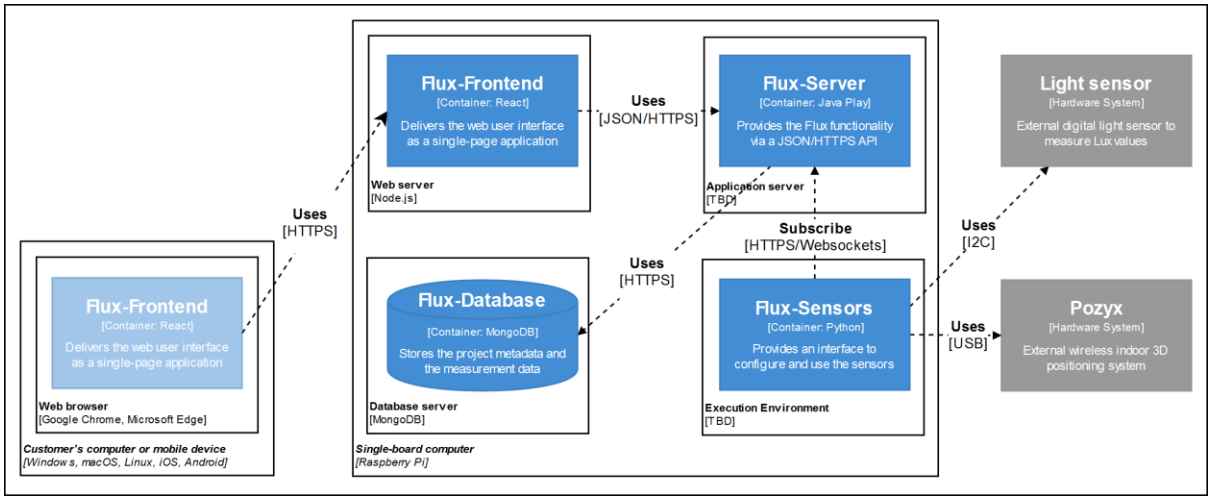


Abbildung 4 Deployment Diagramm mit Single-board Computer

Die zweite Variante in Abbildung 4 zeigt eine mögliche Anwendung für die Durchführung von Messungen bei einem Kunden. Die einzelnen Container werden auf einem Single-board Computer, wie beispielsweise dem Raspberry Pi ausgeführt, wo auch die Sensoren per USB und I2C angeschlossen sind. So können auch ohne Verbindung zum Internet neue Messungen erstellt und bereits abgelegte Messungen betrachtet werden.

8 Realisierung

8.1 Flux-Frontend

Das Frontend ist die Schnittstelle für den Benutzer des Flux Coordinator Systems und ist für das Rendern der von der API gelieferten Daten zuständig. Dafür wurde die Technologie ReactJS verwendet. Durch das Lokale Rendern auf dem Client, lässt sich die Reaktionszeit des UIs vermindern, was zu einer besseren Nutzererfahrung führt.

8.1.1 Technologieevaluation

Um eine möglichst gute User Experience zu gewährleisten, wurde schon früh der Einsatz eines modernen Frontend-Frameworks entschieden. Es kamen drei Technologien in Frage: ReactJS, Angular und VueJS. Ausschlaggebend für den Entscheid war die Qualität der Dokumentation, der Bekanntheitsgrad und die voraussichtlich benötigte Einarbeitungszeit.

ReactJS

Im Gegensatz zu den anderen Kandidaten (speziell Angular), positioniert sich React als eine Library und nicht als ein Framework. Das Hauptargument dafür ist, dass React sich hauptsächlich mit dem UI befasst und dem Entwickler das Architekturdesign weitgehend überlässt. Dies ist zugleich Vor- und auch Nachteil, da die Erfahrung des Entwicklungsteams ausschlaggebend ist, um eine funktionierende und längerfristig skalierende Architektur zu entwerfen. Die Lernkurve ist durch die wenigen Einschränkungen des Frameworks viel kleiner – genauso, wie die Einarbeitungszeit. Die Oberfläche der Schnittstellen von React ist überschaubar und relativ einfach zu verstehen. Eine weitere wichtige Eigenschaft ist die nicht vorhandene Trennung von UI-Komponenten und Business-Logik. Das gesetzte Ziel des Frameworks ist eine hohe Kopplung von UI und der dazugehörigen (spezialisierten) Logik.

React kann mit Vanilla-JavaScript verwendet werden, eine Verwendung von einer Typisierungstechnologie wird jedoch empfohlen. Facebook hat als Entwickler von React «Flow» als Alternative zu TypeScript (Microsoft) entwickelt. Es ist zu erwarten, dass die Flow Typisierungssprache stärker unterstützt wird als TypeScript, wobei die Kritik an Flow (zum Teil berechtigt) stark ist. Viele Libraries, wie die beliebte Material-UI Library haben Flow den Rücken gekehrt und setzen TypeScript ein [4]. Eine breite Unterstützung von Flow ist nicht gegeben, da die Community stark gespalten ist.

React bietet eine Starthilfe mit dem Namen «create-react-app» für neue Projekte. Es handelt sich hierbei um eine Depdendency, die viele Befehle bereitstellt und die komplizierten Build-Einstellungen – speziell von Webpack – einer React Anwendung abstrahiert. Es werden dabei viele best-practice Einstellungen gesetzt, die für die meisten Projekte funktionieren. Leider lässt sich durch die Abstraktion das Tooling nicht mehr auf eine einfache Weise auf spezifische Anforderungen anpassen. Auch können die Dependencies nicht mehr eigenständig aktualisiert werden, da diese von create-react-app verwaltet werden. Falls sich eine der gewählten Einstellungen aus dem create-react-app Projekt als inkompatibel mit den Anforderungen des Projektes herausstellt, muss oft ein «Eject» durchgeführt werden. Ein Eject bedeutet, dass sich das create-react-app Tool aus dem Projekt löscht und dabei die komplette Konfiguration offenlegt. Das Entwicklerteam muss sich ab diesem Zeitpunkt selber um die Konfiguration der Anwendung bemühen, was den Komplexitätsgrad der Anwendung enorm erhöht. Ein Eject ist nicht umkehrbar und sollte deshalb so weit, wie möglich vermieden werden. Eine Anleitung mit Workarounds zu häufig auftretenden Problemen mit der create-react-app ist auf dem GitHub Repository von create-react-app verfügbar [5].

Angular

Das zweitgrösste Frontend-Framework nach ReactJS ist Angular. Das Framework wird wie die anderen Kandidaten ebenfalls OpenSource entwickelt. Geführt wird das Projekt unter Google's Aufsicht. Angular sollte nicht mit dem Vorgänger AngularJS verwechselt werden. Ausser dem Namen haben diese zwei Frameworks nämlich nicht viel gemeinsam. Verglichen mit ReactJS, bietet Angular ein komplettes Framework mit Workflow und einer etablierten Architektur. Der einzelne Entwickler ist hier viel stärker an die Vorgaben des Frameworks gebunden und hat somit weniger Spielraum. Dies ist in diesem Fall nur selten ein Nachteil, denn das Framework bietet eine solide und mittlerweile geprüfte Architektur, die für die allermeisten Fälle genügt.

Angular wurde komplett in TypeScript entwickelt [6] und empfiehlt auch die Verwendung von TypeScript für Angular Anwendungen [7]. Diese Typisierungstechnologie scheint sich dabei im Gegensatz zu Flow in React bei den meisten Entwicklern durchgesetzt zu haben. Im Zweifelsfall ist auch eine Verwendung von JavaScript möglich, doch alle Beispiele in der Dokumentation von Angular sind in TypeScript geschrieben.

Die Zielanwendungen von Angular sind primär grosse, komplexe Anwendungen und das Framework bietet dafür eine solide Ausgangslage. Die Komplexität des Frameworks hat eine steile Lernkurve zur Folge. Bevor angefangen werden kann, produktiven Code zu schreiben, sollte ein Entwickler sich ausgiebig mit den Konzepten von Angular auseinandersetzen. Die Architektur der entstehenden Anwendung ist dafür oft sauberer und durchdachter, als ohne ein Framework.

Durch die AngularCLI wird ein mächtiges Tool geliefert, um die Entwicklung von Angular zu vereinfachen. Das Erstellen von Projekten und Komponenten können mit den Befehlen `ng new` oder `ng generate` automatisiert werden.

VueJS

Dieses Framework stellt sich als dritte Alternative zu den bekannteren zwei Frontend Frameworks ReactJS und Angular. Es ist sehr interessant, dass Vue viele Ähnlichkeiten mit React und mit Angular besitzt [8]. Im Gegensatz zu React und Angular, wird Vue jedoch nicht von grossen Unternehmen unterstützt. Der Erfinder Evan You wurde während seiner Zeit bei Google durch AngularJS inspiriert und entschied sich, die für ihn besten Eigenschaften von AngularJS in einem neuen Framework zu implementieren [9].

Das Ziel des Frameworks ist, schlank und einfach zu lernen zu sein. Die Syntax sieht teilweise der Syntax von AngularJS sehr ähnlich. Die Performance ist mit der von React vergleichbar. Eine Typisierungstechnologie ist nicht inbegriffen. Die Entwickler sollen sich für eine Typisierungstechnologie entscheiden müssen.

Ähnlich wie Angular bietet Vue eine CLI für das Erstellen von neuen Projekten. Die Einstellungen der eingesetzten Werkzeuge, wie Webpack sind in Vue komplett einstellbar. Ein Ejecten, wie in React ist nicht nötig [10]. Als Alternative für die CLI, gibt es eine graphische Oberfläche für das Erstellen von neuen Projekten. Dieses extensive Tooling zeugt von einer guten Organisation der VueJS Community und vom Potential des Frameworks.

Fazit

TODO

Hinweis: Der grosse Unterschied zwischen klassischen Server-Side Rendering und Frontend-Rendering einer Webseite liegt darin, dass die Webseite bei Frontend-Rendering besser auf Benutzereingaben reagieren kann. Eine Webseite, die im Frontend gerendered wird (auch Single-Page-Application) wird nur einmal geladen. Danach geschieht die gesamte Navigation der Anwendung lokal auf dem Client des Benutzers. Wenn die Anwendung Daten aus einer Schnittstelle lädt, kann diese dem Benutzer angemessenen Feedback geben, in Form eines Ladebalkens. Bei klassischen Webseiten, würde hier die gesamte Seite neu geladen.

8.1.2 Containers und Components

Um den Datenfluss von React Applikationen besser zu ordnen, wird oft das Container-Component Pattern verwendet [11]. Containers stellen die Daten für die Components bereit. Sie können die Daten aus einer beliebigen Quelle, wie einem REST-Service oder einer Datei holen. Sobald die Daten vorhanden – und möglicherweise bearbeitet – sind, werden sie der untergeordneten Component weitergegeben.

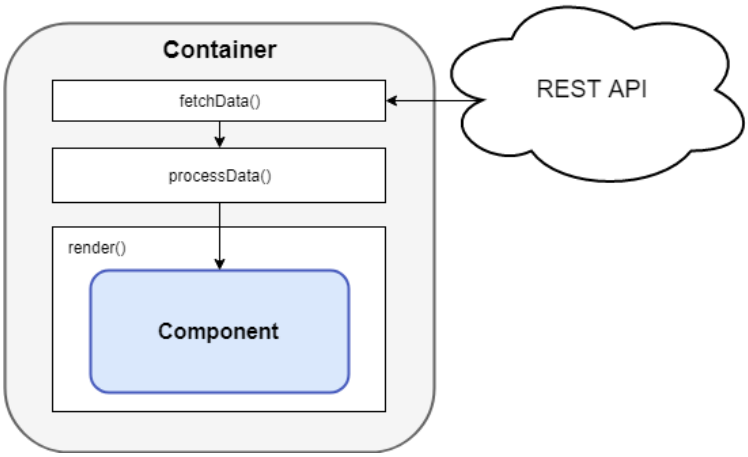


Abbildung 5 Container-Component Pattern Veranschaulichung

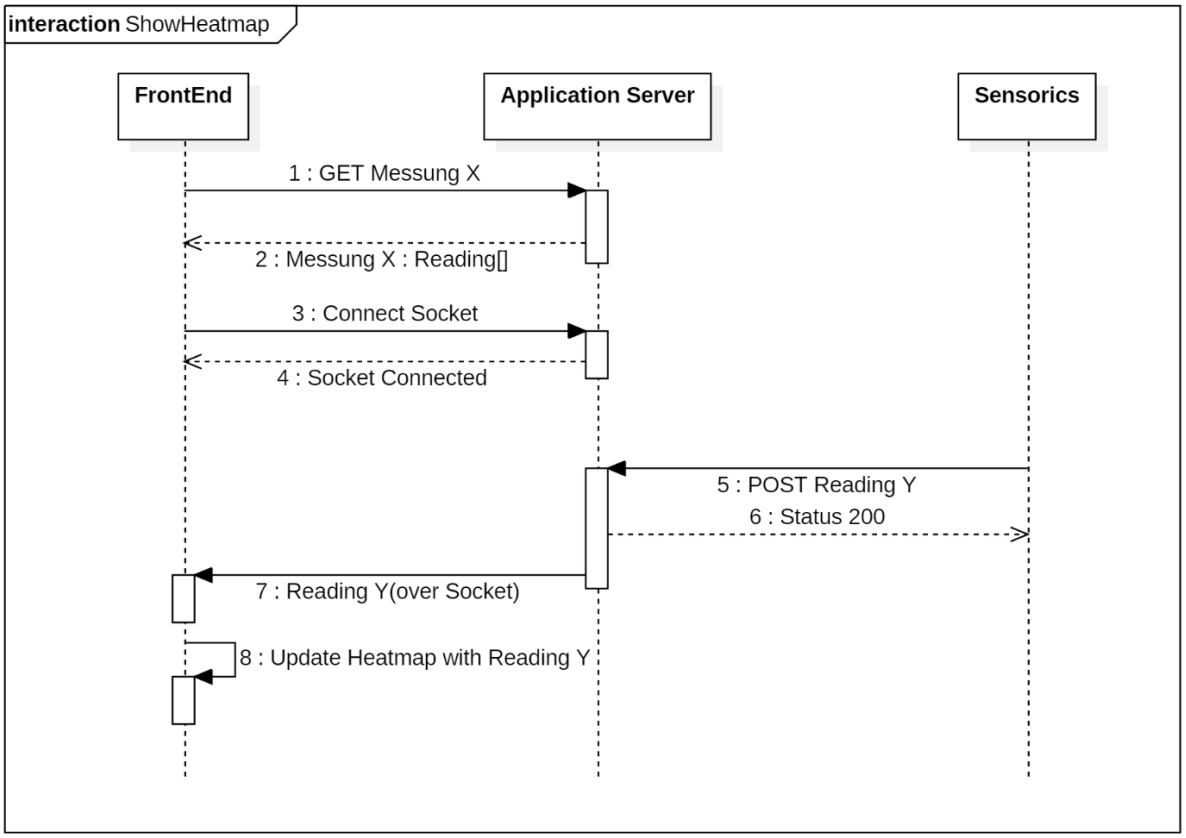
Im Objektorientierten Sinn verfolgt diese Abstraktion das gleiche Ziel, wie ein Interface. Die Component soll unabhängig davon funktionieren, woher die Daten geholt werden. Meistens sind Components stateless, weshalb sie als einfache Funktion erstellt werden können. Container hingegen speichern die geholten Daten in einem State. Aus diesem Grund müssen Container als Klasse definiert werden. Durch dieses Pattern werden zwei Ziele erreicht: Vereinfachung von Testen durch die Trennung von UI und Business Logik und verhindern von zu grossen und komplizierten Komponenten, was die Weiterentwicklung und die Wartung der Software vereinfacht.

8.1.3 Heaptmap Library

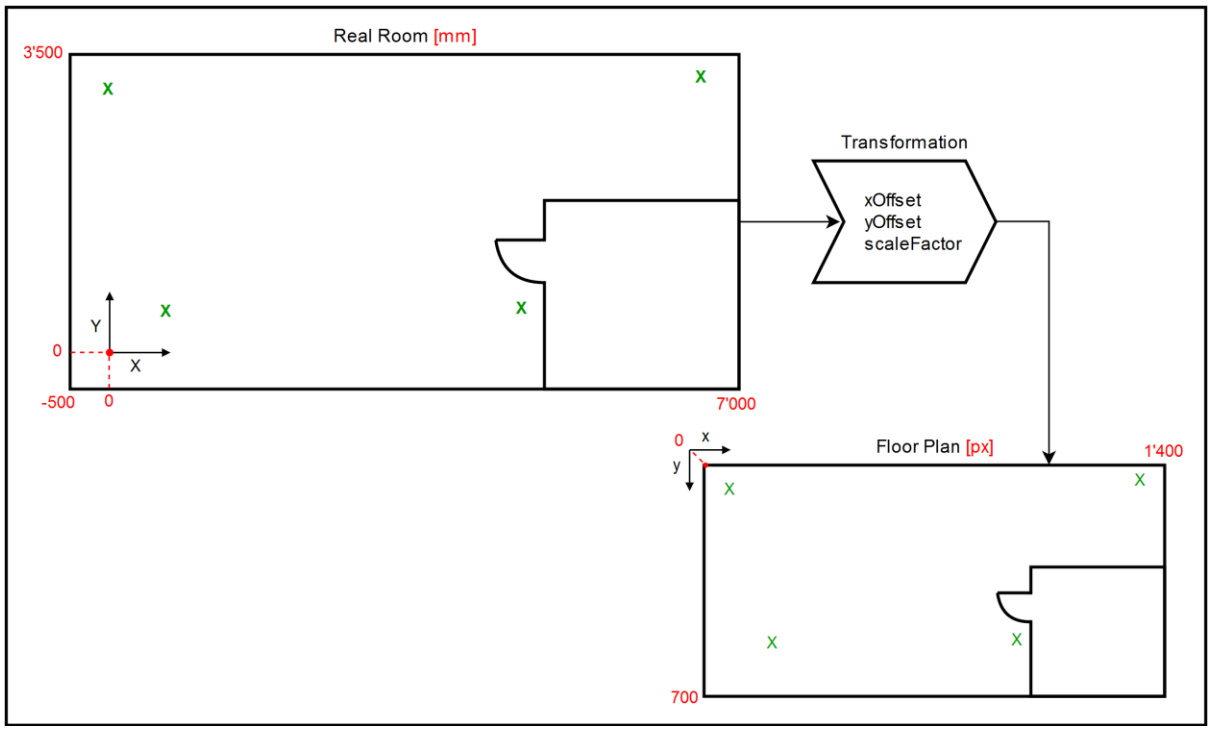
Library wurde geforkt und jetzt über NPM bereitgestellt.

Anzeigen der Heatmap

Die Heatmap lädt am Anfang die schon auf der Datenbank verfügbaren Messungen von der Schnittstelle. Anschliessend öffnet der Client eine Socketverbindung zum Application Server. Diese Socketverbindung wird gebraucht, damit die neuen Messungen der Sensoren direkt und fast in Echtzeit auf das Frontend übertragen werden. Der Socket wird wieder geschlossen, sobald der Benutzer die Kartenansicht verlässt.



Transformation der Koordinaten



8.2 Flux-Server

Als Knotenpunkt für alle verteilten Komponenten dient eine Applikationsschnittstelle (API). Diese soll auf einem Application Server bereitgestellt werden und auf dem REST Prinzip aufbauen, sodass die Kommunikation zwischen den Komponenten und der Schnittstelle über HTTP stattfindet. Als Datenformat soll JSON dienen, da sich dieses Format bei ähnlichen Schnittstellen als geeignet erwiesen hat und sich in der Technologiewelt durchzusetzen scheint. Das Datenformat XML wäre zwar prinzipiell möglich, würde jedoch keinen Vorteil bringen und wesentlich mehr Datenverkehr erzeugen. Weitere Vorteile sind die einfache Objektserialisierung von JSON in die vom Flux-Frontend verwendeten Programmiersprache JavaScript.

8.2.1 Technologieevaluation

Für den Flux-Server kamen mehrere Technologien in Frage. Die Kandidaten müssen Multi-Plattform fähig sein, gute Möglichkeiten bieten Hilfe zu holen sowie auch einigermassen etabliert sein. Natürlich spielen die bereits gesammelten Erfahrungen der Teammitglieder ebenfalls eine grosse Rolle bei der Auswahl der Technologie.

In den folgenden Abschnitten werden die Kandidaten vorgestellt, die evaluiert worden sind.

Play Framework

Dieses relativ moderne Framework wurde in der Programmiersprache Scala entwickelt und unterstützt die Entwicklung von Web Applikationen sowohl mit Java, wie auch mit Scala zu. Ein wichtiges Ziel des Frameworks ist es, die Arbeit der Entwickler schneller und einfacher zu gestalten. Dies ist auch erfolgreich gelungen. Eine Anwendung lässt sich mit sehr wenig Aufwand erstellen und die Entwicklung der eigentlichen Software kann sofort anfangen. Durch den «Convention over Configuration» Ansatz, welcher das Entwicklerteam des Playframeworks fährt, muss der Entwickler wenig einstellen und kann sich auf die Entwicklung der Software konzentrieren.

Da das Framework in Scala entwickelt wurde, wird das Scala Build Tool (SBT) verwendet, um den Buildprozess zu automatisieren. SBT ist hochautomatisiert und der Entwickler muss nach dem Starten des Entwicklungsmodus mit dem Befehl `sbt run` (oder über IntelliJ IDEA's Run Button) keine Befehle mehr eingeben. Bei Änderungen des Sourcecodes, wird die Anwendung automatisch kompiliert und ein Browser Refresh durchgeführt.

Nicht zuletzt, weil Play Framework von Lightbend – die Firma der Erfinder von Scala und Akka – entwickelt wird, setzt das Framework im Backend auf die Technologie Akka HTTP und Akka Streams und ist daher komplett asynchron aufgebaut. Da das System viele I/O handeln muss, ist dies ein wichtiger Pluspunkt in der Auswahl der Technologie für Flux-Server.

Spring Boot

Spring ist wohl das traditionellste Application Framework für die Java Plattform. Es ist bereits 15 Jahre alt und ist die reifste Technologie, die für dieses Projekt evaluiert wurde. Die Spring Boot Variante bietet einen «Convention over Configuration» Ansatz, welcher den normalerweise beträchtlichen Konfigurationsaufwand einer Spring Anwendung enorm vermindert. Die Idee ist – ähnlich wie beim Play Framework –, dass der Entwickler sich auf die Entwicklung konzentrieren kann, anstatt Zeit mit der Konfiguration von Spring zu verbringen. Dieser Entwicklungsansatz wird auch Rapid Application Development (RAD) gennant. Tatsächlich ist es so, dass im Hintergrund das normale Spring Framework verwendet wird, es jedoch mit einer bestmöglichen Konfiguration vorkonfiguriert wird. Ebenfalls sind bereits ausgewählte Libraries von Drittanbietern eingebunden. Die Konfiguration wird aus Spring's best-practices gewähl, kann jedoch manuell überschrieben werden.

Als Buildtools werden von Spring Boot sowohl Maven (ab 3.2), wie auch Gradle (ab 4.0) unterstützt. Andere Buildtools, wie Ant sind prinzipiell möglich, werden jedoch offiziell nicht unterstützt und werden auch nicht empfohlen. Es ist empfohlen, dass die Build Systeme dependency management unterstützen und Artefakte vom Maven Repository «Maven Central» konsumieren können [12].

Spring Boot befand sich beim Start dieses Projektes gerade in einer transitiven Phase von Version 1.5 zu Version 2.0. Da es sich hierbei um einen Major Release handelt, wäre eine Migration zu Version 2.0 wahrscheinlich mit einem grösseren Aufwand verbunden gewesen. Die grösste Änderung bei diesem Update ist die Verwendung der neuen Spring 5.0 Version.

.NET Core Web Application

Dies ist die jüngste Technologie, die evaluiert wurde. .NET Core ist eine neue quelloffene Version des älteren .NET Frameworks. Sie wurde entwickelt, um die neu entstehenden Anforderungen der Cloud zu entsprechen. Im Gegensatz zum .NET Framework, ist .NET Core dabei auf Plattformunabhängigkeit ausgelegt. Hauptentwicklungssprache ist C#. Seit Version 2.0 unterstützt .NET Core die meisten APIs, wie das .NET Framework. Unterstütze Plattformen sind Windows (ab Windows 7), MacOS (ab 10.12) und verschiedene Linux Distributionen [13]. Es werden ebenfalls ARM Architekturen unterstützt, was einen Einsatz auf einem mobilen Gerät (z.B. Raspberry Pi) ermöglicht.

Bei .NET Projekten wird meistens die Entwicklungsumgebung Visual Studio verwendet, wobei bei Core Projekten auch andere Entwicklungsumgebung (z.B. Atom, VSCode) Unterstützung anbieten. Die .NET Core SDK bietet eine CLI, um Prozesse, wie das Erstellen eines Projektes oder das Builden der Anwendung zu automatisieren. Als Package Manager wird das ebenfalls von Microsoft angebotene Tool Nuget verwendet. Dieses Werkzeug bezieht die Dependencies aus einem öffentlichen Nuget Respository.

Der in Frage kommende Projekttyp ist der ASP.NET Core Web API und beinhaltet eine einfache Struktur ähnlich der Struktur von Playframework. Im Vergleich zum Playframework oder Spring Boot, sind die bereits eingebundenen Module auf ein Minimum gehalten. Durch den stark modularen Aufbau von .NET Core, lassen sich die benötigten Funktionalitäten als Module einfach durch die CLI einbinden. Die vorwiegend verwendete Sprache C# ist im Vergleich zu Java sehr modern.

ExpressJS (NodeJS)

Ein relativ einfach gehaltenes Web-Framework für Node.js. Im Vergleich zu den anderen evaluierten Frameworks, ist Express relativ einfach gehalten. Eine Struktur der Anwendung ist nicht wirklich gegeben, weshalb hier die Erfahrung der Entwickler noch stärker zum Tragen kommt, als wenn eine grundlegende Struktur vorgeschlagen wird. Die erstellte Architektur liegt also fast komplett in den Händen der Entwickler.

Der verwendete Package Manager ist normalerweise Node Package Manager (npm), wobei hier eine grosse Menge an Alternativen besteht. Wie bei NodeJS Anwendungen üblich, ist JavaScript die einzig viable Entwicklungssprache. Dies bringt natürlich das – für das Team als Nachteil empfundene – Fehlen eines Typsystems für die Flux-Server Komponente. Das Verwenden von TypeScript wird von NodeJS nicht von Haus aus Unterstützt (obwohl es einige Templates gibt, die eine Unterstützung implementieren).

Eine mit den anderen Technologien vergleichbare Performance ist wohl erst mit viel Erfahrung und Aufwand zu erreichen. Das Implementieren einer guten Architektur auf einem ExpressJS

Server ist vor allem durch die grosse und offene Auswahl von Middlewares schwieriger zu bewerkstelligen, als bei den anderen Frameworks.

Fazit

Die Evaluation ergab bei den Teammitgliedern eine gewisse Richtung. Zum einen war ein typisiertes System eine must-have Anforderung für das Backend. Da das Projekt für einen produktiven Einsatz gedacht ist, soll die für das Backend verwendete Technologie sich bereits bei möglichst vielen kommerziellen Anwendungen bewährt haben. NodeJS fällt wegen diesen zwei Kriterien durch. Es gibt zwar kommerzielle Express Anwendungen mit NodeJS, jedoch hatten diese noch zu wenig Zeit, sich zu bewähren.

.NET Core ist ebenfalls eine relativ neue Technologie (neuer, als Express), jedoch baut sie auf das bewährte .NET Framework aus. Durch den traditionell starken Einfluss von Microsoft bei kommerziellen Anwendungen, scheint sich .NET Core schnell auch in produktiven Umgebungen durchzusetzen. Dennoch, wurde gegen .NET Core entschieden, da das Framework noch zu jung ist und erst seit .NET Core 2.0 überhaupt eine ähnliche Funktionalität anbieten kann, wie das .NET Framework. Für dieses Projekt kommen nur stabile Frameworks in Frage, weshalb .NET Core durch die Evaluation durchfiel.

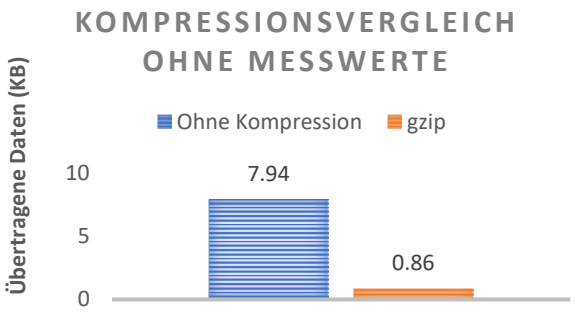
Die Auswahl zwischen Spring Boot und Playframework war sehr eng und wahrscheinlich hätten beide Technologien zu einem Erfolg des Projektes geführt. Beide Teammitglieder hatten bereits Erfahrungen mit den jeweiligen Frameworks gesammelt und waren zufrieden. Die Entscheidung fiel schlussendlich auf das Play Framework, weil es ein gutes Verhältnis zwischen einem modernen Framework und Stabilität bietet. Durch den Einsatz von Akka als Technologie im Backend, wurde ein starkes Argument für die Multithreading-Eigenschaften des Frameworks gesetzt, welches sich später durch den Projektverlauf auch bewahrheiten sollte. Weitere Argumente für das Framework waren die einfache Konfiguration, der bequeme Entwicklungsworkflow mit automatisiertes Kompilieren und die zentrale Routing Konfiguration in einer Datei.

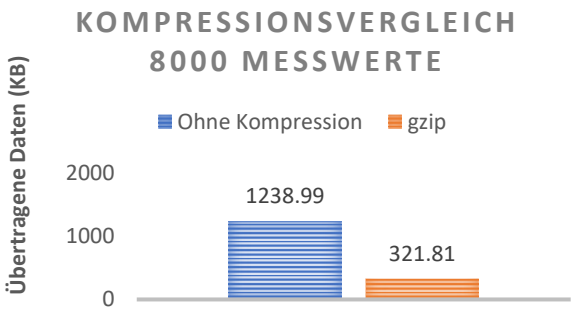
8.2.2 Kompression der übertragenen Nutzlast

Das Flux-Coordinator System soll auf möglichst vielen Plattformen bereitgestellt werden können. Die Komponenten müssen dabei nicht unbedingt auf der gleichen Hardware ausgeführt werden, sondern könnten sehr gut auch bei zwei unterschiedlichen Cloud Anbietern bereitgestellt werden. Auch die Qualität der Netzwerkverbindung zwischen Flux-Frontend und Flux-Server kann variieren. Wenn die Komponenten entfernt von einander sind, oder zum Beispiel über den Hotspot des Raspberries verbunden sind, kann davon ausgegangen sein, dass die Latenz der Verbindung relativ hoch sein wird. Um die übertragene Nutzlast und damit verbunden auch die Anzahl der Pakete und der benötigten Requests gering zu halten, wurde eine Kompression der Nutzlast evaluiert und später auch durchgeführt.

Die Einführung einer Kompression (in diesem Fall mit dem gzip Codec), geht mit einem Performance Impact beim Server und Client einher. In unserem Fall versuchen wir mit dem Server relativ schlank zu bleiben und deshalb ist das Verwenden einer Kompression nicht eine einfache Entscheidung. Es wurden verschiedene Tests durchgeführt, um die eine begründete Entscheidung zu fällen.

Vergleich der übertragenen Datenmengen

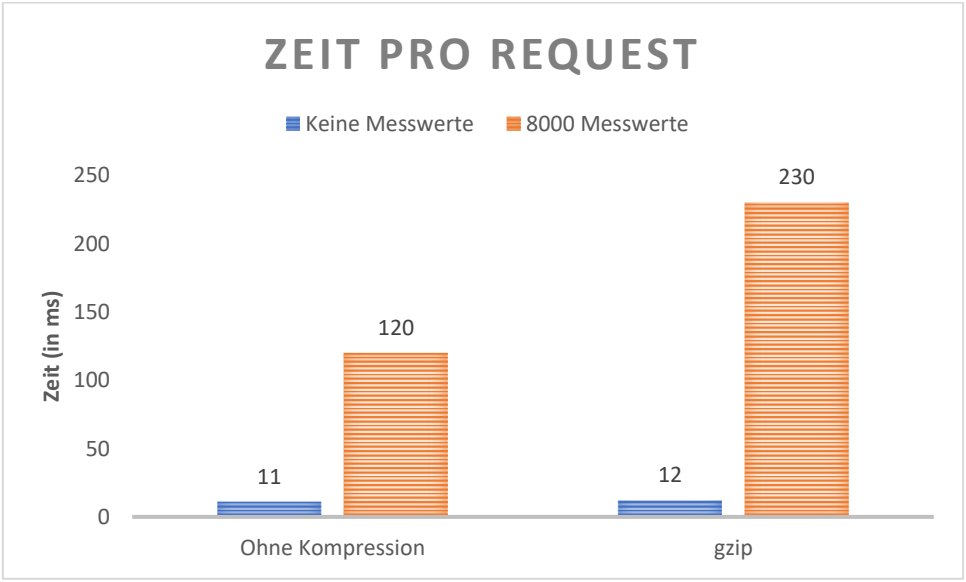




Durch die Verwendung der gzip Kompression, zeigt sich eine Verminderung der übertragenen Daten ohne Messwerte um bis zu 90%. Mit Messwerten ergibt sich noch eine Kompression auf knapp 25% der ursprünglichen Grösse.

Zeitaufwand vor und nach der Kompression

Die Kompression macht zwar einen grossen Unterschied in der Grösse der übertragenen Daten, doch auch im für die Kompression benötigten Rechenaufwand gibt es grosse Unterschiede. Der höhere Rechenaufwand zeigt sich in einer längeren Reaktionszeit auf Anfragen. Die längeren Antwortzeiten werden erst dann sichtbar, wenn die komprimierte Datenmenge steigt. Bei 8000 übertragenen Messwerten zeigt sich eine Verdopplung der Antwortzeit.



Fazit

Die Evaluation zeigt, dass die Kompression der zu sendenden Daten einen beträchtlich höheren Rechenaufwand nach sich zieht. Dieser wird sich vor allem bei grösseren Datenmengen bemerkbar machen. Dennoch ist anzunehmen, dass die Übertragung der vierfachen Menge an Bytes (8000 Messwerte ohne Kompression) über ein drahtloses Netzwerk ein Vielfaches der Zeit aufwenden würde. Bei einer Verbindung über einen Cloud Provider, würde sich die Latenzzeit der Anfrage wiederum um ein Vielfaches verlängern. Bei kleinen Datenmengen ist der Unterschied mit, oder ohne Kompression verschwindend klein und macht keinen praktischen Unterschied.

Ein Einsatz von gzip als Kompressionsalgorithmus ist aus den obigen Gründen wohl meistens die richtige Wahl, weshalb entschieden wurde, es als Default von Flux-Server einzusetzen. Möglicherweise ist eine Deaktivierung von Kompressionen angebracht, wenn die Flux-Server Komponente auf einem CPU-Schwachen Rechner (wie ein Raspberry Pi) ausgeführt wird. Die beste Lösung

wäre eine automatische Erkennung der verfügbaren Ressourcen und das dynamische ein oder abschalten der Kompression je, nach verfügbarer Leistung.

Anmerkungen zu den Messungen

Die empirischen Messungen, die für diese Auswertung durchgeführt wurden, zeigen die durchschnittlichen Antwortzeiten von 100 Requests in einer «localhost» Umgebung. Die Flux-Server Instanz wurde dabei auf einer Workstation der HSR im Development Modus ausgeführt. Die aufgezeigten Resultate können natürlich je nach Hardware variieren und sollten als Richtwert genommen werden. Die durchgeführten Abfragen von 8000 Messwerten sind im produktiven Einsatz eher selten anzutreffen. Eine Abfrage von 8000 Messwerten würde nur geschehen, wenn der Benutzer eine bereits durchgeführte Messung lädt. Dies ist ein Schritt, welcher voraussichtlich nicht in schneller Abfolge mehrmals durchgeführt werden wird und deshalb ist die Akzeptanz des Benutzers gegenüber der Reaktionszeit der Anwendung auch automatisch höher.

8.2.3 Übertragung neu einkommender Messwerte

Flux-Frontend bietet eine Heatmap an, die den aktuellen Stand der Messungen – für die Wahrnehmung des Benutzers – «live» anzeigen soll. Um diesen Effekt zu erreichen, muss die Flux-Frontend Komponente stets die letzten verfügbaren Messwerte anzeigen können. Deshalb wurde entschieden, die neu eintreffenden Messwerte über die WebSocket Technologie vom Server zum Frontend zu pushen.

Der gewählte Ablauf besteht aus einem Mix von GET Abfragen und dem öffnen und offen halten des WebSockets während sich der Benutzer in der Ansicht der Heatmap befindet. Da die WebSocket Verbindung ein Timeout besitzt, muss vom Client regelmässig ein keep-alive gesendet werden. Dieser wird vom Server verworfen.

Sobald das Frontend die Verbindung erstellt hat, wird die Verbindung im Server zwischengespeichert. Wenn der Server neue Messungen erhält, werden diese zuerst in in der Datenbank gesichert und direkt danach an den offenen WebSocket gesendet.

Play Framework setzt für WebSockets eine auf Akka Streams basierende Lösung ein. WebSockets werden dabei als Flows – Datenflüsse – modelliert und werden wie ein Stream behandelt. So entsteht eine saubere und sichere asynchrone Abarbeitung der einkommenden Messwerte. Das Schlagwort ist bei dieser Technologie (und dem Java Standard Reactive Flows) «Backpressure». Dabei gibt die Quelle des Datenflusses, also in diesem Fall flux-sensors den Takt an und pusht die Daten in den Verarbeitungsmechanismus. Das Ergebnis ist, dass der Server keine Zeit mit dem Warten auf Daten verliert und diese dann beim tatsächlichen Eintreffen sofort verarbeiten kann.

8.2.4 Asynchronität von Flux-Server

Beim Entwickeln von Flux-Server wurde viel Wert auf Effizienz gelegt, mit dem Ziel, dass die Anwendung auf möglichst vielen Plattformen – auch auf nicht sehr leistungsstarke Plattformen, wie ein Raspberry – funktioniert. Um die volle Leistung der Hardware verwenden zu können, ist Flux-Server auf eine sehr hohe Parallelität ausgelegt.

Das zugrundeliegende Playframwork setzt für die angestrebte Asynchronität schon gute Voraussetzungen, da es von Grund auf asynchron aufgebaut ist [14]. Die Controller, die die Requests abarbeiten sind dabei asynchron und verwenden Threads aus dem sogenannten «Default Execution Context». Ein Blockieren des Controller Threads ist nicht erwünscht, da in dieser Zeit andere Requests von diesem Thread nicht behandelt werden können und die Anwendung deshalb nicht gut skalieren würde. Aufgrund der Architektur des Projektes, werden von den Controller-Klassen

oft Funktionen des Data Access Layers (ab jetzt DAL) aufgerufen, die für eine längere Zeit blockieren können. Für ein gutes Skalieren der Anwendung ist es deshalb wichtig, dass diese besonders lange dauernde Operationen ausserhalb des Default Execution Context ausgeführt werden.

Alle Methoden des DAL's geben eine Promise zurück, die der Controller danach an das Backend des Playframworks übergibt. Das Framework wartet dann, bis die Promise (in Java CompletableFuture) fertiggestellt ist. Ein neuer Ausführungskontext mit dem Namen «Database Execution Context» wurde für das DAL erstellt. Dieser besitzt einen eigenen Threadpool, welches über die Konfigurationsdatei jederzeit angepasst werden kann. Beim Aufruf einer DAL-Methode durch einen Controller, wird die Ausführung der DAL-Methode in den Database Execution Context verschoben und der Thread des Default Execution Context wird für weitere Abfragen freigegeben. Sobald die DAL-Methode abgeschlossen ist, erhält der Controller den Rückgabewert über die Promise der DAL-Methode. Der Rückgabewert wird dabei wieder in den Default Execution Context gewechselt, damit der Controller den Kontext des Aufrufes nicht verliert. In diesem Schritt ist allenfalls eine Nachbearbeitung des Rückgabewertes der DAL-Methode möglich – geht aber mit einem negativen Impact der Skalierbarkeit der Anwendung daher. Die Promise des DAL's kann ansonsten direkt zum Playframework-Backend zurückgegeben werden, damit der Request vom Playframework beantwortet werden kann.

Hinweis: Die Anzahl der Threads in einem Ausführungskontext ist entscheidend für die Performance der Anwendung. Die Grösse des ThreadPools des Database Execution Context sollte der Grösse des Connection Pools für die Datenbank entsprechen, damit der Connection Pool voll ausgereizt werden kann, aber gleichzeitig keine Threads verschwendet werden. Der Connection Pool sollte gemäss dem ConnectionPool Provider (HikariCP) wie folgt berechnet werden [18]:

$$Datenbankverbindungen = ((Physische\ Prozessorkerne * 2) + Spindelanzahl\ der\ Festplatte)$$

Für einen Raspberry Pi mit vier physischen Prozessorkernen ergibt sich also ein Connection Pool von 9, das heisst der Database Execution Context sollte ebenso viele Threads beinhalten.

8.3 Flux-Sensors

- Python vs. C, keine Library für Lichtsensor (I2C)
- Wieso Raspberry und kein Mikrocontroller

Die Sensoren werden von einem Raspberry Pi gesteuert. Dieser Client übermittelt die Daten an die Schnittstelle über http POST Requests, sobald sie bereitstehen. Wichtig ist, dass die Sensordaten des Luxmeters und des Positionierungssystems lokal auf dem Client zusammen verschmolzen und mit einem Timestamp versehen werden.

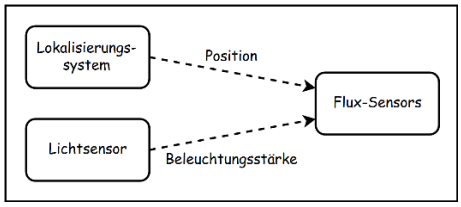
Hinweis: Der Grund, weshalb POST und nicht PUT Anfragen verwendet werden ist, dass PUT Anfragen idempotent sein sollen. Dabei ist es «good practice», wenn der Client dem Server bei einer PUT Anfrage auch den Namen gibt, unter welchem die Ressource zugreifbar sein soll. Im Falle der Messwerte behält jedoch der Server die Hoheit über die Datenhaltung.

8.3.1 Technologieevaluation
C vs. Python

8.3.2 Synchronisierung der Sensordaten

Problem

Die gemessenen Werte für die Beleuchtungsstärke müssen mit den Positionsdaten, bei denen sie erfasst wurden, übereinstimmen. Trotz der Synchronisierung soll die mögliche Anzahl Messwerte nicht zu stark vermindert werden.



Lösungsansatz

Beim langsameren Sensor, in unserem Fall Pozyx, wird per Polling ein neuer Messwert abgefragt. Dazu wird die Library-Funktion doPositioning()[9] aufgerufen, welche jeweils ca. 70ms dauert[10].

Anschliessend wird der letzte gemessene Lux-Wert im Register des Lichtsensors ausgelesen. Durch angleichen der Taktfrequenz des Lichtsensors auf die 70ms wird garantiert, dass in der Zwischenzeit auch jeweils ein neuer Lichtwert gemessen wurde.

Folgen

Als Anforderung an den Lichtsensor ergibt sich somit eine «Messzeit» von maximal 70ms. Andernfalls müsste eine zusätzliche Wartezeit beim Positionierungssystem hinzugefügt werden. Durch den eher aufwändigen Positionierungsalgorithmus[11] von Pozyx im Gegensatz zur Messung der Helligkeit sollte diese Anforderung in den meisten Fällen gegeben sein.

Daraus resultiert eine maximale Zeitliche Differenz der beiden Messwerte von 70 ms. Dies entspricht bei einer durchschnittlichen Schrittgeschwindigkeit von 5 km/h

$$5\frac{km}{h} \cdot \frac{1000}{3.6 \cdot 10^6} \cdot 70ms = 0.097\bar{2}m \approx 10cm$$

einer Distanz von ungefähr 10 cm.

[9] https://www.pozyx.io/Documentation/Datasheet/python
[10] https://www.pozyx.io/Documentation/Datasheet/RegisterOverview#POZYX_DO_POSITIONING
[11] https://www.pozyx.io/Documentation/how_does_positioning_work

8.3.3 Kommunikation

Die nachfolgenden Kommunikationsabläufe zwischen Flux-Sensors und Flux-Server sind als vereinfachte UML Sequenzdiagramme dargestellt.

Polling Flux-Server

Die Interaktion *Polling* in Abbildung 6 visualisiert den Wartezustand der Sensoreinheit, wenn ein Verbindungsversuch mit dem Flux-Server stattfindet oder auf die Aktivierung einer Messung gewartet wird.

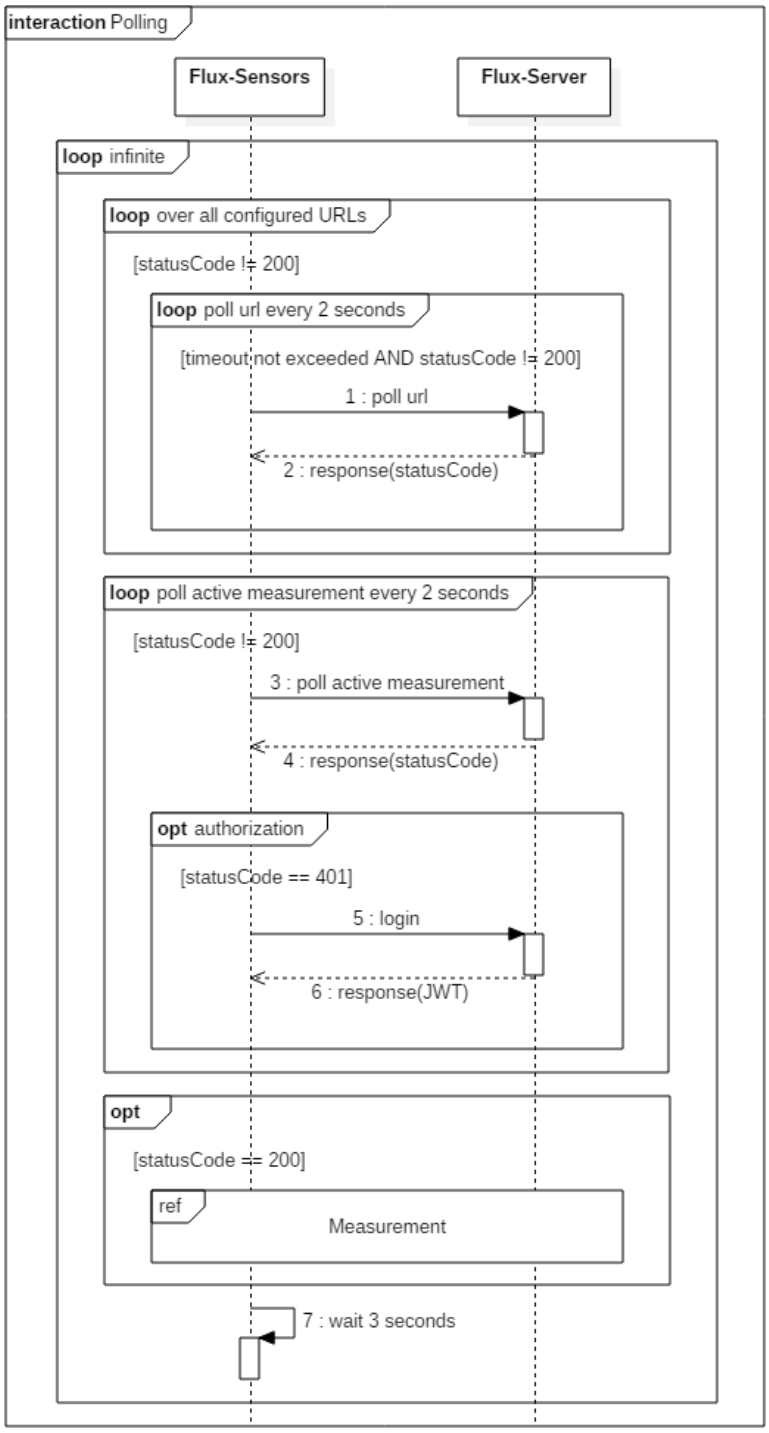


Abbildung 6 UML Sequenzdiagramm der Polling Interaktion des Flux-Sensors

Die Interaktion Polling kann in drei Teile unterteilt werden. Dies sind die beiden aufeinanderfolgenden Loop-Blöcke in der zweiten Ebene und der Opt-Block am Ende des Diagramms. Im ersten Teil werden alle per Konfigurationsdatei geladenen Server URLs nach ihrer Erreichbarkeit geprüft. Bei der ersten positiven Antwort wird die URL zwischengespeichert und in Phase zwei übergegangen. Hier wird nun der Server nach einer aktiven Messung abgefragt und erneut auf eine positive Antwort gewartet. Da die Route vor unbefugtem Zugriff geschützt ist, muss zuerst eine Autorisierung durchgeführt werden. Im dritten und letzten Teil der Interaktion wird die Messung schliesslich gestartet. Das Starten und Stoppen wird lediglich über den Rückgabewert der Route zur aktiven Messung realisiert. Für den Ablauf der Messdurchführung ist nachfolgend ein eigenes Sequenzdiagramm aufgeführt.

Der Übersichtlichkeit halber wurde auf die Darstellung der weiteren möglichen Server-Instanzen verzichtet. In der oben beschriebenen ersten Phase werden in der Realität zumeist verschiedene Server URLs kontaktiert, bis eine entsprechende Antwort zurückgegeben wird.

Messdurchführung

Das Sequenzdiagramm der Messdurchführung in Abbildung 7 zeigt nun auch die Kommunikation mit den beiden Sensoren für die Lichtmessung und die Positionsbestimmung (Pozyx).

Zu Beginn wird die aktive Messung abgefragt und daraus die Positionen der Pozyx Anchors und die Zielhöhe der Messung ausgelesen, um anschliessend die Sensoren mit diesen Daten zu initialisieren und für die Messung vorzubereiten. Dann startet der Loop mit den tatsächlichen Messungen. Die Schleife kann nur durch Erreichen des Timeouts oder durch Stoppen der Messung, mit einem Response Code 404, beendet werden. Der Ablauf startet mit der Positionsbestimmung über Pozyx und dem Auslesen des zuletzt gemessenen Lux-Werts. Einzelheiten zum Abfragen und Synchronisieren der Messwerte sind im Kapitel <Referenz zur Synchronisierung> ausgeführt.

Am Ende des Loops wird der letzte erhaltene Response Code abgefragt und ausgewertet, ob bereits eine Antwort vorliegt. Sollte dies der Fall sein, so wird ein neuer Batch von Messwerten an den Server gesendet. Ist die Antwort ein 404-Code, so wurde die Messung vom Server beendet. Da die Übermittlung der Messwerte asynchron geschieht, ist der Zeitpunkt des Eintreffens der Antwort nicht eindeutig bestimmt. Es kann zu Beginn oder am Ende passieren oder auch über mehrere Durchläufe der Schlaufe hinweg ausbleiben. Die gemessenen Werte werden in diesem Fall in einer Liste zwischengespeichert und gemeinsam als Batch an den Server übertragen. Der implementierte Ablauf zur Übertragung der Messwerte ist im Kapitel <Referenz zu Kapitel Übertragung> beschrieben.

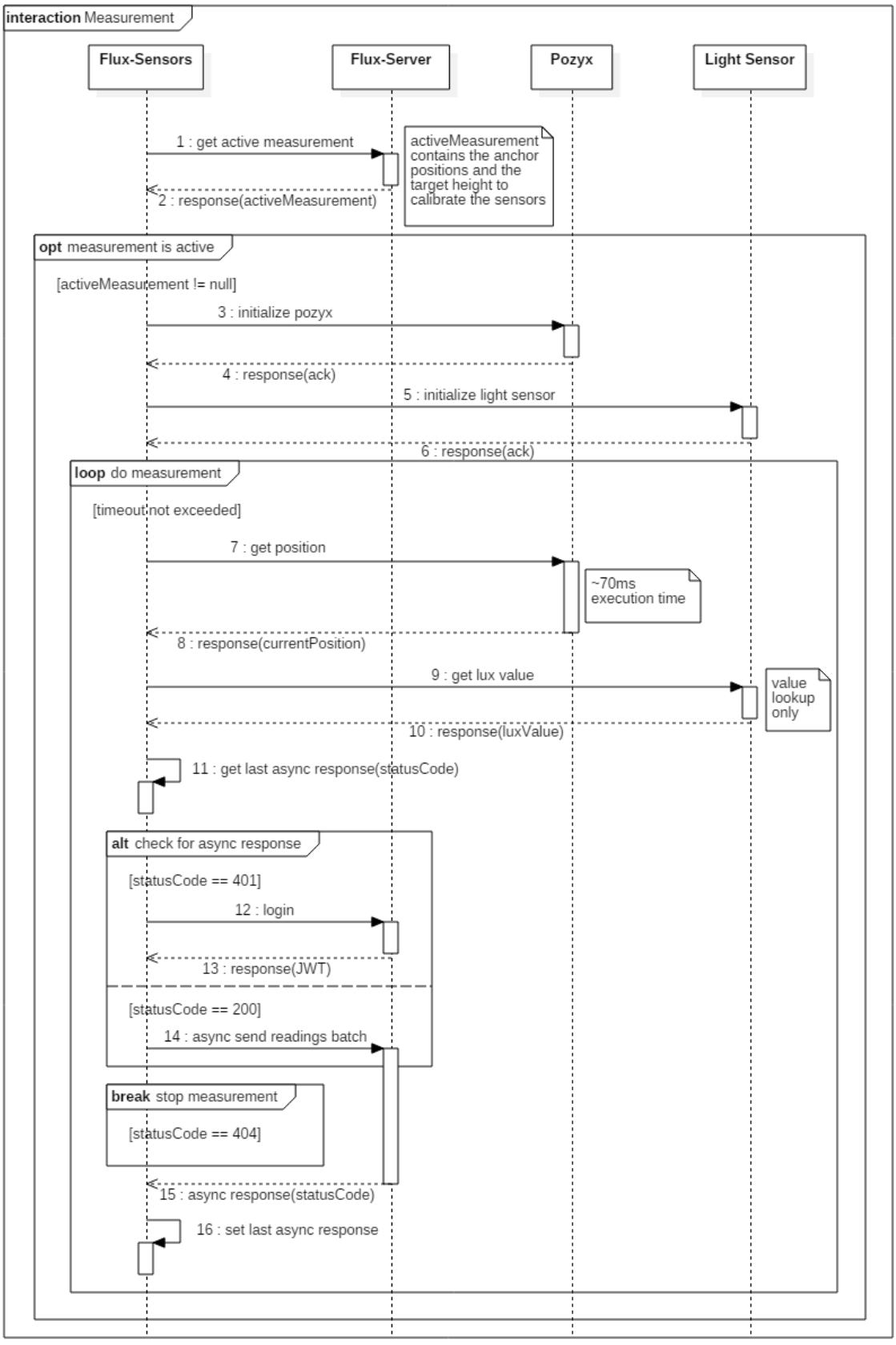


Abbildung 7 UML Sequenzdiagramm der Messdurchführung

8.3.4 Übertragung der Messwerte

In einem ersten Prototyp wurden die Messwerte synchron an den Server übertragen und erst nach einer positiven Antwort mit einer weiteren Messung begonnen. Dies entspricht der Abfolge ganz links aus Abbildung 8. Um den Durchsatz zu verbessern, wurde nachfolgend eine Optimierung vorgenommen.

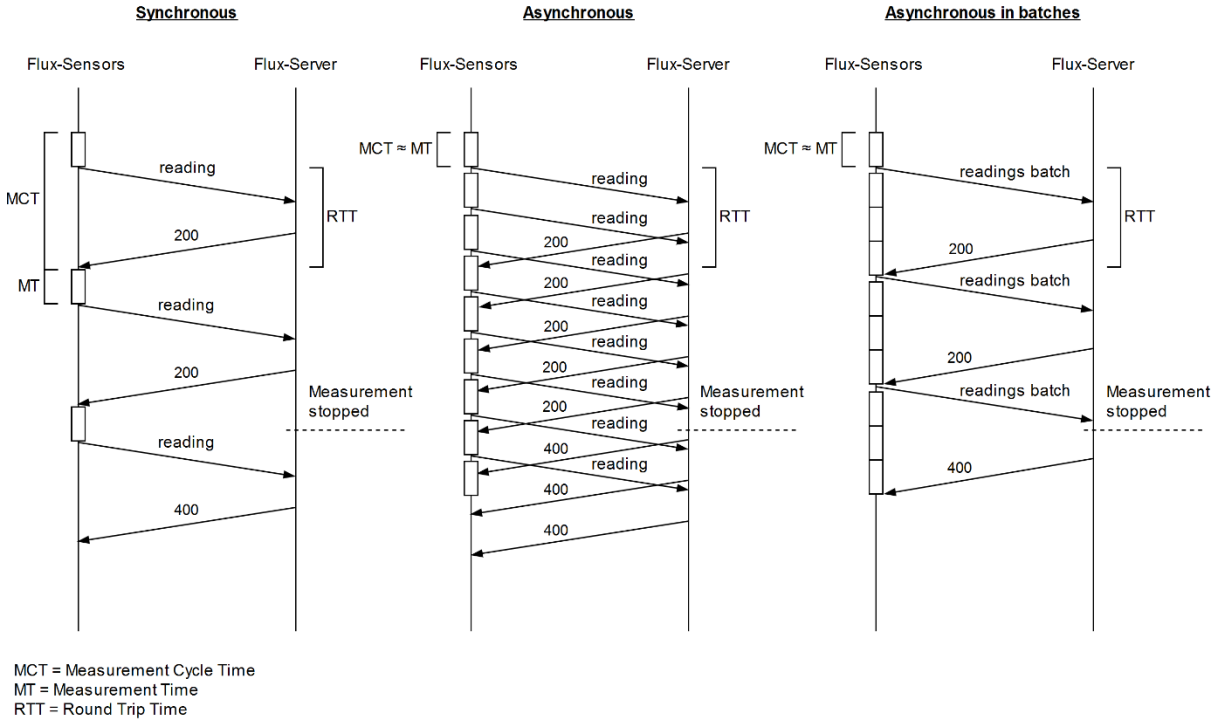


Abbildung 8 Sequenzdiagramm der Übertragung der Messwerte

Durch eine asynchrone Übertragung der Messwerte, wie in der mittleren Abfolge ersichtlich, kann in der Theorie beinahe der maximal mögliche Durchsatz erreicht werden. Es ist jedoch auch schnell zu erkennen, dass dabei im Gegensatz zur ersten Variante ein Vielfaches an Nachrichten zusätzlich versendet werden. Dieses Problem wird durch Variante drei gelöst, indem wie zu Beginn eine positive Antwort des Servers abgewartet wird und erst dann eine Liste mit den neuen Messwerten als Batch übertragen wird. So können die Vorteile beider Lösungen kombiniert werden.

Grundsätzlich gilt bei allen Varianten, dass die Paketumlaufzeit (RTT) und die Zeit für eine Messung (MT) konstant bleiben. Das Ziel der Optimierung ist also die Zeit für einen vollen Messzyklus (MCT) von Beginn einer Messung bis zum Beginn der nächsten Messung zu senken.

Als Technologie für die asynchronen Nachrichten wurde die Python Library requests-futures[12] gewählt, da sie besonders leichtgewichtig ist und gemeinsam mit der für die synchronen Abfragen verwendeten Library Requests[13] eingesetzt werden kann. So müssen die bisherigen Abfragen nicht geändert werden.

Um die Theorie zu bestätigen, wurden mit allen drei Varianten Messungen durchgeführt. Dabei wurde die Sensoreinheit einmal mit einem lokal installierten Flux-Server und einmal mit einem in der Heroku Cloud instanziierten Flux-Server verbunden und genau eine Minute lang gemessen.

[12] https://github.com/ross/requests-futures
[13] http://docs.python-requests.org/en/master/

Variante	Synchron		Asynchron		Asynchron in Batches	
Installation	Lokal	Cloud	Lokal	Cloud	Lokal	Cloud
Messungen pro Minute	500	119	714	224	754	853

Tabelle 4 Messung der Anzahl übertragener Messwerte pro Minute in drei Varianten

Die gewonnenen Resultate bestätigen die zuvor aufgestellte Theorie. Es fallen jedoch zwei widersprüchliche Resultate auf: Die asynchrone Übertragung in die Cloud ist fast um ein Vierfaches langsamer, als die Variante mit den Batches. Dabei sollte der Unterschied lediglich bei den zusätzlichen Verbindungsaufbauten liegen. Als zweites fällt auf, dass die dritte Variante mit der Cloud noch schneller ist als Lokal, wobei die RTT um ein Vielfaches grösser ist.

Der erste Widerspruch lässt sich durch die maximale Anzahl Worker Threads erklären. Diese ist standardmässig auf zwei begrenzt und erklärt somit das Resultat. Um dem obigen Diagramm gerecht zu werden, müssten also weitere Worker Threads zur Verfügung gestellt werden.

Das zweite Paradoxon mit der vermeintlich schnelleren Cloud-Verbindung wird dadurch verursacht, dass die lokale RTT kleiner ist, als die Zeit für eine Messung, und somit für jeden Messwert wieder eine neue Nachricht gesendet wird. Durch Festlegung einer minimalen Batch-Grösse konnte das Problem gelöst und die gleichen Resultate erzielt werden. Dies zeigt auch, dass die Grösse der RTT nun keinen Einfluss mehr auf den möglichen Durchsatz hat.

Als Überprüfung, ob das Optimum erreicht wurde, kann die Versuchszeit von einer Minute durch die konstanten 70ms der Positionsbestimmung dividiert werden:

$$\frac{60s}{7 \cdot 10^{-2}s} = 857.14$$

Dies entspricht ziemlich genau dem mit Variante drei maximal gemessenen Wert. Somit liegt der Flaschenhals nun bei der Positionsbestimmung von Pozyx.

8.3.5 Positionsdaten zu ungenau

- Mail Pozyx Support
- Störfaktoren im Laborraum

8.4 Datenbank

Die gemessenen Messwerte und auch die dazugehörigen Metadaten, wie Projekte und Räume werden in einer Datenbank gespeichert.

8.4.1 Technologieevaluation

Die Auswahl der richtigen Technologie für die Datenbank war keine leichte Aufgabe. Das System hat die spezielle Anforderung, dass es auf mehreren Plattformen lauffähig sein muss. Eine x86 Architektur soll genauso unterstützt werden, wie die ARM Architektur eines Raspberry Pi 3. Die Datenbank muss gut auf einer Cloudplattform bereitgestellt werden können. Die angestrebte Eigenschaft einer Multiplattform stellt auch spezielle Anforderungen an die Performance. Die Datenbank sollte eine gute Leistung mit sich bringen und auch effiziente Abfragen von tausenden von Messwerten zulassen.

Die Erfahrung der Teammitglieder war ausschlaggebend für die Auswahl der evaluierten Technologien. Völlig unbekannte Technologien stellen ein zu grosses Risiko für das Projekt dar, speziell

weil das System am Ende der Arbeit produktiv eingesetzt werden soll. Als zwei mögliche Datenbankkategorien kamen die traditionellen relationalen sowie auch dokumentbasierte Datenbanksysteme in Frage. Für das Abspeichern der Messwerte wäre auch eine Key-Value Storage Technologie, wie Redis denkbar gewesen. Der Zugriff auf Messwerte würde hierbei eine (amortisierte) konstante Zeit ($O(1)$) benötigen [15]. Die restlichen abzuspeichernden Daten, wie Projekte und Räume müssten auf einer anderen Datenbanktechnologie abgespeichert werden. Da das System schon aus mehreren Komponenten besteht und die Zugriffszeit der anderen evaluierten DBMS ausreicht, wurde entschieden auf eine Komplementärtechnologie spezifisch für Messwerte zu verzichten.

Als Implementationen der zwei ausgewählten Datenbankkategorien wurde PostgreSQL und MongoDB ausgewählt. PostgreSQL wurde im Rahmen der Datenbankmodule DBMS I und II genau angeschaut. MongoDB wurde von den Teammitgliedern schon in anderen Projekten eingesetzt. Wichtig für die Auswahl war ausserdem, dass die Software frei von Lizenzkosten – auch im produktiven Einsatz – ist. In den folgenden Unterkapiteln wird detailliert auf die Eigenschaften der zwei DBMS eingegangen.

PostgreSQL (Relationales DBMS)

Genau genommen ist PostgreSQL eine Objektrelationale Datenbank [16]. Die Unterschiede zu einem relationalen DBMS sind subtil. Die objektorientierten Eigenschaften finden in diesem Projekt vorerst keine Verwendung.

MongoDB (Dokumentbasiertes DBMS)

8.4.2 Datenbanktechnologie

Die Datenbank speichert die ermittelten Messwerte und enthält auch die Metadaten über Projekte, Räume und Messdurchführungen. Aufgrund des spezifischen Datenmodells und der im Projekt evaluierten Anforderungen wurde der Einsatz eines nicht-relationalen DBMS diskutiert. Der Entscheid fiel schliesslich auf ein dokumentbasiertes DBMS. Als Implementierung wurde aufgrund der grossen Verbreitung und der aktiven Community MongoDB gewählt. Die nachfolgenden Gründe waren für die Entscheidung relevant.

Einfachere Bereitstellung

Eine Anforderung an das Projekt ist eine einfache Bereitstellungsstrategie. Relationale DBMS (RDBMS) sind erfahrungsgemäss schwieriger bereitzustellen, als dokumentbasierte DBMS. Der Application Server bietet eine Schnittstelle im JSON Format an, weshalb sich eine Datenhaltung in JSON zusätzlich lohnt.

Flexibles Schema

Die Austauschbarkeit der Sensoren zählt zu den nichtfunktionalen Anforderungen und Ideen für diverse mögliche Erweiterungen wurden bereits diskutiert. Das flexible Schema einer NoSQL Datenbank würde sich dafür optimal eignen.

Synergien durch verwendete Technologien

Dadurch, dass JSON sowohl für die Datenhaltung, wie auch beim Application Server und im Frontend verwendet wird, erübrigt sich ein objektrelationales Mapping. Dies ist eine zusätzliche Entlastung für die Wartung des Projektes und steigert die Effizienz der Entwicklung.

8.5 Datenbankschema

Zwar entfällt bei einer dokumentbasierten Datenbank das Erstellen eines Schemas im traditionellen Sinn, doch die optimale Aufteilung der Daten spielt dennoch eine grosse Rolle. In MongoDB werden die Daten in Datenbank, Collections und Documents aufgeteilt. Diese drei Kategorien lassen sich grob in Datenbank, Tabellen und Zeilen für RDBMS übersetzen. Für dokumentbasierte DBMS muss jedoch gegenüber traditionellen RDBMS ein Umdenken stattfinden. Zum Beispiel sind in dokumentbasierten DBMS denormalisierte Daten öfter erwünscht, als in RDBMS.

8.5.1 References vs. Embedded Documents

Ein Document in MongoDB kann eingebettete Dokumente mit verwandten Daten enthalten. Generell ist es besser, Dokumente in andere Dokumente einzubetten, anstatt eine Referenz zu einem anderen Dokument (möglicherweise in einer anderen Collection) zu speichern. Es gibt jedoch Einschränkungen bei eingebetteten Dokumenten zu beachten. So ist die Dokumentgrösse auf 16 MB begrenzt, was bei mehreren Messungen mit je einer grossen Anzahl Messwerte längerfristig ein Problem darstellen könnte.

Referenzen bieten jedoch eine grössere Anpassbarkeit, als eingebettete Dokumente. Sie helfen auch, Dokumente klein zu halten und so die Performance von Abfragen zu verbessern.

Das Mapping der Datenbank mit den Workflowobjekten sieht wie folgt aus:

Database Die Datenbank beinhaltet die gesamten Daten des Systems.

Collection Eine Collection (analog zu Tabelle in RDBMS) teilt die verschiedenen Dokumente in Kategorien auf. Bei dokumentbasierten Datenbanken muss eine Balance zwischen dem Hinzufügen aller Werte in einem einzelnen Document oder dem Aufteilen der Werte in verschiedene Collections. Ein Document wird nämlich immer vollständig herausgegeben. Ein «herauspicken» von Daten innerhalb eines Documents gestaltet sich als schwierig.

Es gibt folgende Collections: projects und readings.

Document Ein Document beinhaltet einen konkreten Wert (analog zu Zeilen in RDBMS). Ein Beispiel für ein Dokument ist ein konkretes Projekt oder eine Messung.

8.5.2 Vorgehen bei der Modellierung

Bei der Modellierung einer dokumentbasierten Datenbank stellen sich gegenüber klassischen relationalen Datenbanken grundsätzlich drei Fragen:

- Soll ein Objekt referenziert oder eingebettet werden?
- Wie soll die Referenz implementiert werden? (Falls das Objekt nicht eingebettet wird)
- Sollen gewisse Daten denormalisiert werden? (Diese Frage wird z.T. auch in relationalen Datenbanken gestellt.)

Die Frage nach Referenz oder Einbettung darf nicht verwechselt werden mit der Frage nach Attribut oder eigener Tabelle in relationalen Datenbanken. Dort wird nämlich eher die Komplexität

der Daten untersucht, wobei die Überlegung in dokumentbasierten Datenbanken eine andere ist. Folgende Überlegungen sind für die obigen drei Fragen entscheidend:

- Use Cases und häufigste Queries: Was wird häufig zusammen abgefragt?
- Kardinalität der Beziehungen
- Volatilität der Daten

Use Cases

Für die Visualisierung einer Messung müssen stets alle zugehörigen Messwerte geladen werden. Für die Darstellung der Navigation reichen jeweils der Name und die Beschreibung der einzelnen Projekte, Räume und Messungen aus.

Kardinalität

Im Gegensatz zu relationalen Datenbanken zählt hier nicht nur die Unterscheidung zwischen 1:1, 1:n oder n:m, sondern sind vielmehr auch konkrete Zahlen von Bedeutung.

Ein Projekt hat in den meisten Fällen wahrscheinlich nicht mehr als eine Hand voll Räume mit je ebenso vielen Messungen. Eine Messung hingegen kann aus tausenden Messwerten bestehen.

Volatilität

Mit Volatilität ist die Langlebigkeit bzw. Flüchtigkeit der gespeicherten Daten gemeint. Optimalerweise sollten im selben Dokument vorwiegend Daten mit ähnlicher Volatilität beherbergt werden.

Eine abgeschlossene Messung mit ihren Messwerten wird im Nachhinein nicht mehr verändert. Name und Beschreibung von Projekt, Raum und Messung können sich ab und zu ändern. Sich häufig ändernde Daten sind in dieser Problemdomäne keine vorhanden.

Dies sind optimale Voraussetzungen für eine Denormalisierung. Das macht erst Sinn bei einem Verhältnis von vielen Lesezugriffen auf wenige Schreibzugriffe, denn der Nachteil einer Denormalisierung liegt in der erschwerten Aufrechterhaltung der Konsistenzbedingungen.

8.5.3 Ergebnis der Modellierung

Als Ergebnis wird ein einzelnes Dokument für sämtliche Projekte und Räume mit den Metadaten der Messungen erstellt. Zusätzlich wird pro durchgeführte Messung für alle Messwerte ein weiteres Dokument erstellt.

Für eine einfachere Verständlichkeit sind die aufgeführten Schemas mit Beispieldaten ausgefüllt. Eine ausführliche Dokumentation als JSON-Schema[14] ist im Anhang abgelegt.

[14] http://json-schema.org/

```
{
  "projects":[
    {
      "projectId":0,
      "name":"HSR 2018",
      "description":"Messungen an der HSR im 2018",
      "rooms":[
        {
          "roomId":0,
          "name":"Zimmer 1.262",
          "description":"SA/BA Labor",
          "floorPlan":"91837645918723641982466345 6.png",
          "width":17300,
          "length":5600,
          "floorSpace":76.25,
          "measurements":[
            {
              "measurementId":0,
              "name":"Messung 1",
              "description":"Erste Probemessung",
              "creator":"Paul Schmidt",
              "startDate":"2018-03-23 13:01:23",
              "endDate":"2018-03-23 13:17:56",
              "state":"done",
              "targetHeight":800,
              "heightTolerance":50,
              "offset":125,
              "factor":1.5,
              "anchorPositions":[
                {
                  "anchorId":0,
                  "name":489,
                  "xPosition":0,
                  "yPosition":0,
                  "zPosition":2000
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```


Abbildung 9 Datenbankschema der Metadaten

```
{
  "measurementId":0,
  "readings":[
  {
    "readingId":0,
    "luxValue":489,
    "xPosition":14849,
    "yPosition":2316,
    "zPosition":822,
    "timestamp":"2018-03-23 13:01:36"
  }
  ]
}
```

Abbildung 10 Datenbankschema einer Messdurchführung

8.6 Probleme & Lösungsansätze

8.6.1 Verfügbarkeit der verteilten Systeme

Das auszuliefernde System besteht aus mehreren voneinander gekapselten Komponenten. Einige der Komponenten sind abhängig von der Funktionsfähigkeit der anderen Komponenten. Um effizient entwickeln zu können, muss zum Beispiel der Application Server verfügbar sein. Wenn der Application Server nicht verfügbar ist, können die Sensoren keine Daten an das Frontend übermitteln. Der Application Server seinerseits ist von der Datenbank abhängig, sonst kann er keine Daten abspeichern.

Um diese Abhängigkeiten möglichst einfach zu erfüllen, wurde entschieden, auf Continuous Delivery zu setzen. Nach dem Durchlaufen und Bestehen der Tests einer Komponente, wird diese automatisch auf einem Heroku Dyno bereitgestellt und ist somit für die anderen Komponenten über das Internet verfügbar. Durch diese ständige Verfügbarkeit kann sich der Entwickler stets darauf verlassen, dass die für die Entwicklung benötigten Komponenten erreichbar sind und muss nicht manuell eigene Instanzen deployen. Somit kann sich der Entwickler besser auf die eigentliche Arbeit konzentrieren.

8.6.2 Einschränkungen bei der Datenbankgrösse

Überarbeiten

Durch das Auswählen von MongoDB als Datenbank für das Abspeichern von Daten, müssen wir auch die Nachteile von MongoDB beachten. Eines davon ist, dass die maximale Dokumentgrösse 16 MB beträgt. Diese Dokumentgrösse wird erreicht, wenn übermässig viele Daten gesammelt werden. Es müssen hierbei mehrere Stunden lang Messwerte gesammelt werden. Das System hat mehrere Mechanismen, um das Benutzererlebnis durch diesen Nachteil hoch zu halten. Mechanismen:

- Kompression der Daten in der Datenbank (https://www.mongodb.com/blog/post/new-compression-options-mongodb-30)
- Anzeigen eines Balkens im Frontend, mit dem aktuellen «Füllwert» des Dokuments (wie ein Ladebalken).
- Warnung über das Frontend an den Benutzer, wenn sich das Dokument zu sehr füllt.

Um das Problem vollständig zu beheben gibt es ebenfalls mehrere Möglichkeiten, welche sich allerdings ausserhalb des Scopes dieser Arbeit befinden:

- Datenbank ändern auf RDBMS oder zum Beispiel CouchDB (maximale Dokumentgrösse auf CouchDB beträgt 4GB).
- Verwenden der GridFS Technologie von MongoDB.

8.6.3 Heatmap Library Issues

- Durchschnitt der einzelnen Datenpunkte nehmen anstatt addieren
- Interpolation am Rand (nicht gelöst) -> evtl. Idee?

8.6.4 Konsequenzen einer öffentlich verfügbaren Applikation

Sicherheitsfeatures

Indexierung von Suchmaschinen

8.6.5 Raspberry Pi als Hotspot

Die Idee war, den bei der Vermessung notwendigen WLAN-Router durch das eingebaute Wifi-Modul im Raspberry Pi zu ersetzen.

Es wurden zwei verschiedene Open Source Lösungen[15][16] evaluiert, allerdings scheint die Verwendung als Wifi Client und Hotspot gleichzeitig nicht möglich zu sein[17]. Auch die Verbindung ist nicht gleich stabil, wie mit einem dedizierten WLAN Router.

Als Lösung wird weiterhin ein WLAN Router eingesetzt, um eine bessere Verfügbarkeit des Systems zu garantieren.

8.7 Qualitätssicherung

8.7.1 Testing der Sensor-Komponente

Sensoren zu testen ist grundsätzlich schwierig.

Ansatz: Register und I2C Bus gemockt und dem Lichtsensor-Modul bei der Instanziierung mitgegeben. Am Ende das Register mit einem korrekt initialisierten Register verglichen.

[15] https://github.com/sabhiram/raspberry-wifi-conf
[16] https://github.com/billz/raspap-webgui
[17] https://github.com/billz/raspap-webgui/issues/151

9 Versuchsaufbau

9.1 Hardware

9.2 HSR Labor

Nachteil: viele Hindernisse (PC-Bildschirme, Personen, störende elektronischen Geräte)

Grundriss und Foto zeigen

9.3 HSR Forschungszentrum Lichtraum

10 Schlussfolgerung und Ausblick

10.1 Erfüllte und nicht erfüllte Anforderungen

- TargetHeight weggelassen, da die Z-Werte von Pozyx eine zu grosse Streuung haben (evtl. mit Messung belegen)
- Keine Interpolation am Rand (Issue link)

Möglichkeiten zur weiteren Automatisierung und Verbesserung der Usability:

- Mehrere aktive Messung erlauben (in Zukunft eine Art Mapping zwischen Sensor und Messung denkbar)
- Generische / Vordefinierte Setups der Pozyx Anchors (z.B. quadratisch mit einstellbarer Seitenlänge)
- ScaleFactor automatisch berechnen, sodass bereits zu Beginn alle Anchors im Bild sind und nur noch

10.2 Beurteilung der Ergebnisse

Einsatztauglichkeit?

10.3 Projekt und Aufgabenstellung

Herausforderung: Fertiges Produkt anstatt Prototyp -> Fokus war eher ganzheitlich und es blieb wenig Zeit einer einzelnen Sache genauer nachzugehen (z.B. Resultate der Positionsbestimmung und Lichtmessung verbessern)

10.4 Einsatz und Weiterverwendung des Flux-Coordinators

Probleme: mehrere verschiedene Technologien eingesetzt

Vorteile: Software ist in unabhängige und separat «deploybare» Komponenten aufgeteilt und kann so in Zukunft modular weiterentwickelt werden oder einzelne Komponenten können bei Bedarf auch komplett ersetzt werden (z.B. Python Flask Server für mehr Performance anstatt Java Play Server)

11 Literaturverzeichnis

[1] International Organization for Standardization, «Software engineering -- Product quality -- Part 1: Quality model,» 2001.

[2] E. Evans, Domain-Driven Design. Tackling Complexity in the Heart of Software, Addison-Wesley, 2003.

[3] S. Brown, «The C4 model for software architecture,» [Online]. Available: https://c4model.com/. [Zugriff am 8 April 2018].

[4] K. Ross, «Flow support is broken as of 0.57.0,» 27 November 2017. [Online]. Available: https://github.com/mui-org/material-ui/issues/9312. [Zugriff am 08 Juni 2018].

[5] Facebook Inc., «create-react-app/README,» Facebook Inc., 23 März 2018. [Online]. Available: https://github.com/facebook/create-react-app/blob/master/packages/react-scripts/template/README.md. [Zugriff am 09 Juni 2018].

[6] «angular/angular,» 09 Juni 2018. [Online]. Available: https://github.com/angular/angular. [Zugriff am 09 Juni 2018].

[7] «Angular - Angular Glossary,» Google, [Online]. Available: https://angular.io/guide/glossary#typescript. [Zugriff am 09 Juni 2018].

[8] Vue.js, «Comparison with Other Frameworks,» Vue.js, 3 Juni 2018. [Online]. Available: https://vuejs.org/v2/guide/comparison.html. [Zugriff am 09 Juni 2018].

[9] V. Cromwell, «Between the Wires: An interview with Vue.js creator Evan You,» 30 Mai 2017. [Online]. Available: https://medium.freecodecamp.org/between-the-wires-an-interview-with-vue-js-creator-evan-you-e383cbf57cc4. [Zugriff am 09 Juni 2018].

[10] «Vue CLI | Overview,» 08 Juni 2018. [Online]. Available: https://cli.vuejs.org/guide/. [Zugriff am 09 Juni 2018].

[11] L. R. w. chantastic, «Container Components,» 7 März 2015. [Online]. Available: https://medium.com/@learnreact/container-components-c0e67432e005. [Zugriff am 09 Juni 2018].

[12] P. Webb, D. Syer, J. Long, S. Nicoll, R. Winch, A. Wilkinson, M. Overdijk, C. Dupuis, S. Deleuze, M. Simons, V. Pavić, J. Bryant und M. Bhave, «Spring Boot Reference,» [Online]. Available: https://docs.spring.io/spring-boot/docs/2.0.2.RELEASE/reference/htmlsingle/#using-boot-build-systems. [Zugriff am 06 06 2018].

[13] L. Coward und R. Lander, «core/2.1-supported-os,» 1 Juni 2018. [Online]. Available: https://github.com/dotnet/core/blob/master/release-notes/2.1/2.1-supported-os.md. [Zugriff am 06 Juni 2018].

[14] Lightbend Inc., «Handling asynchronous results,» 3 Mai 2018. [Online]. Available: https://www.playframework.com/documentation/2.6.x/JavaAsync. [Zugriff am 08 Juni 2018].

[15] «Redis,» [Online]. Available: https://redis.io/topics/memory-optimization. [Zugriff am 10 Juni 2018].

[16] PostgreSQL Global Development Group, «PostgreSQL: About,» [Online]. Available: https://www.postgresql.org/about/. [Zugriff am 10 Juni 2018].

[17] MongoDB Inc, "JSON and BSON," MongoDB Inc, [Online]. Available: https://www.mongodb.com/json-and-bson. [Accessed 11 03 2018].

[18] B. Wooldridge, «About Pool Sizing,» HikariCP, 8 Januar 2017. [Online]. Available: https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing. [Zugriff am 08 Juni 2018].

Anhang

A. Beispielanhang

Json-Schema, Swagger API, Mockups, usw.

B. Glossar

C. Installation Raspberry Pi

Für die Installation des Raspberry Pi sind nachfolgend eine manuelle und eine automatisierte Variante beschrieben. Um Fehler zu vermeiden wird die automatisierte Installation empfohlen. Die manuelle Installation kann als Checkliste zur Fehlersuche oder bei der Verwendung einer anderen Hardware verwendet werden. Die Resultate der beiden Vorgehensweisen sind identisch und wurden mit folgenden Raspberry Pi Modellen getestet:

- Raspberry Pi 3 Model B+[18]

Manuelle Installation

Als erstes muss das Betriebssystem des Raspberry Pi heruntergeladen werden. Die Installation wurde getestet mit dem aktuell empfohlenen Raspbian Stretch in der Version Lite ohne grafische Benutzeroberfläche.

Das Image kann unter folgender Quelle bezogen werden: https://www.raspberrypi.org/downloads/raspbian/

Anschliessend muss das Image auf die SD-Karte geladen werden. Dieser Vorgang ist abhängig vom lokal installierten Betriebssystem. Es kann die offizielle Anleitung befolgt werden: https://www.raspberrypi.org/documentation/installation/installing-images/.

Bevor die SD-Karte in den Raspberry Pi eingefügt wird, müssen noch zwei Dateien hinzugefügt werden, um das System im «Headless Mode» ohne Monitor und Tastatur betreiben zu können.

Folgende Dateien müssen ins Root-Verzeichnis der SD-Karte hinzugefügt werden:

- Leere Datei mit dem Namen ***ssh***, um den ssh Service automatisch zu aktivieren
- Konfigurationsdatei mit dem Namen ***wpa_supplicant.conf*** und folgendem Inhalt:

```
1  country=CH
2  ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
3  update_config=1
4
5  network={
6      ssid="flux-network"
7      scan_ssid=1
8      psk="fcwFCKLKVa5C7j6R3Jrt"
9      key_mgmt=WPA-PSK
10 }
```

> **Hinweis:** Dateiname, -endung und Inhalt müssen genau der Beschreibung oben übereinstimmen, damit sie vom Raspberry Pi korrekt geladen und verarbeitet werden. Die Datei *ssh* hat keine Dateiendung.

Nun kann die SD-Karte ausgeworfen und in den Raspberry Pi eingefügt werden. Nach Abschluss des Boot-Vorgangs kann eine SSH-Verbindung aufgebaut werden:

```
$ ssh pi@192.168.1.147
```

[18] https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

Hinweise:

- Der verwendete Computer und der Raspberry Pi müssen sich im selben WLAN-Netzwerk mit der oben beschriebenen Konfiguration befinden (Datei *wpa_supplicant.conf*).
- Die IP-Adresse des SSH-Befehls muss mit der Adresse des Raspberry Pi übereinstimmen. Diese kann beispielsweise im WLAN-Router ausgelesen werden.
- Das Default Passwort des Benutzers *pi* ist *raspberry*.
- Unter Windows kann eine SSH Client Software, wie z.B. PuTTY verwendet werden.

Nach der Anmeldung kann der Raspberry Pi über folgenden Befehl konfiguriert werden:

```
sudo raspi-config
```

Es müssen folgende Einstellungen vorgenommen werden:

- Dateisystem erweitern
- Hostname: fluxpi
- User-Passwort: fluxPiUser
- I2C aktivieren
- Zeitzone einstellen

Nach diesen Einstellungen muss der Raspberry Pi neugestartet werden und anschliessend erneut eine SSH-Verbindung aufgebaut werden.

Nun soll das Betriebssystem mit folgenden Befehlen aktualisiert werden:

```
sudo apt-get update
sudo apt-get upgrade
```

Und anschliessend folgende Software installiert werden:

Todo: Installation Flux-Sensors und Access Point

Automatisierte Installation

Für die automatisierte Installation des Raspberry Pi wird die Software PiBakery[19] benötigt. Nach der lokalen Installation und Ausführung von PiBakery kann unter *Import* die Konfigurationsdatei \Ressourcen\RaspberryPi\PiBakery\fluxpi.xml geladen werden.

[19] http://www.pibakery.org/

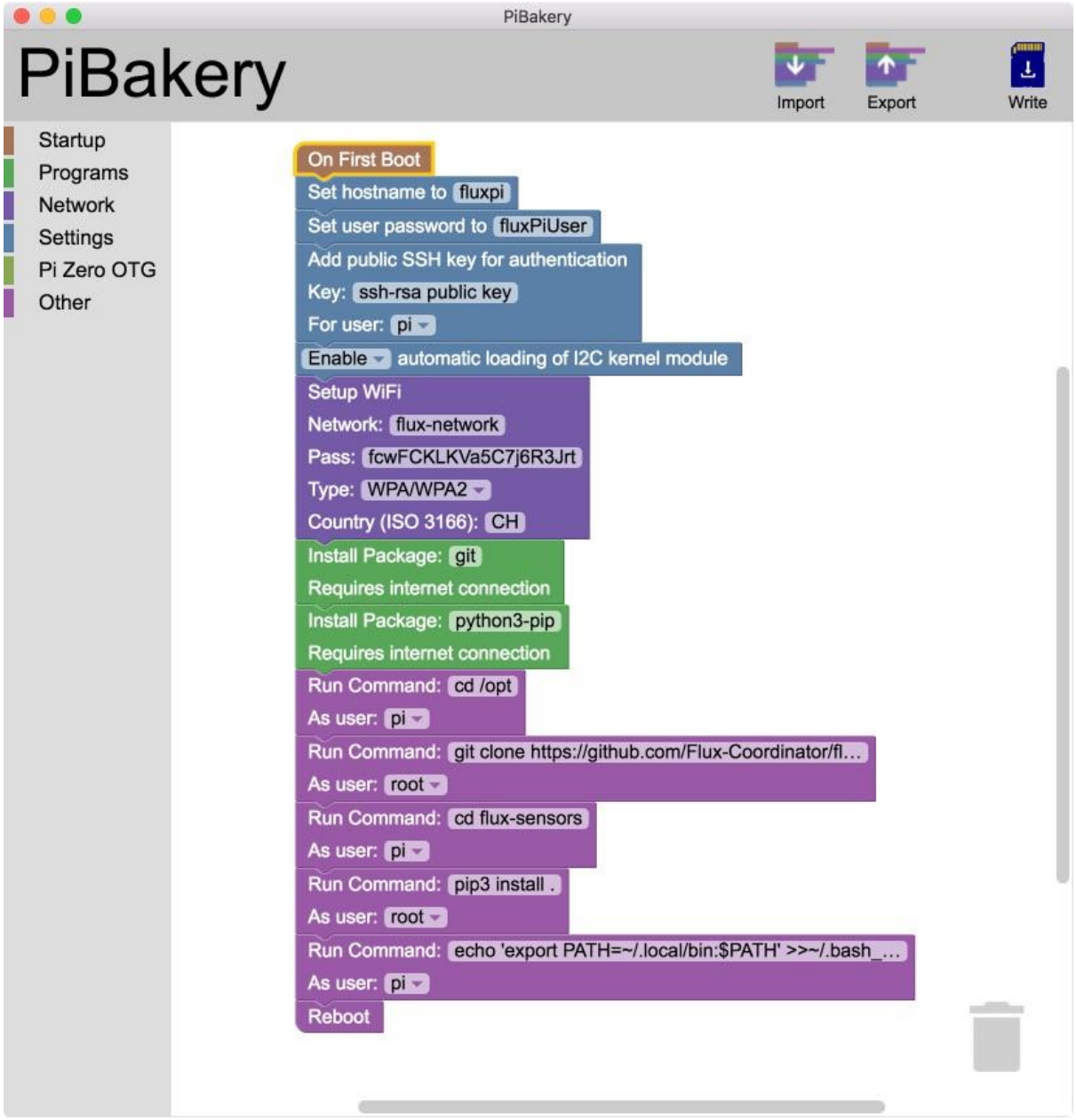


Abbildung 11 PiBakery mit importierter Konfiguration

Sofern die SD-Karte am Computer angeschlossen ist, kann die Konfiguration unter *Write* auf die SD-Karte geschrieben werden. Als Betriebssystem sollte ebenfalls Raspbian-lite gewählt werden.

Warnung: Das gewählte Laufwerk wird gelöscht und überschrieben, das heisst alle Daten, die darauf gespeichert sind, werden dauerhaft gelöscht.

Hinweis: Für das Herunterladen der GitHub Repositories (git clone) wurde ein PiBakery Block der Community verwendet, der zur Zeit dieser Bachelorarbeit noch nicht zu PiBakery hinzugefügt wurde.

Pull Request: https://github.com/davidferguson/pibakery/pull/127

Installationsanleitung: http://www.pibakery.org/docs/contribute.html

Nach Abschluss des Schreibvorgangs kann die SD-Karte ausgeworfen und in den Raspberry Pi eingefügt werden. Direkt nach dem Einschalten des Raspberry Pi werden die Scripts ausgeführt und das System vorbereitet. Dieser Vorgang kann einige Minuten dauern.

Analog zur manuellen Installation kann nun per SSH auf den Raspberry Pi verbunden werden. Als erstes sollten die Logs von PiBakery unter /boot/PiBakery/ überprüft werden, um festzustellen, ob die Installation erfolgreich abgeschlossen wurde. Mit dem Befehl `tail -f /boot/PiBakery/firstboot.log` kann die Installation überwacht werden.

Danach müssen über den Raspbian Wizard noch einige letzte Einstellungen manuell konfiguriert werden:

```
sudo raspi-config
```

Es müssen folgende Einstellungen vorgenommen werden:

- Dateisystem erweitern
- Zeitzone einstellen

Installation von PiJuice

PiJuice[20] ist eine portable Stromversorgung für den Raspbery Pi. Die Installation der Management Software pijuice-base[21] ist optional, wird jedoch empfohlen, falls die PiJuice Hardware verwendet wird:

```
sudo apt-get install pijuice-base
```

Anschliessend sollte geprüft werden, ob der PiJuice Service korrekt ausgeführt wird:

```
systemctl status pijuice.service
```

Über den Befehl `pijuice_cli.py` kann PiJuice anschliessend auf der Kommandozeile konfiguriert werden. Weitere Informationen sind in der Dokumentation[22] im GitHub Repository zu finden.

An drei benutzerdefinierten Buttons können Standardfunktionen oder eigene Scripts zugewiesen werden. Die Werkseinstellung kann ebenfalls in der Dokumentation im GitHub Repository gefunden werden[23].

[20] https://uk.pi-supply.com/products/pijuice-standard
[21] https://github.com/PiSupply/PiJuice
[22] https://github.com/PiSupply/PiJuice/tree/master/Software#pijuice-cli
[23] https://github.com/PiSupply/PiJuice/tree/master/Software#buttons