Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Der Intelligente Spiegel - Ein Companion zur Unterstützung der Selbstwahrnehmung Prototyp 1.0

Maria Lüdemann
Hauptprojekt Master
Wintersemester 2017/2018

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Maria Lüdemann

Hauptprojekt Master
Wintersemester 2017/2018

Der Intelligente Spiegel - Ein Companion zur Unterstützung der Selbstwahrnehmung Prototyp 1.0 eingereicht im Rahmen des Grundprojekts
im Studiengang Master Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuer: Prof. Dr. Kai von Luck und Dr. Susanne Draheim

Abgegeben am 21.Januar 2018

Inhaltsverzeichnis

1 Einleitung

Die Debatte um die Forschung rund um Daten im Raum Qunatified Self, Fitness und Gesundheit wird immer größer, dabei sind sowohl die Vorteile und die enormen Möglichkeite, wie auch die Gefahren die daraus erwachsen Thema des Diskurses. In dieser und in den vorangegangenen Arbeiten wurde sich damit beschäftigt, welchen Nutzen solche Daten für den Benutzer selber haben können. Können vom Nutzer im Alltag erhobene Daten eine Selbsterfahrung erweitern ohne direkt Druck aufzubauen? Können Mehrwerte generiert werden und dabei die Daten möglichst lokal gehalten werden, um sie nicht mit Firmen aller Art oder Krankenkassen zu teilen?

In dieser Arbeit soll beschrieben werden, was aufbauend auf den Seminararbeiten Lüdemann (2016) (eine nähere Auseinandersetzung mit Quantified Self), Lüdemann (2017b) (das Umfeld der intelligenten Spiegel) und dem Grundprojekt Lüdemann (2017a)(ein erster Prototyp eines eigenen intelligenten Spiegels) am Intelligenten Spiegel entwickelt wurde. Ziel dieser Arbeit ist es, die Plattform des Spiegels voran zu bringen und in einem Durchstich der Architektur die einzelnen Komponenten möglichst in Kommunikation zueinander zu bringen sowie eine Datenhaltung zu ermöglichen.

In den folgenden Kapiteln soll gezeigt werden ,dass es Entwicklungen sowohl in der architektonischen Planung wie auch in der prototypischen Umsetzung gibt. Ziel des Hauptprojekts war es, den Datenfluss zu ermöglichen und dafür die ersten Schritte zu gehen, um die Kommunikation der einzelnen Teile untereinander zu erproben und in einem ersten Abschnitt zu implementieren. Des Weiteren fehlte es noch an einer Persistenz der Daten auf die der Spiegel zugreifen kann. Diese muss ausgewählt und aufgesetzt werden, um darauf erste Analysen und Visualisierungen zu testen und eine Idee davon zu entwickeln, in welche Richtung es in der Masterarbeit gehen kann.

Damit soll das Spiegel Projekt auf einen Stand gebracht werden, der es ermöglicht in der Masterarbeit möglichst schnell die Möglichkeit zum Datenfluss hergestellt werden kann, um sich auf die Verarbeitung und Analyse der Daten zu konzentrieren.

2 Protopyp - Architektur

In dieser Arbeit soll beschrieben werden, was im Rahmen des Hauptprojekts am Intelligenten Spiegel realisiert wurde.

Die bisherige Architektur wurde im Sinne der drei Schichten Architektur betrachtet. Diese unterteilt sich in drei Schichten (Anzeige, Logik, Persistenz). In Lüdemann (2017a) wurde die Architektur im Detail erklärt, deshalb wird hier nur kurz darauf eingegangen. Die folgende Abbildung 1 zeigt das Architekturbild mit der Einordnung

in die Schichten aus dieser Arbeit. Die Architektur zeigt sechs Komponenten. Dabei handelt es sich unter anderem um die Datenbankkomponente, die sich um die Speicherung der Daten und die Kommunikation der Datenbank mit den anderen Teilen kümmert. Sie ist mit einem schwarzen Rahmen umgeben und findet sich in der Datenbank-/Persistenzschicht wieder. Alle nicht umrandeten Komponenten gehören der Logikschicht an. Zum Einen die Datenbereinigungskomponente, die die Daten aus den Sensoren abholt und für die weitere Verarbeitung bereinigt, zum Anderen die Verarbeitungs- und Analysekomponente, die die Daten aufbereitet und analysiert. Des Weiteren die Visualisierungskomponente die geeignete Visualisierungen erzeugt und an die Anzeige weiter leitet. Ebenso die Kommunikationskomponente die die Kommunikation des Nutzers von der Anzeigekomponente entgegennimmt und verarbeitet. Rot umrandet ist die Anzeigekomponente. Sie kümmert sich um das Anzeigen der von der Logikschicht bereitgestellten Daten auf einem geeigneten Medium, welches im Augenblick der Spiegel ist.

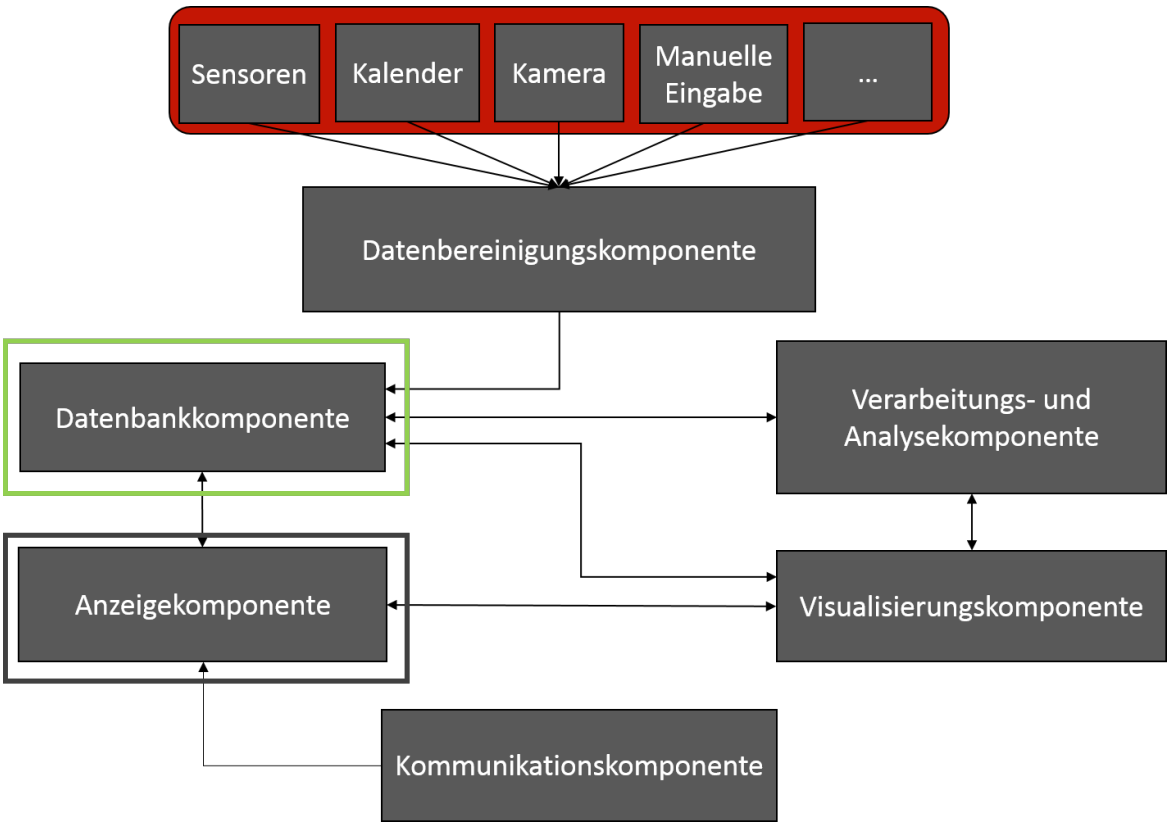

Abbildung 1: Architektur

Da es sich bei dem Spiegel um eine verteilte Anwendung handelt, muss betrachtet werden wie sie verteilt werden soll. Dafür wurde die vertikale Verteilung aus Tanen-

baum und van Steen (2008) betrachtet und die Spiegelarchitektur dort eingeordnet. Am besten geeignet schien dabei ein Thin Client Modell, zu sehen in Abb. 2 a). Auf der Anzeigefläche, hier dem Spiegel, liegt möglichst wenig Logik, somit kann die Berechnung von rechenstarken Servern übernommen und in Hochsprachen geschrieben werden. Darüber hinaus bietet diese Verteilung die Möglichkeit, Teile der Anzeige auch auf anderen Geräten anzubieten, zum Beispiel als Einsicht der Daten über ein Tablet oder Smartphone. Der Unterschied zu der in der Abbildung gezeigten Version b) ist, dass auf dem Clienten nicht alle Grafiken und Visualisierungen erstellt werden, sondern auch diese zum Teil von der Logik vorbereitet um dann vom Frontend nur noch angezeigt zu werden. Dies findet sich unter anderem in der Entscheidung wieder, die Visualisierungskomponente in die Logikschicht zu nehmen. Anhand diesem aktuellsten Stand der Architekturplanung wurde das Hauptprojekt durchgeführt.

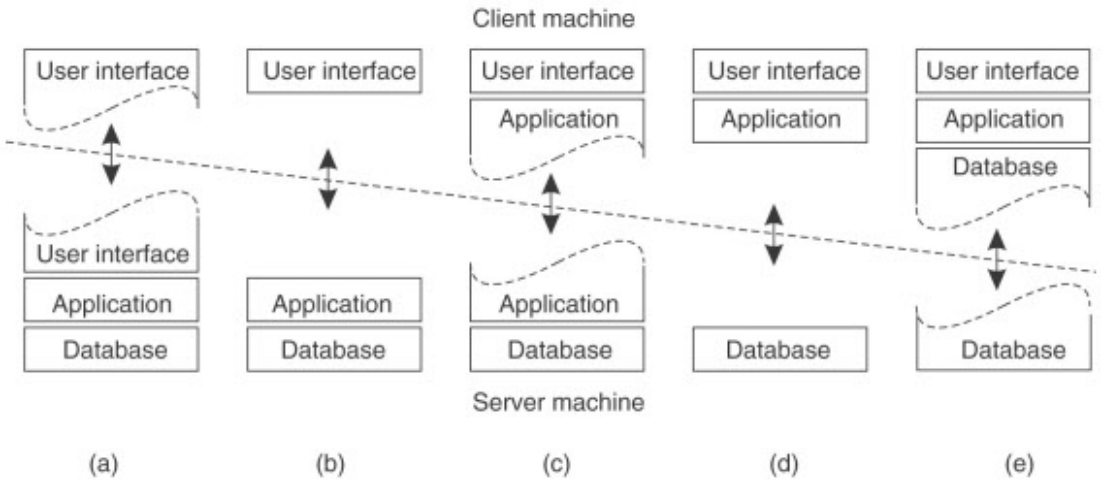

Abbildung 2: Three Tier Architecure

Um die prototypische Implementation voran zu treiben, wurden in einem Durchstich durch die Architektur essentielle Komponenten genauer betrachtet und ausgebaut. Dies betrifft die Anbindung der Anzeigekomponente an die Middleware und die Datenbankkomponente, da eine Datenbank getestet und für erste Daten verwendet wurde. Um die Daten verwenden zu können, wurde die Datenbereinigungskomponente um ein Modul erweitert, das Daten in das Datenbank-spezifische Format bringt. Durch die Art der Datenbank konnte ein erster Test der Datenvisualisierung umgesetzt werden, um zu erproben, welche Visualisierungen sich eignen würden.

Die einzelnen Schritte werden in den folgenden Abschnitten genauer erklärt, dabei wird nicht das Schichtenmodell von oben nach unten durchgearbeitet, sondern in der Rei-

henfolge des Bearbeitens. Änderungen in der Datenbankschicht führten zu Änderungen in der Logik, welche neue Möglichkeiten mit sich brachten.

2.1 Anzeige

Die Anzeigeschicht ist die Ansicht, die der Benutzer auf dem Spiegel sieht. Sie soll das Spiegelbild sowie diverse Darstellungen von Daten umfassen. Derzeit ist, wie auch schon in Lüdemann (2017a) beschrieben, eine HTML Übersicht eingebettet in ein C# Projekt das Herzstück der Anzeige. Sollte sich die Wahl der Kamera ändern, ist es möglich das C# Projekt zu ändern, da die Wahl des C# Projekts an der im Augenblick verwendeten Kamera hängt. Im Moment wird die Kinect 2 genutzt. Die HTML Oberfläche soll davon unabhängig sein und ist so auch konzipiert. Sie ist auch ohne das Spiegelbild im Browser (nur lokal) aufrufbar. Dies ist zum einen praktisch für das Testen und Entwickeln, ermöglicht darüber hinaus aber auch eine Verwendung für andere Geräte als den Spiegel. Außerdem entspricht es so dem modularen Gedanken die einzelnen Teile möglichst unabhängig voneinander zu implementieren, um sie ggf. leichter erweitern oder austauschen zu können.

2.1.1 Middlewareintegration

Ein essentieller Schritt für die schon bestehende Seite war es, den Datenfluss anzubinden, das heißt die statischen Daten auf der Prototyp 1.0 Seite variabel zu machen und durch Daten die über die Middleware (das gewählte Kommunikationsmedium) abgefragt werden, zu aktualisieren. Dafür wird eine Middleware Integration der Anzeigekomponente sowie der Datenbank bzw. des Datenbankconnectors benötigt. In dieser Arbeit wurde die Anzeigekomponente angebunden, um das Arbeiten mit der Middleware zu erlernen und in der Masterarbeit die Datenbank leichter anbinden zu können.

Die Middleware ist eine publish-subsrcibe Anwendung, die eine Kommunikation zwischen verteilten Anwendungen ermöglicht. Dabei kann ein Agent unter einer Gruppe Nachrichten posten, die für diejenigen sichtbar sind die aus dieser Gruppe lesen. Dabei wird kein Queuing betrieben, sodass die Kommunikation in Echtzeit funktioniert Eichler (2014).

Abbildung 3 verdeutlicht, wie die Kommunikation zwischen den einzelnen Schichten der Drei-Schichten-Architektur(Three Tier Architecure) gedacht ist. Dabei sollen die einzelnen Schichten nur über die Middleware zur nächst höheren oder niedrigeren Schicht kommunizieren. So soll es z.B nicht möglich sein, von der Anzeigeschicht direkt auf die Datenbank zuzugreifen, sondern es sollen immer über die Logik des Systems Anfragen gestellt werden. Dies stellt zum Einen sicher, dass nicht Daten durch die

Oberfläche angezeigt werden können, zu denen der Benutzer vielleicht keinen Zugriff haben sollte. Zum Anderen kann so gewährleistet werden, dass die Daten erst für den Nutzer vorbereitet werden können, um eine verbesserte Ansicht zu bieten. Dies bedeutet aber auch, dass die in den vergangenen Arbeiten geplante Architektur siehe Abb. 2 so nicht ganz realisierbar ist, da auf dem Bild eine Kommunikation zwischen der Anzeigekomponente und der Datenbankkomponente zu sehen ist. Diese Kommunikation muss also über die Datenverarbeitungskomponente bzw. Visualisierungskomponente laufen, um Daten zur Ansicht auszuwählen bzw. zu visualisieren. Da die Daten, die auf dem Spiegel angezeigt werden, abgesehen vom Spiegelbild, keine Echtzeitdaten sind, macht dies auch keine Probleme, was die Latenz angeht. Da das Abfragen der Daten in regelmäßigen Abständen und nicht in Echtzeit erfolgt (z.B bei Pulsdaten), kann der Benutzer vor dem Spiegel nicht sehen, wie sein Puls jetzt in diesem Augenblick ist, sondern sieht Daten, die mindestens eine Abfrageperiode zurück liegen. Dies könnten für Pulsdaten Zeitintervalle von einer bis mehreren Stunden sein. Dadurch müssen die Berechnungen nicht in Echtzeit geschehen. Die Abfragen, die der Nutzer vor dem Spiegel stellt, sollten dahingegen möglichst schnell auf dem Spiegel angezeigt werden können. Deswegen sollten die Visualisierungen auf einem leistungsfähigen Server erstellt werden.

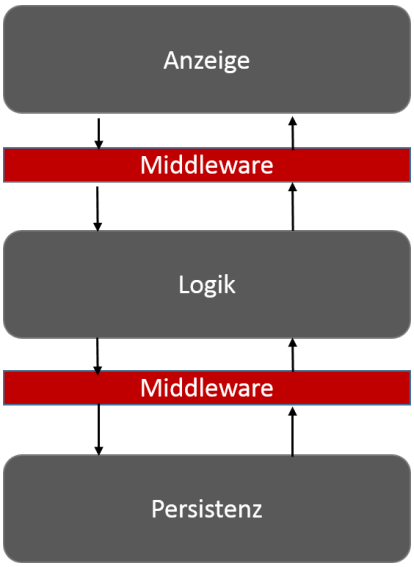

Abbildung 3: Kommunikation zwischen den Schichten

Um diese Anbindung zu integrieren war es notwendig, einen Connector für die HTML Seite zu schreiben, der mittels Javascript die Middleware anbindet. Da dies eine noch relativ wenig genutzte Sprache im Rahmen der Middlewareanbindung ist, mussten erst diverse Hürden überwunden und auch Fehler behoben werden. Die neuste Ver-

sion der Middleware im CSTI ermöglicht nun eine deutlich einfachere und weniger fehlerbehaftete Nutzung per Javascript.

Neben dem Connector wurde die HTML Ansicht variabel gestaltet, dafür wurde die Ajax [1] Bibliothek genutzt, um asynchrone Datenübertragung zu ermöglichen. Der aktuelle Stand umfasst eine Weboberfläche, die auf dem Spiegel läuft und über die Middleware Daten anfragen kann und die diese von einem Java Connector in der Logikschicht erhält. Diese Daten sind im Augenblick Dummydaten, da die Persistenz noch nicht an die Middleware angeschlossen ist.

2.2 Datenbank

Die Datenbankschicht wird im Augenblick mit mehreren Datenbanken geplant. So soll eine Datenbank die aus den APIs der Wearables gezogenen Dateien als Backup speichern, um die Rohdaten nicht zu verlieren und um sie zu späteren Zeiten und nach evtl. Fehlern neu auswerten zu können. Des Weiteren ist es möglich, dass durch später hinzukommende Informationen die im Augenblick herausgefilterten Teile der Daten interessant werden könnten.

Eine weitere Datenbank soll die bereinigten zeitspezifischen Daten erfassen und mit möglichst internen Mitteln erste Analysen zulassen. Dafür eignet sich eine Zeitreihendatenbank (DalmatinerDB, InfluxDB, Prometheus, etc). Sie eignet sich besonders für Zeitreihendaten (Puls, Schritte, Schlaf), aber auch für weniger häufig auftretende, aber auf Zeitstempel basierende Daten wie Gewicht oder Blutdruck, da sie diese Daten anhand ihres Zeitstempels gruppiert, aggregiert und direkt in zeitliche Reihenfolgen bringen kann.

2.2.1 Persistenz InfluxDB

In erster Instanz wurde die Zeitreihendatenbank aufgesetzt, um erste Analysen und Datenansichten zu generieren und somit eine Idee davon zu bekommen welche Anforderungen die Datensätze haben aber auch um zu sehen, welche Möglichkeiten mit der Zeitreihendatenbank kommen.

Bei der Auswahl der Datenbank wurde besonders auf einen einfachen Einstieg und eine kostenfreie Basisversion geachtet, die schnell für Testzwecke aber auch im späteren Lauf aufgesetzt werden kann. Des Weiteren sollte sie eine angenehme Abfragesprache haben und Daten einfach integrieren können. Ebenfalls sollten gewisse Analyse- oder Grafikmöglichkeiten bestehen. Die Wahl fiel auf die InfluxDB, da sie einfach mit einem Docker Containier aufgesetzt werden kann, kostenlos zur Verfügung steht, eine SQL

[1] https://developer.mozilla.org/de/docs/Web/Guide/AJAX

ähnliche Abfragesprache besitzt und es ermöglicht, große Datenmengen über Dateien einzulesen. Die Anzeige und Auswertungsmöglichkeiten neben der Abfragesprache selber werden dadurch ausgemacht, dass die Datenbank relativ einfach an Grafana angeschlossen werden kann und darüber diverse Möglichkeiten für explorative Datenanalyse bestehen. InfluxDB besitzt außerdem eine gut geführte Dokumentation.

Die InfluxDB ist eine schemalose Datenbank, die auf die Daten anhand von Labeln zugreift. Das untenstehende Codesnippet zeigt, wie die Datenstruktur der Datenbank zum Einfügen von Daten aussieht.

```
<measurement>[,<tag-key>=<tag-value>...] <field-key>=<field-value>
[,<field2-key>=<field2-value>...] [unix-nano-timestamp]
```

Einem Datensatz ist ein *measurement* zugewiesen, also zum Beispiel die Sensor Art der erfassten Daten, darunter können viele Datenpunkte zusammengefasst, sowie verschiedene Datensätze unterschiedlicher Zeitpunkte darüber vereint werden. Des Weiteren kann ein Datensatz mehrere *field-key , field-value* Paare haben. Diese stellen die Messdaten dar. Also zum Beispiel *Puls, 89* oder *Systole, 102*. Dadurch können zu einem Messzeitpunkt viele zueinander gehörende Daten erfasst werden. Als letztes wird jedem Datensatz ein Zeitstempel mitgegeben, der im Beispiel oben in Nanosekunden vergeben wird.

Die InfluxDB wurde in einem Dockercontainer aufgesetzt, der eine lokale Platte mountet, um die Daten zu hinterlegen. Dies geschieht vor allem für Testzwecke auf einem mobilen Rechner und soll im finalen Setup auf einem rechenstarken Server im CSTI aufgesetzt werden.

Um die vorhandenen Daten in die Datenbank zu ziehen, mussten diese erst noch einmal geparst, werden um sie in das oben beschriebene Format zu bringen und die vorhandenen Zeitstempel in Nanosekunden umzurechnen. Im Abschnitt 2.3.1 wird beschrieben, wie dies umgesetzt ist und welche Schwierigkeiten es dabei gab. Nachdem die Daten in die Datenbank geschrieben wurden, wurde an der Anbindung an Grafana gearbeitet um erste Visualisierungen zu erhalten. Die Ergebnisse dessen sind in Abschnitt 2.3.2 beschrieben.

2.3 Logik

Die Logikschicht des Projekts musste erweitert werden, um die Wahl der Datenbank zu unterstützen. Die vorhandenen Parser mussten erweitert werden, um Daten in das von der InfluxDB benötigte Format zu bringen. Nachdem die Daten in die InfluxDB geschrieben wurden, wurden erste Testvisualisierungen in der Grafana Oberfläche erstellt, die wie die InfluxDB auf dem mobilen Rechner in einer Docker Umgebung läuft.

2.3.1 Datentransformation

Als erstes wurden die Blutdruckdaten exemplarisch geparsed um im Rahmen der Masterarbeit die anderen Daten problemlos nachziehen zu können. Dafür mussten die durch Semikolon separierten Daten in das InfluxDB Format gebracht werden.

Ein ungeparster Beispieldatensatz:

```
Datum - Uhrzeit;Systole;Diastole;Puls;Stimmung;Aktivität;Anmerkungen
"02.06.2015 - 15:39";"114 mmHg";"73 mmHg";"71 bpm";"0";"2";"  "
```

Hier zu sehen sind unter Anderem Informationen, die nicht benötigt werden aber dennoch aus der Schnittstelle des Herstellers abgerufen werden wie zum Beispiel die Stimmung, Aktivität und Anmerkungen. Diese Daten fallen beim Parsen weg. Dazu kommen die Einheiten der Messwerte, die nicht mit in die Datenbank aufgenommen werden sondern als implizites Wissen zu sehen sind. Das Datum im Quelldatensatz im Format *DD.MM.JJJJ - hh:mm* muss in Nanosekunden umgerechnet werden. Dies machte beim ersten Test mit der Datenbank jedoch Schwierigkeiten, da die Zeit aus dem Quelldatensatz nur auf Minuten genau ist. Das Umrechnen in Nanosekunden führte zu Fehlern beim Einlesen in die Datenbank. So musste die Zeit auf die Unixzeit im Sekunden Zeitstempel umgerechnet und die Datenbank auf die Nutzung mit Sekundenzeitstempeln umgestellt werden.

Der obige Datensatz sieht in der geparsten Version wie folgt aus:

```
bloodpressure,user=01 systole=114,diastole=73,puls=71 1433252340
```

Zu sehen ist, dass das gewählte *Mesurement bloodpressure* ist, um darunter alle Werte der Blutdruck Messungen zusammen zu fassen. Der Tag *user* wurde gewählt, um eine Benutzung durch mehrere Benutzer zu vereinfachen, indem jeder Datensatz einem Benutzer eindeutig durch eine ID zugeordnet wird. Nach dieser ID kann bei Abfragen gruppiert werden. Eine Messung des Blutdrucks besteht immer aus drei Werten und einem Zeitstempel. Die Werte sind die Systole, die Diastole und der Puls. Diese werden jeweils als *field-key* Paar dargestellt.

Ein weiterer Fehler trat beim Einlesen der Datei in die Datenbank auf, da InfluxDB Linux/iOS Zeilenendungen benötigt, das genutzte System aber Windows Zeilenendungen verwendet. So musste die Datei einmal dahingehend überarbeitet werden, dass die richtigen Zeilenendungen verwendet werden.

Um den Erfolg des Importierens zu testen können sämtliche Datensätze des *mesurements bloddpressure* über die Konsole mit folgendem Befehl abgerufen werden:

```
curl -i -G 'http://192.168.99.100:8086/query?db=mydb&precision=s'
 --data-urlencode 'q=SELECT * FROM "bloodpressure"'
```

In diesem Befehl zeigt sich auch die Nähe zur Abfragesprache SQL. Das Ergebnis der Abfrage zeigt die Abbildung 6 im Anhang.

Im Augenblick befinden sich Blutdruckmesswerte vom 2.06.2015 bis zum 12.12.2017 in der Datenbank. Dies umfasst ca. 1049 Datensätze, da am Anfang einige Zeit zweimal am Tag gemessen wurde, später nur noch einmal am Tag (abzüglich von Messfehlern und Messausfällen aus diversen Gründen).

2.3.2 Datenvisualisierung

Nachdem die Daten in die Datenbank geschrieben wurden, wurde eine Grafana Instanz aufgesetzt, die über einen Docker Container gestartet und über eine Weboberfläche geöffnet werden kann. Darin lässt sich InfluxDB als Quelle auswählen und über die Oberfläche diverse Abfragen auf dem Datensatz erzeugen. Die Abbildung 7 im Anhang zeigt, wie solche Abfragen auf der Oberfläche aussehen.

Hier zu sehen ist oben im Bild eine komplette Abfrage, die sich durch Vorgaben zusammen klicken lässt. Dadurch können Abfragen sehr schnell und ohne ein hohes Maß an Sprachkenntnis erstellt werden. Des Weiteren bietet Grafana verschiedene Berechnungen für passende Datensätze. In der Abbildung zu sehen ist zum Beispiel, dass der Median der Werte gebildet wird und die Lücken zwischen den Werten linear gefüllt werden, um einen Grafen zu bekommen wie in Abbildung 4 gezeigt. Ohne das lineare Auffüllen wäre keine Durchgängige Linie zu sehen, da nur an einem Zeitpunkt am Tag (oder Zweien) überhaupt Daten vorliegen. Die Abbildung 7 zeigt zudem die vier Abfragen, die für die Grafik 4 benutzt werden, die Grafik ist im Anhang noch einmal größer zusehen (siehe Abb. 5). Dies sind jeweils eine Abfrage für Systole, Diastole und Puls, alle mit den gleichen Einstellungen, und einmal eine Art Trendwert für die Systole, dies kann einfach durch eine weitere Abfrage über Grafana erzeugt werden.

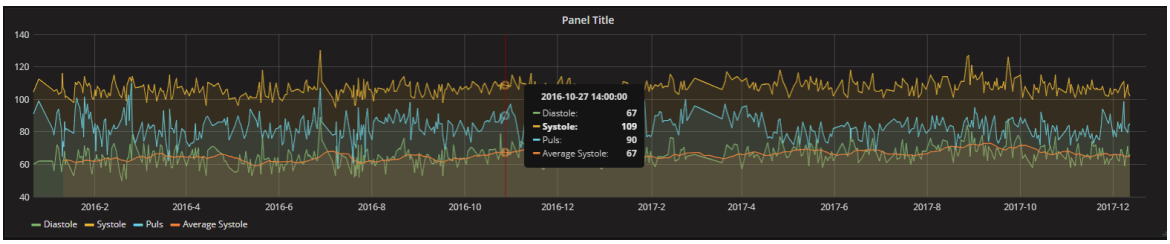

Abbildung 4: Visualisierungsbeispiel in der Grafana Oberfläche

In Abbildung 4 ist eine der Visualisierungen zu sehen, wie sie Grafana auf der Weboberfläche anbietet, diese hier zeigt eine Übersicht der Blutdruckmesswerte als Liniendiagramm. Mit dem Cursor kann über die einzelnen Messpunkte gefahren werden um deren genaue Daten zu sehen, wie in dem kleinen Kasten etwa mittig zu sehen. Die einzelnen Werte lassen sich durch ihre Farbe anhand der Legende zuordnen.

Diese Grafik wurde genutzt, um eine Idee davon zu entwickeln wie nützlich diese oder ähnliche Grafiken sein können und ob sie sich für die Nutzung auf dem Spiegel eignen. Bei so detaillierten Visualisierungen ist das fragwürdig, sie sollten allerdings dem Nutzer zur Verfügung gestellt werden, da sie durchaus dazu geeignet sind, die Daten zu erfassen und eigene Schlüsse daraus zu ziehen. Des Weiteren gibt sie einen guten Überblick und kann, vor allem auch mit Trendwerten, dazu genutzt werden eine Veränderung über Zeit zu erahnen. Diese Daten können visualisiert eine andere Bedeutung entfalten als als Zahlenwert. (Bspl. Ein mittlerer Datenanstieg von 3 mmHg)

In einem nächsten Schritt wurden die Werte genauer betrachtet, die besonders auffällig sind. Dies können entweder Messungenauigkeiten sein, normale Ausschläge oder aber auch Anzeichen auf Veränderungen durch andere Einflüsse. Im ersten Schritt wurden die drei auffälligsten Werte(Diastole höher als 120 mmHg) näher angeschaut, die Messwerte sind der untenstehenden Tabelle zu entnehmen.

Wert	**28.06.2016**	**27.08.2017**	**23.09.2017**
Diastole	89	82	88
Systole	130	127	126
Puls	107	87	87

Tabelle 1: Werte mit einer Diastole von mehr als 120 mmHg

Diese Daten wurden mit den Kalenderdaten abgeglichen daraus ergab sich, dass jeder dieser Datensätze mit einem enormen Stressfaktor zusammen zu hängen scheint, da jedem dieser Werte ein Ereignis mit hoher Belastung voraus geht.

Als nächster Schritt in der Masterarbeit sollen Wetterdaten in die Datenbank eingepflegt werden um zu sehen, ob sich anhand des Wetters, z.B der Temperaturen, Korrelationen finden lassen. Des Weiteren könnten weitere Anomalien heraus gesucht werden, bei denen die Werte höher als normal erscheinen und anhand der Kalenderdaten auf Stress oder Alkoholkonsum am Vortag untersucht werden.

3 Zusammenfassung

In dieser Arbeit wurde beschrieben, welche Veränderungen und Erweiterungen die Architekturplanung betrafen und wie die HTML Oberfläche des Spiegels an die Middleware angebunden wurde. Daran zeigte sich, dass die Datenerneuerung über die Messages der Middleware ein gangbarer Weg sind und für den Rest des Projekts weitergeführt werden können. Des Weiteren wurde ein erster Schritt für die Datenpersistenz unternommen und gezeigt, dass sich durch einfache Visualisierungen bereits Datenanalysen ergeben können. Um weitere Tests durchzuführen können die restlichen Daten ebenfalls für die InfluxDB Lösung vorbereitet und eingelesen werden.

4 Ausblick

In dieser Arbeit wurde ein Grundstein für die Plattform des Intelligenten Spiegels gelegt, welcher in der Masterarbeit weiterverfolgt werden kann.

Dabei wurde klar, dass es notwendig ist die Datenbestände weiter zu kontextualisieren und zu individualisieren um sie besser analysieren zu können. Das Individualisieren betrifft, wie in dieser Arbeit schon gezeigt, das Festlegen einer individuellen Norm anhand der vorhandenen Daten. In dieser Arbeit war es die Ermittlung eines ungewöhnlichen Blutdrucks, der bei diesen Daten an Punkten als ungewöhnlich gilt wo er laut WHO als völlig normal gilt. Es muss also für weitere Daten eine individuelle Norm gefunden werden, in der die Datenbestände eingeordnet werden können um erkennen zu können, wann Daten ungewöhnlich sind. Um diese als ungewöhnlich markierten Werte nach Korrelationen untersuchen zu können sollten die Daten weiter kontextualisiert werden. Hier ist dies durch Kalenderdaten geschehen, im nächsten Schritt sollte dies mit Wetterdaten angegangen werden.

Im Rahmen der Masterarbeit ist des Weiteren zu unterscheiden welche Form der Analyse für die Daten gewählt werden sollte. Es muss dabei ein statistisch analytisches oder ein auf Data Mining fundierendes Verfahren gewählt werden. Geeignete Werkzeuge dafür wurden hier bereits erwähnt.

Es wurde hier gezeigt, dass durch Betrachten der Daten mithilfe von einem gewissen Kontext, hier Kalenderdaten, Korrelationen erkannt werden können. Eine mögliche Fragestellung für die Masterarbeit könnte sein, ob sich dieses Erfassen und Analysieren bzw. Erkennen solcher Korrelate auf den vorliegenden Daten automatisieren lässt oder ob ein menschlicher Blick immer noch notwendig ist. Des Weiteren könnte untersucht werden, ob aus diesen Daten mithilfe der Kontextualisierung ein Mehrwert an Selbsterfahrung für den Nutzer generiert werden kann ohne ihn dem Stress der Optimierung auszusetzen, sondern eher um seine natürliche Neugier über sich selbst

zu unterstützen. Eine ebenso spannende Frage wäre es, welche Visualisierungen sich positiv auf die Erkenntnisse bei der Selbsterfahrung auswirken würden und welche eher weniger geeignet sind.

Literatur

[Eichler 2014] Eichler, Tobias: Agentenbasierte Middleware zur Entwicklerunterstützung in einem Smart-Home-Labor. (2014). – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/eichler.pdf

[Lüdemann 2016] Lüdemann, Maria: Quantifed Self nicht nur zum Selbstzweck. In: *Grundseminar Arbeit published an der HAW Hamburg* (2016). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2015-gsem/luedemann/bericht.pdf

[Lüdemann 2017a] Lüdemann, Maria: Der Intelligente Spiegel - Ein Companion zur Unterstützung der Selbstwahrnehmung Prototyp 1.0. In: *Grundprojekt Arbeit published an der HAW Hamburg* (2017). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2016-hsem/luedemann/bericht.pdf

[Lüdemann 2017b] Lüdemann, Maria: Der intelligente Spiegel- Ein Companion zur Unterstützung der Selbstwahrnehmung. In: *Hauptseminar Arbeit published an der HAW Hamburg* (2017). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2016-hsem/luedemann/bericht.pdf

[Tanenbaum und van Steen 2008] Tanenbaum, A.S. ; Steen, M. van: *Verteilte Systeme: Prinzipien und Paradigmen*. Pearson Studium, 2008 (It Informatik). – URL https://books.google.de/books?id=V6I6PQAACAAJ. – ISBN 9783827372932

5 Abbildungen

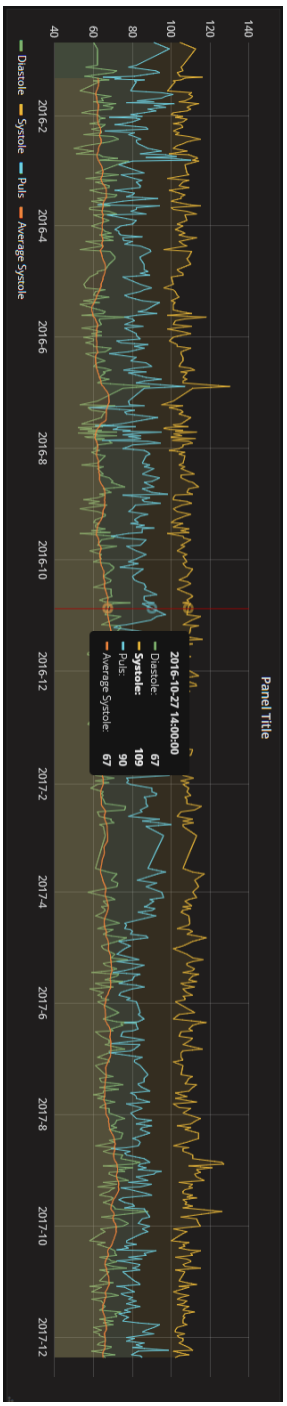

Abbildung 5: Visualisierungsbeispiel in der Grafana Oberfläche

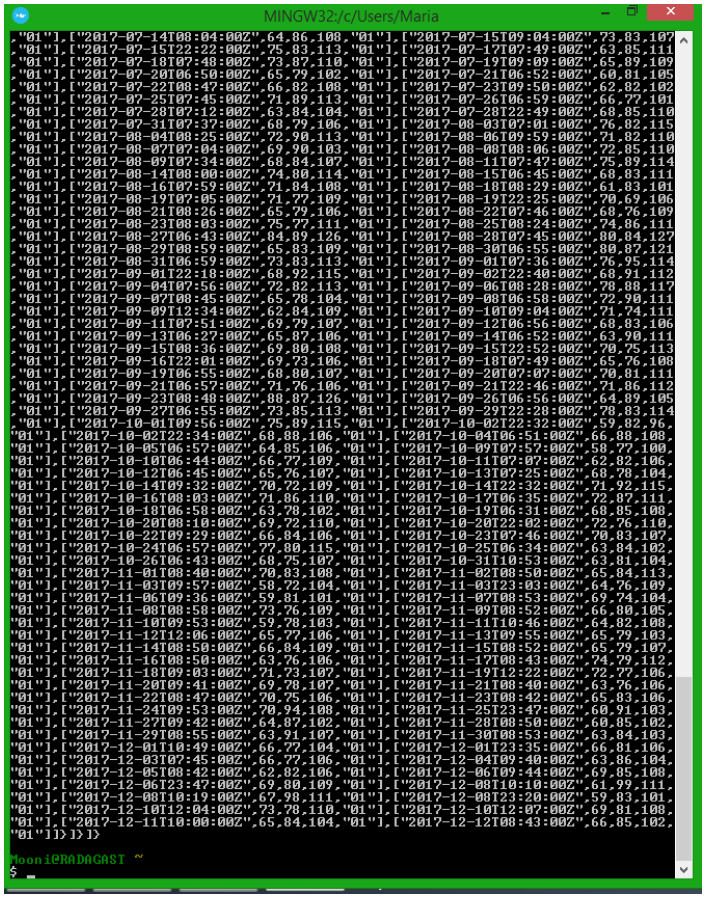

Abbildung 6: Abfrage aller Daten der Datebank

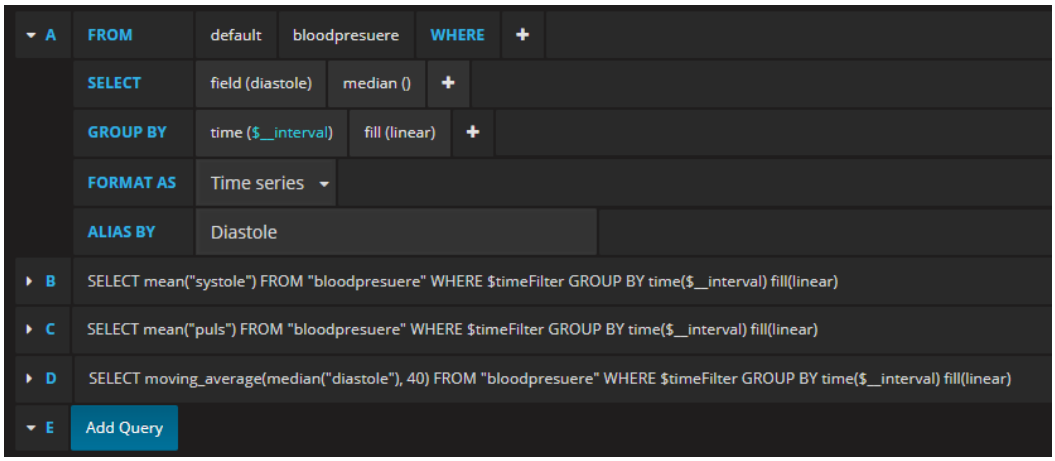

Abbildung 7: Abfrage Beispiel in der Grafana Oberfläche