HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ins institute for networked solutions

Bachelorarbeit, Abteilung Informatik

Internet of Things:

Management of Gadgeteer devices

Hochschule für Technik Rapperswil

Frühjahrssemester 2015
12. Juni 2015

Autoren:	Nathan Bourquin & Oliver Frischknecht
Betreuer:	Prof. Hansjörg Huser
Arbeitsperiode:	16.02.2015 - 12.06.2015
Arbeitsumfang:	720 Stunden, 12 ECTS pro Student

Abstract

In der heutigen Zeit gewinnt der Begriff „Internet of Things“ immer mehr an Wichtigkeit. Doch wie einfach ist es eine solche Umgebung in Betrieb zu nehmen und was braucht es alles um eigene Applikationen dafür zu entwickeln? „Internet of Things“-Systeme bestehen aus vielen verschiedenen Komponenten und unzähligen Technologien. Deshalb ist es eine Herausforderung alle vorhandenen Komponenten so aufeinander abzustimmen, dass ein kontrollierter Kommunikationsfluss entsteht. Zu den verwendeten Technologien gehören nebst dem „Lab of Things“-Framework und der Cloud Computing-Plattform Microsoft Azure, Hardware-Module von GHI Electronics. Sie erfüllen den .NET Gadgeteer-Standard (Microsoft) und gewährleisten somit die Kompatibilität mit dem .NET Micro Framework.

Zu den Modulen der Hardware Plattform gehören sowohl Aktoren, wie Display und LED, als auch verschiedenste Sensoren. Ein Mainboard verbindet die einzelnen Komponenten miteinander. Mit dem darauf laufenden .NET Micro Framework können die Hardware Module programmatisch ins System integriert werden. Auf dem Gerät werden Webschnittstellen eingerichtet, um die Aktoren zu steuern und die Sensordaten zu erhalten.

Das „Lab of Things“-Framework schreibt drei verschiedene Komponenten vor um Geräte einzubinden. Dazu gehören Scouts, Driver und Apps. Der Scout erkennt die Hardware durch das Erhalten von Broadcast Messages im Netzwerk. Die Driver greifen per HTTP auf die Webschnittstellen der Hardware-Plattform zu. So kann die Kommunikation zwischen den beiden Technologien realisiert werden. Die Applikation erhält die Daten von den Drivern. Sie besteht zusätzlich aus einer mit AngularJS und Bootstrap erstellten Webseite. Per WCF-Schnittstelle gelangen die Daten von der Applikation zur Webseite. Zur Datenanalyse werden die Microsoft-Services Event Hub und Stream Analytics eingesetzt. Dabei dient der Event Hub als Sammelbecken der vom Gadgeteer per AMQP gesendeten Sensordaten und ist die optimale Lösung für ein skalierbares System. Die vom Stream Analytics-Service ausgewerteten Daten wandern direkt in eine von Azure gehostete SQL-Datenbank, von wo sie dann von der Applikation zur Darstellung abgeholt werden können.

Das Resultat der Arbeit ist eine umfängliche „Internet of Things“-Umgebung. Sie ist vollständig mit dem Browser bedienbar und liefert sowohl statistische Auswertungen als auch aktuelle Messwerte. Sie zeigt die Möglichkeiten eines solchen Systems auf und ist einfach erweiterbar.

Inhaltsverzeichnis

I. Technischer Bericht 5

1. Management Summary 6
1.1. Ausgangslage . . . 6
1.2. Vorgehen, Technologien . . . 6
1.3. Ergebnisse . . . 6

2. Analyse 8
2.1. Lab of Things . . . 8
2.1.1. Installation . . . 8
2.1.2. Architektur . . . 9
2.2. .NET Gadgeteer . . . 10
2.2.1. Allgemein . . . 10
2.2.2. Hardware . . . 10
2.2.3. Requirements . . . 10
2.2.4. Installation . . . 10
2.2.5. Entwicklung . . . 11
2.2.6. Lab of Things Vorgabe . . . 11
2.3. Microsoft Azure . . . 12
2.3.1. Event Hub . . . 12
2.3.2. Stream Analytics . . . 13
2.4. Requirements . . . 15
2.4.1. Use Cases . . . 15
2.4.2. Non Functional Requirements . . . 16

3. Prototypen 18
3.1. Prototyp 1 - Lab of Things zu Gadgeteer . . . 18
3.1.1. Vorgaben . . . 18
3.1.2. Umsetzung . . . 19
3.1.3. Ablauf . . . 20
3.2. Prototyp 2 - Lab of Things zu Azure . . . 21
3.2.1. Lab of Things . . . 21
3.3. Prototyp 3 - Gadgeteer zu Azure . . . 23
3.3.1. Gadgeteer . . . 23
3.3.2. Azure . . . 24

4. Design 27
4.1. Komponenten . . . 27
4.2. Architektur . . . 28
4.2.1. Lab of Things . . . 28
4.2.2. Gadgeeteer . . . 29
4.3. Kommunikation . . . 30
4.3.1. Kommunikation zwischen Lab of Things und Gadgeteer . . . 30
4.3.2. Kommunikation Gadgeteer zu Azure Event Hub . . . 31
4.4. Sequenzdiagramme . . . 32
4.4.1. Gadgeteer zu Lab of Things . . . 32
4.4.2. Lab of Things . . . 32

4.5. Rules . . . 35
4.5.1. Grammatik . . . 35
4.5.2. Umsetzung . . . 35
4.6. Wireframes . . . 37
4.6.1. Übersicht . . . 37
4.6.2. Statistiken . . . 38
4.6.3. Kameras . . . 39
4.6.4. Bildschirme . . . 39
4.6.5. LEDs . . . 40
4.6.6. Regeln/Aktionen . . . 41
4.7. Testplanung . . . 42
4.7.1. LoT . . . 42
4.7.2. Gadgeteer . . . 43
4.7.3. Microsoft Azure . . . 44

5. Umsetzung **45**
5.1. Lab of Things . . . 45
5.1.1. Scout . . . 45
5.1.2. Drivers . . . 45
5.1.3. App . . . 47
5.1.4. Probleme . . . 50
5.2. Gadgeteer . . . 52
5.2.1. Setup . . . 52
5.2.2. DisplayWrapper . . . 52
5.2.3. ConnectionManager . . . 52
5.2.4. EventHubManager . . . 52
5.2.5. IdentifierSetting . . . 53
5.2.6. WebEventManager . . . 53
5.2.7. Probleme . . . 53
5.3. Azure . . . 55
5.3.1. Event Hub . . . 55
5.3.2. Stream Analytics . . . 55
5.3.3. SQL Database . . . 55

6. Abschluss **56**
6.1. Verbesserungen . . . 56
6.1.1. Rules . . . 56
6.1.2. GadgeteerResponse-Klasse anpassen . . . 56
6.1.3. Temperatur-Driver aufspalten . . . 56
6.1.4. Gadgeteer Architektur überarbeiten . . . 57
6.1.5. Konsequente Verwendung von DeviceIDs . . . 57
6.2. Fazit . . . 58

Teil I.

Technischer Bericht

1. Management Summary

1.1. Ausgangslage

Die aufkommende Integration unserer Haushaltsgeräte ins Netz ist ein weiterer Schritt in die Zukunft. Die Möglichkeit, dass auch Maschinen untereinander kommunizieren können, ermöglicht eine gute Koordination der Geräte und Komponenten im Haushalt. In jedem System braucht jemand die Kontrolle, damit alles geregelt abläuft. Solch eine Administrations-Software ist bereits in verschiedenen Projekten umgesetzt worden, dazu gehören zum Beispiel „openHAB“ und „Lab of Things“. Das letztere Framework soll verwendet werden und unter Einsatz der HW-Plattform .NET Gadgeteer soll ein System entwickelt werden, welches Microsoft Azure als Cloud Plattform benutzt. Zudem soll in dieser Arbeit ein Show-Case ausgearbeitet werden.

1.2. Vorgehen, Technologien

Die vorgegebenen Technologien werden in einem ersten Schritt genauer unter die Lupe genommen, damit Architekturentscheide und Risiken besser abgeschätzt werden können. Dazu gehören einerseits „Microsoft Azure“ und „Lab of Things“, welches auf dem .NET Framework basiert, und andererseits auch „Gadgeteer“, welches eine HW-Plattform mit Sensoren (Temperatur, Luftfeuchtigkeit, Lichteinstrahlung) und Aktoren (Kamera, LED) ist. Auf dem Microcontroller wird das .NET Micro Framework ausgeführt. Auf dem Controller wird ein Webserver eingerichtet mit Schnittstellen für das Messen der Sensorwerte und das Steuern der Aktoren. Für „Lab of Things“ wird ein Scout geschrieben, welcher für die Erkennung des Gadgeteer-Gerät zuständig ist. Zusätzlich werden Driver geschrieben, welche über HTTP mit dem Webserver auf dem Gadgeteer-Gerät kommunizieren und den Datenaustausch zwischen den beiden Technologien sicherstellen. Eine Applikation nimmt die von den Drivern erhaltenen Daten entgegen und speichert sie im Memory. Wie der Scout und die Driver ist auch die Applikation in „Lab of Things“ eingebunden. Zusätzlich beinhaltet die Applikation eine Webseite basierend auf HTML und AngularJS. Sie stellt die Daten benutzerfreundlich dar. Über eine WCF-Schnittstelle erfolgt der Zugriff auf die von der Applikation gelieferten Messwerte. Die Cloud Computing-Plattform Azure wird für eine statistische Auswertung verwendet. Deshalb werden die vom Gadgeteer gemessenen Sensordaten zusätlich mittels dem Advanced Message Queuing Protocol (AMQP)[9] an den Event Hub gesendet. Dieser leitet die Daten an den Stream Analytics-Service weiter, welcher eine Datenanalyse in Echtzeit durchführt und die Ergebnisse in einer SQL-Datenbank speichert. In einem letzten Schritt greift die Applikation auf die erwähnte Datenbank zu und stellt die erhaltene Auswertung ebenfalls über die WCF-Schnittstelle der Webseite zur Verfügung.

1.3. Ergebnisse

Aus dieser Bachelorarbeit sind verschiedene Ergebnisse entstanden. Für die Bedienseite ist ein ansprechendes, flaches Web-Interface entwickelt worden. Alle Sensoren und Aktoren lassen sich damit anschauen und bedienen. Es ist in „Lab of Things“ integriert und bietet somit eine gute

Erweiterbarkeit. Sensoren und Aktoren können mit dem System dynamisch hinzugefügt und wieder entfernt werden. Zudem sind Driver entwickelt worden, welche zum Datenaustausch zwischen den Systemkomponenten dienen. Ausserdem ist ein Scout geschrieben worden, welcher das Netz nach Gadgeteer-Komponente durchsucht und diese im System registriert.

Auf der Prozess-Seite ist ein interessantes Produkt entstanden. Zum einen stellt der Controller eine Verbindung zu Azure her und schickt zur Datenanalyse in regelmässigem Zeitabstand seine Sensorwerte. Zum anderen ist eine Webschnittstelle eingerichtet worden, welche es den vorher erwähnten Drivern ermöglicht, das Gerät zu steuern und die Sensorwerte abzufragen.

2. Analyse

2.1. Lab of Things

Lab of Things (LoT) [12] ist ein Framework, welches hauptsächlich für die experimentelle Forschung geschaffen wurde. Wie im Name schon angedeutet wird es benutzt, um eine „Internet of Things“-Umgebung zu realisieren. LoT macht es möglich alle Geräte durch die Definition gewisser Schnittstellen in einem einzigen System zu vereinen. Dieses System erleichtert die Analyse und Überwachung verbundener Geräte.

Die Clientseite von LoT kann auf jedem Windows PC installiert werden, eine solche Instanz wird HomeHub genannt. Alle Sensoren und Geräte werden mit dem HomeHub verbunden. In Form von Apps werden die erhaltenen Gerätedaten je nach Anwendungsbereich analysiert, verarbeitet und dem Benutzer dargestellt.

Die Serverseite hingegen besteht aus einer Anzahl von Cloud-Diensten, welche auf der Cloud Computing-Plattform Azure laufen. Dort können die Daten, die von der Clientseite her kommen gespeichert, verändert und überwacht werden.

2.1.1. Installation

Die Installation der Entwicklungsumgebung ist ziemlich einfach und braucht nur ein paar Schritte.

Zum einen müssen folgende Sachen installiert sein oder werden: [7]

- Windows (Ab Version 7)
- Microsoft .NET Framework 4.5
- Visual Studio 2012 oder Visual Studio 2013
- Microsoft Sync Framework 2.1 Redistributable Package x86 (Synchronization und Provider Services)
- Windows Azure SDK 2.3

In den nächsten Schritten wird gezeigt, wie die Umgebung installiert und konfiguriert wird. Zunächst muss der Quellcode bezogen werden. Für Git-Benutzer kann das Projekt direkt von der URL `https://labofthings.codeplex.com/SourceControl/latest` geklont werden. Ansonsten kann auch per Browser unter der selben Addresse der Code heruntergeladen werden. Mit dem Visual Studio kann dann die Core Solution, welche im Hub-Ordner zu finden ist, geöffnet und das Projekt erstellt werden.

Ist das Projekt erstellt, so kann die Plattform entweder über die im Hub\output-Ordner enthaltene Datei „startplatform.bat“ oder direkt im Visual Studio über die Schaltfläche „Starten“ gestartet werden. Die Plattform kann nun per Browser unter der Adresse `http://localhost:51430/guiweb/` bedient werden.

2.1.2. Architektur

LoT ist in mehreren Teile gegliedert. Die Pakete „Platform“, „Common“ und „DashboardWeb“, kümmern sich um die Kernfunktionalität des Systems und sind nicht dafür gedacht, verändert zu werden. Andere Pakete wie Scouts, Drivers und Apps sind so aufgebaut, dass neue Projekte hinzugefügt werden können, um so die Plattform zu erweitern. Die folgenden Absätze erklären diese Pakete genauer. [11]

2.1.2.1. Scout

Die Scouts sind dafür zuständig, Geräte im Netz automatisch zu finden und ins laufende System zu integrieren. Zudem hosten sie die Konfigurationsseite des betreffenden Gerätes.

Für jedes unterschiedliche Gerät muss ein Projekt im Scouts-Ordner angelegt werden. Ein Scout besteht in der Regel aus zwei Klassen. Eine Scout-Klasse, die für das eigentliche Erkennen zuständig ist und eine Service-Klasse. Sie verknüpft die Konfigurationsseite mit der Scout-Klasse.

Die Scout-Klasse implementiert das „IScout“-Interface. Dieses schreibt die folgenden Methoden vor:

Init Initialisiert eine Scout Instanz.

Dispose Räumt die Instanz auf.

GetDevices Hier wird nach den betreffenden Geräten gesucht und eine Liste davon zurückgegeben.

2.1.2.2. Driver

Driver sind das Mittel, um die Hardwarekomponente mit dem System zu verbinden. Die Start-Methode des Drivers wird vom System aufgerufen, sobald ein neues Gerät hinzugefügt wird. Sie instanziiert alle benötigten Klassen, legt mittels Roles fest, welche Funktionalitäten von ihm angeboten werden und instanziiert ein Port-Objekt, welches das hinzugefügte Gerät eindeutig erkennt. Zusammen mit den Roles wird der Port im System registriert, so dass Applikationen davon Gebrauch machen können.

Die Hauptaufgabe eines Drivers wird in der „Work()„-Methode festgelegt. Diese wird ebenfalls von der Start-Methode gestartet, jedoch als separater Thread.

Mittels der Stop()-Methode wird der Worker-Thread und somit der Driver vom System gestoppt.

2.1.2.3. Apps

Bei Apps handelt es sich einerseits um Webseiten, die Gerätedaten analysieren und darstellen. Andererseits um Softwarekomponenten, welche die Daten von den Drivern erhalten und diese über eine WCF-Schnittstelle der Webseite anbieten.

2.2. .NET Gadgeteer

2.2.1. Allgemein

Die .NET Gadgeteer-Plattform kombiniert das Zusammensetzen elektronischer Geräte ohne dabei zu löten und das objektorientierte Programmieren dieser Geräte mit Hilfe des .NET Micro-Framework.

2.2.2. Hardware

Die Gadgeteer-Hardware baut auf ein Mainboard mit integrierter CPU auf. Es können dann je nach Belieben Module über Sockets mit dem Mainboard verbunden werden, um das System zu erweitern. Zu den gängigen Modulen gehören Displays, Kameras, Netzwerkschnittstellen, Datenspeicher, Sensoren und andere Eingabegeräte. GHI Electronics[8] gilt dabei als marktführender Hersteller dieser Hardware.

2.2.2.1. Sockets

Das Gadgeteer-Mainboard hat verschiedene Sockets, welche mit einem oder mehreren Buchstaben versehen sind. Diese Buchstaben zeigen auf, welche Module daran angehängt werden können. Es muss also darauf geachtet werden, dass ein Modul an dem dafür vorgesehenen Socket angebunden wird.[14]

2.2.3. Requirements

Um mit der Gadgeteer-Plattform zu arbeiten, werden diese verschiedenen Komponenten benötigt:

Hardware
Die physischen Komponenten, welche das .NET Micro-Framework und jegliche Anwendungen ausführen.

Software Tools
Für die Entwicklung wird das Visual Studio als Entwicklungsumgebung verwendet. Zudem wird das .NET Micro-Framework SDK benötigt, um Anwendungen zu schreiben, deployen, und die Applikationen auf der Hardware zu debuggen.

Board Spezifische Software
Bibliotheken und Drivers für die spezifischen Hardware Komponenten.

2.2.4. Installation

Folgende Schritte müssen auf dem Computer durchlaufen werden, um mit dem .NET Micro-Framework arbeiten zu können. [15]

1. Microsoft Visual Studio 2013 installieren
2. Alle alten Versionen vom .NET Micro-Framework SDK deinstallieren
3. Alle .NET Micro-Framework third party SDK Software deinstallieren
4. .NET Micro-Framework herunterladen, entpacken und dann die Datei „MicroFrameworkSDK.MSI“ ausführen
5. .NET Micro-Framework VSIX Erweiterungen installieren

a) In den entpackten Dateien aus Schritt 4 „netmfvs2013.vsix“ ausführen

6. .NET Gadgeteer Core installieren

7. NETMF und Gadgeteer Package installieren

2.2.5. Entwicklung

Die Entwicklung mit Gadgeteer ist ziemlich einfach. Nach der Installation des Frameworks kann im Visual Studio unter Angabe des verwendeten Mainboards ein Gadgeteer-Projekt erstellt werden. In einem frisch eingerichteten Gadgeteer-Programm sind mehrere Komponenten enthalten.

Zum einen ist ein Designer enthalten. Mit ihm kann der Entwickler die Verbindungen der einzelnen Module konfigurieren, damit die einzelnen Hardwareteile richtig initialisiert werden können. Um dies zu machen, muss der Entwickler aus einer Liste von Gadgeteer-Hardwareteilen die richtigen Teile auswählen und dann im Designer die richtigen Verbindungen einstellen. Mittels des Designers können die Module auch benannt werden. Diese Namen können dann im Code direkt verwendet werden. Der Designer generiert im Hintergrund eine Softwareklasse und ininiialisiert dort die Module als Fields mit den eingestellten Namen.

Zum anderen gibt es die Programm-Klasse. Sie ist in zwei partielle Klassen aufgeteilt. Die eine Hälfte davon wird vom Designer generiert. Sie wird vom Gadgeteer beim Systemstart ausgeführt und ruft die ProgramStarted()-Methode in der nicht generierten Hälfte der partiellen Klasse auf. Durch ihre Abhängigkeit können auf die generierten Instanzvariablen von beiden Teilen her zugegriffen werden.

2.2.6. Lab of Things Vorgabe

Im LoT-Paket [10] ist bereits ein Gadgeteer-Projekt vorhanden, das darauf ausgelegt ist, Gadgeteer-Hardware ins Framework zu integrieren. Es ist in der .NET Framework-Version 4.2 geschrieben und baut eine Netzwerkverbindung via Wireless Adapter auf. Im Kapitel 3.1 wird genauer darauf eingegangen.

2.3. Microsoft Azure

Azure[13] ist eine von Microsoft entwickelte Cloud Computing-Platform, welche sowohl PaaS (Platform as a Service) als auch IaaS (Infrastructure as a Service) anbietet. Dabei werden zahlreiche Programmiersprachen und Frameworks unterstützt. Azure läuft auf einem gleichnamigen spezialisierten Betriebssystem. So werden Services wie Datenbanken, Hostings für Webapplikationen, virtuelle Maschinen und Cloud Services ermöglicht. Dazu gehören die in den folgenden Kapiteln beschriebenen Services Event Hub[3] und Stream Analytics[18].

2.3.1. Event Hub[4]

2.3.1.1. Einstieg

Immer mehr Anwendungen sind darauf angewiesen, Ereignisse von mehreren Geräten, welche sich an unterschiedlichsten Orten befinden, übers Internet gesammelt und zu verarbeitet zu werden. Die Herausforderung besteht darin, grosse Datenmengen sicher und zuverlässig aufzunehmen. Der Event Hub-Service wurde speziell dafür entwickelt, um diese Probleme anzupacken. Folgende Begriffe sind fürs Verständnis der weiteren Kapitel notwendig.

Event Publisher
: Eine Komponente, welche ein Ereignis auslöst und dieses dann in Form einer Nachricht an den Event-Hub schickt.

Event Consumer
: Eine Komponente, welche auf die Ereignisse der Event-Publisher wartet und diese dann verarbeitet.

2.3.1.2. Funktionsweise

Alle Event-Publisher senden ihre Daten via HTTP oder AMQP auf einen zentralen Service-Bus. Dabei gilt das FIFO-Prinzip. Bei den beiden Diensten Service Bus-Topics und Service Bus-Queues streiten nun die Event Consumer darum, wer den nächsten Event vom Bus lesen und verarbeiten darf. Da nicht zwei Event Consumer den gleichen Event verarbeiten können. Beim Event Hub wird hingegen das „Partitioned Consumer“-Pattern eingesetzt. Der ganze Bus wird in Partitionen unterteilt, so dass ein Event-Consumer nur auf eine Teilmenge der im Bus enthaltenen Events hört. So können Streitigkeiten umgangen werden, wer welche Nachricht verarbeiten darf.

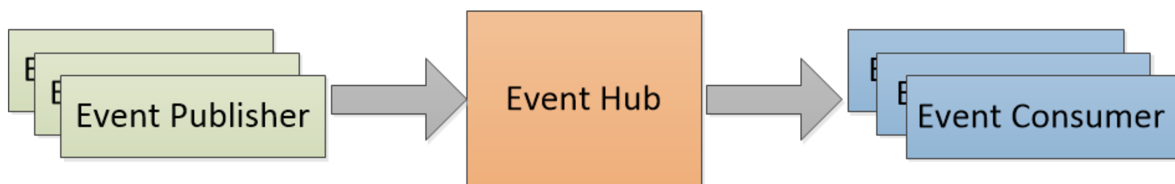


Abbildung 2.1.: Azure Event Hub: Event Publisher/Consumer Übersicht

2.3.1.2.1. Partitionen Eine Partition ist eine geordnete Sequenz von Ereignissen. Man kann sich eine Partition wie eine Warteschlange an der Kasse eines Supermarktes vorstellen. Der Kunde, der zuvorderst in der Warteschlange steht, wird auch zuerst bedient. Neue Ereignisse stellen sich also hinten in die Warteschlange. Solche in der Warteschlange müssen innerhalb einer bestimmten Zeit abgeholt werden, sonst werden sie automatisch vom System verworfen. Ein Event Hub hat zwischen 8 und 32 solcher Warteschlangen.

2.3.1.2.2. Consumer Group Vorgängig wurde beschrieben, dass das „Partitioned Consumer"-Pattern dazu führt, dass ein Event Consumer nur auf ein Subset des ganzen Busses hört. So ist es also nicht möglich, dass zwei Event Consumer die gleichen Ereignisse verarbeiten. Damit auch diese Funktionalität erreichbar ist, kann man Consumer-Groups erstellen. Jede Gruppe bekommt eine Kopie jedes Events. Per Default sind alle Event Consumer der Gruppe „$Default" angeschlossen. Es können jedoch bis zu 20 solcher Gruppen pro Event Hub erstellt werden.

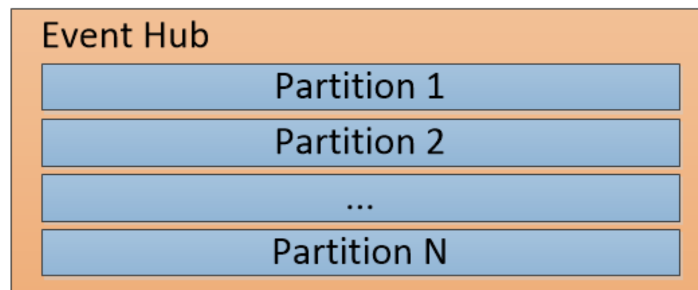


Abbildung 2.2.: Azure Event Hub: Partitionen

2.3.1.2.3. Partition Key Damit Events, die vom gleichen Event-Publisher kommen, in die gleiche Partition eingereiht werden, kann beim Sender der Message ein Partition Key angegeben werden. Dieser wird dann mittels eines von Azure implementierten Hashing Algorithmus einer Partition zugeordnet.

2.3.2. Stream Analytics[6]

Stream Analytics gehört zu den „Data Services" von Microsoft Azure und ist zurzeit als Preview verfügbar. Der Service bietet die Möglichkeit, einen Ereignisstrom zur Laufzeit zu überwachen und auszuwerten. Dabei können auch mehrere Ereignisströme mit einander verglichen werden. Der Service ist darauf spezialisiert, mit einer grossen Datenmenge zu operieren und bietet einen direkten Anschluss an den Azure Event Hub.

2.3.2.1. Requirements

Um den Service nutzen zu können, müssen folgende Komponenten zur Verfügung stehen:

1. Service Bus Namespace
2. Event Hub
3. Storage Acount
4. SQL Database

2.3.2.2. Funktionsweise

Der Stream Analytics-Service operiert auf den ganzen Daten des Event-Hubs. Er braucht also eine eigene Kopie aller Events, welche an den Event-Hub geschickt werden. Somit ist der Stream Analytics-Service ein Event-Consumer in einer separaten Consumer-Group. Die Event Publisher senden also ihre Events an den Event Hub. Dieser kopiert die Events auf alle Consumer-Groups. Zur Verarbeitung der Events in einer Consumer-Group wird ein Event-Consumer benötigt. In diesem Kapitel ist dies der Stream Analytics-Service. Dieser kann mehrere Ereignisströme konsumieren, welche konfiguriert und benannt werden müssen.

Zur Analyse der Events wird ein Query verwendet. So kann zum Beispiel der Stream Analytics-Server die „DeviceId„ und den Lux Wert aus dem Event herauslesen und in eine Datenbank schreiben. Im folgenden Beispiel ist der Ereignisstrom des Event Hubs mit dem Namen „input“ konfiguriert. Hier wäre die Beispielsabfrage:

```
1 SELECT DeviceId, Lux from input;
```

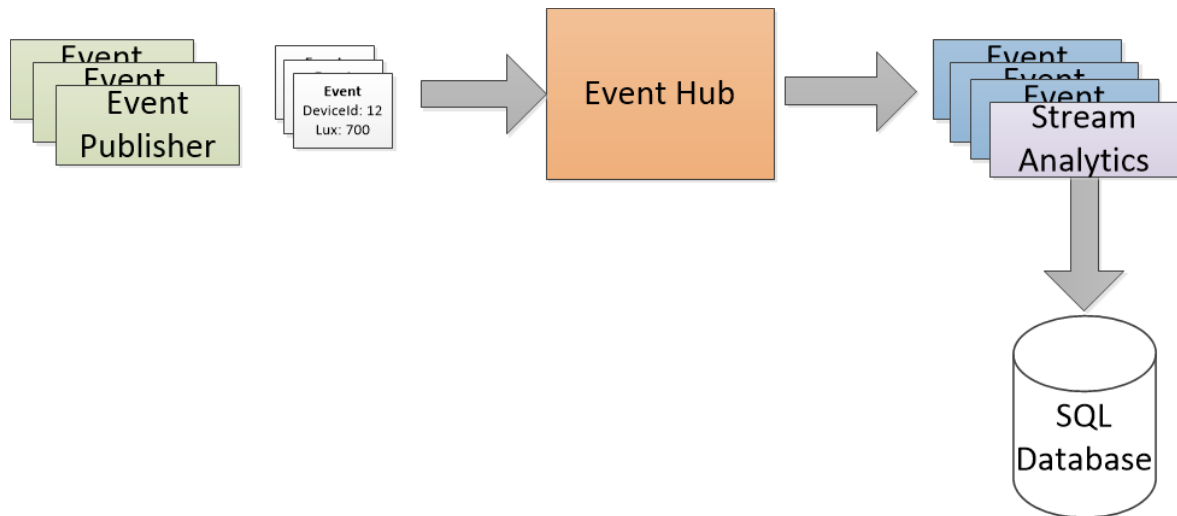


Abbildung 2.3.: Azure Stream Analytics: Übersicht

2.4. Requirements

2.4.1. Use Cases

Es sollen durch verschiedene Use Cases die zu entwickelnde LoT-Applikation beschrieben werden. Diese werden im unten dargestellten Use Case-Diagramm angezeigt. Mit der Anwendung sollen hauptsächlich Sensordaten und deren Auswertung dargestellt werden. Zusätzlich sollen gewisse Komponenten der Gadgeteer-Plattform damit gesteuert werden.

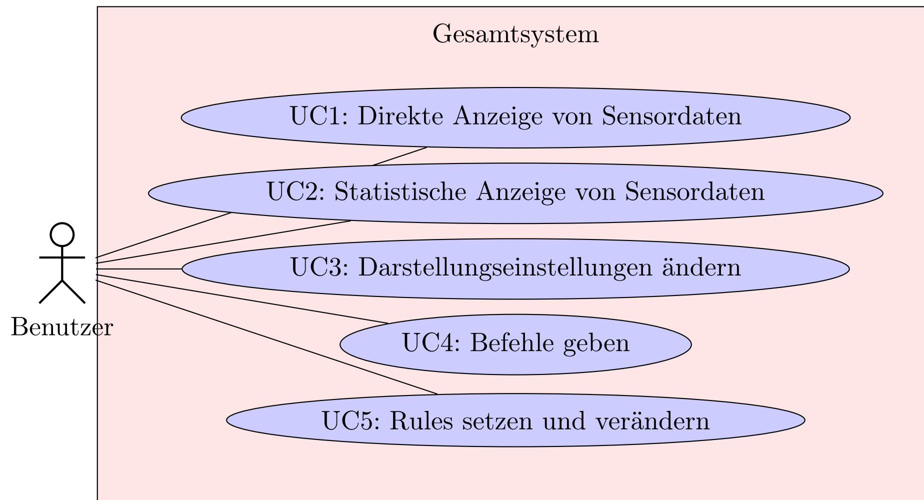


Abbildung 2.4.: UML Diagramm: Show cases

Diese fünf Use-Cases zeigen alles, was in der zu erstellenden App ermöglicht werden soll. Sie werden in folgender Tabelle genauer beschrieben.

Nr.	Name	Ziel	Primary Actor
UC1	Direkte Anzeige von Sensordaten	Sensordaten in Realtime anzeigen. Zum Beispiel wird bei einem Temperatursensor die aktuelle Temperatur angezeigt. Das aktuelle Kamerabild gehört zwar auch dazu, muss aber nicht regelmässig geladen werden.	User
UC2	Statistische Anzeige von Sensordaten	Gespeicherte Sensordaten werden statistisch und graphisch dargestellt.	User
UC3	Darstellungseinstellungen ändern	Für die graphische Darstellung der Sensordaten kann eingestellt werden, über welche Zeitperiode die Daten angezeigt werden sollen.	User
UC4	Befehle geben	Verschiedene Befehle wie LED einschalten, umschalten, den Bildschirm steuern können gegeben werden.	User
UC5	Rules setzen und verändern	Eine Rule besteht aus einem Trigger (Sensorwert verglichen mit einem eingestellten Wert) und einer Aktion. Die Aktionen sind die Befehle aus UC4.	User

Tabelle 2.1.: Use Case Übersicht

2.4.2. Non Functional Requirements

Die folgenden NFRs beinhalten Anforderungen an das Endprodukt, die nicht funktionell sind. Sie werden nach Themengebieten aufgeteilt.

2.4.2.1. Benutzbarkeit

1. Das Design der LoT-Anwendungen ist so aufzubauen, dass es für den Benutzer möglich ist, mit maximal 3 Mausklicks auf jede Funktionalität zugreifen zu können.

2.4.2.2. Skalierbarkeit

1. Die Architektur muss nicht für mehrere LoT-Instanzen geplant werden, sondern lediglich für eine LoT Instanz und einen Azure-Account.

2.4.2.3. Effizienz

1. Die Ladezeit der Webcam-Bilder soll nicht länger als 5 Sekunden dauern. Dafür kann die Bildqualität wenn nötig verringert werden.
2. Sensordaten, die in „real-time“ angezeigt werden sollen, dürfen nicht mehr als 4 Sekunden vom Sensor bis zur Anzeige benötigen.

2.4.2.4. Sicherheit

1. Sichere Verbindungen beim Datenaustausch werden nicht gewährleistet, da es sich um eine experimentelle Anwendung handelt.

2.4.2.5. Portierbarkeit

1. Die Apps werden spezifisch für LoT entwickelt.
2. Der Programmcode für die Gadgeteer-Komponente ist hardwarespezifisch und auch dort wird keine Portabilität unterstützt.

2.4.2.6. Internationalisierung

1. Die Benutzeroberfläche der Apps wird ausschliesslich in deutscher Sprache zur Verfügung stehen.

3. Prototypen

3.1. Prototyp 1 - Lab of Things zu Gadgeteer

Im ersten Prototyp werden die Verbindungen zwischen den Sensoren, Detektoren und Lampen auf dem Gadgeteer und dem LoT-Framework getestet. Das Ziel des Ganzen ist es, einen Durchbruch zu erzielen und verschiedene Sensordaten live auf der Oberfläche einer LoT-App anzuzeigen.

3.1.1. Vorgaben

Im LoT Source-Code sind schon bestimmte Teile des Prototyps vorhanden. Dabei handelt es sich um Scouts und Drivers. Für andere Geräte existieren bereits fertige Implementationen von Scouts und Drivers, welche gut als Vorlage gebraucht werden konnten. Wichtig war es diese gut verstehen zu können, um sie den vorhandenen Bedürfnissen anpassen zu können.

3.1.1.1. Gadgeteer

Die Vorgaben im Gadgeteer bestanden einerseits aus einem Webserver, der Sensordaten gegen Abfrage verschickt, und andererseits aus einer Manager-Klasse, welche mittels Wireless-Lan die Netzwerkverbindung aufbaut. Die Vorgaben waren in der .NETMF Version 4.2 geschrieben. Das System wurde so umgeschrieben, dass es mit der aktuellen Version 4.3 kompatibel ist und damit das Gerät mit dem verwendeten Ethernet Adapter funktioniert. In der oben erwähnten Manager-Klasse war ein Thread vorhanden, der regelmässig Beacons versendet, dieser konnte eins zu eins übernommen werden. Aus dem Grund, dass die LoT-Scouts das Gadgeteer-Gexerät auch ohne das explizite Setzen einer IP Adresse erkennen sollen, werden von Zeit zu Zeit sogenannte Beacons verschickt. Ein Beacon enthält eine Zeichenkette, die zum aktuellen Zeitpunkt einen bestimmten Sensor kennzeichnet. In Zukunft sollte jedoch jeder Sensor, der am Gadgeteer angehängt ist, einzeln identifiziert werden können.

3.1.1.2. Lab Of Things

Damit ein Gerät in die LoT-Umgebung integriert werden kann, müssen drei verschiede Komponenten definiert werden.

Zum einen wurde ein Scout entwickelt, der die Aufgabe hat, Gadgeteer Geräte zu finden. Der Scout sucht nach den vorhin beschriebenen Beacons. Findet der Scout ein solches Beacon, dann instanziiert er mit der Kennzeichnung des Gadgeteers ein Device. Dieses Device wird im passenden Driver weiterverwendet.

Der vorgegebene Driver wurde verwendet, um die vom Scout deklarierten Devices zu konfigurieren. Im Driver geschieht der Datenaustausch zwischen Hardware und Plattform. Zum Beispiel kann unter Angabe der Ip-Addresse und des Pfades /webcam/ via HTTP-Request ein Webcambild vom Gadgeteer geholt werden. Das Bild wird vom Driver allen auf der Camera Role registrierten Applikationen zur Verfügung gestellt.

Als App steht eine Webcam-Anzeige-Applikation zur Verfügung. Diese holt die im Driver stehenden Daten mithilfe der zuvor erwähnten Camera-Role und stellt sie graphisch dar.

3.1.2. Umsetzung

3.1.2.1. Gadgeteer

Im Gadgeteer-Teil wurde die gesamte Temperatur-Funktionalität hinzugefügt. Dabei wurden der Temperatursensor instanziiert und, damit die Temperatur per HTTP abgefragt werden kann, die Services angepasst. Zu guter letzt musste wurde die im Beacon enthaltene Zeichenkette so verändert, dass das Gadgeteer als Temperatursensor angesehen wird und nicht mehr als Webcam.

3.1.2.2. Lab of Things

Die zuvor erwähnte Webcam Anzeige-App wurde angepasst, um die Messwerte eines Temperatursensors anzuzeigen. Ziel war es, durch die Änderungen besser zu sehen, wie das ganze System genau funktioniert.

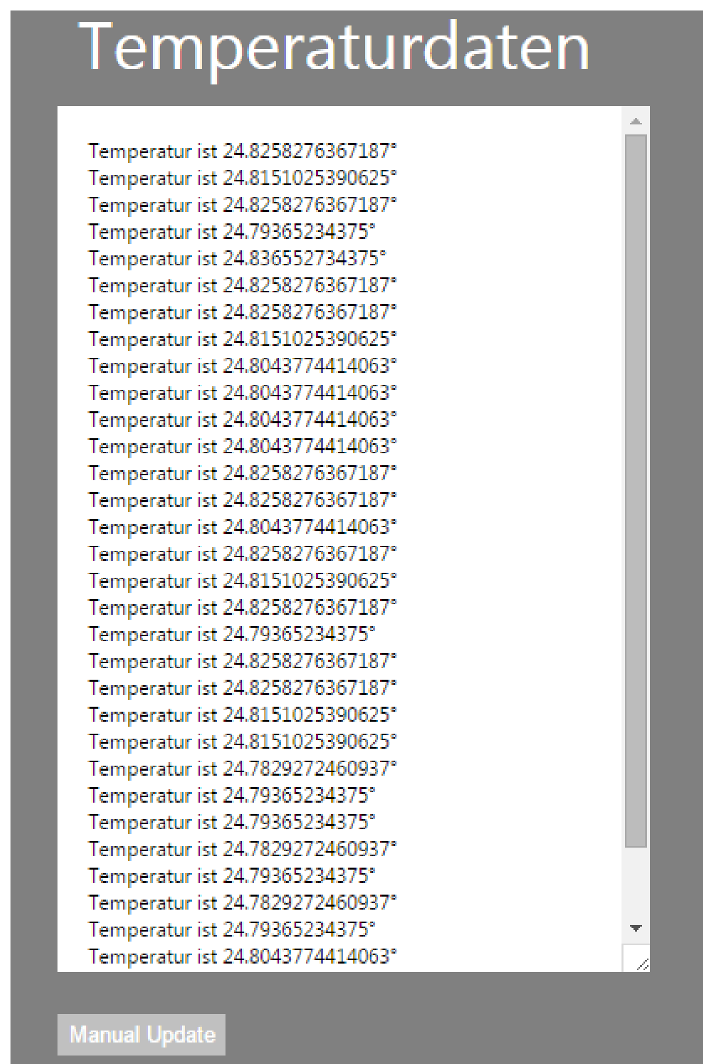


Abbildung 3.1.: Screenshot: Anzeige der Temperaturen

3.1.3. Ablauf

3.1.3.1. Gadgeteer erkennen

Im unteren Sequenzdiagramm ist zu sehen, wie Gadgeteer-Geräte in der aktuelle Prototyp-Version vom Scout erkannt und als Device abgespeichert werden.

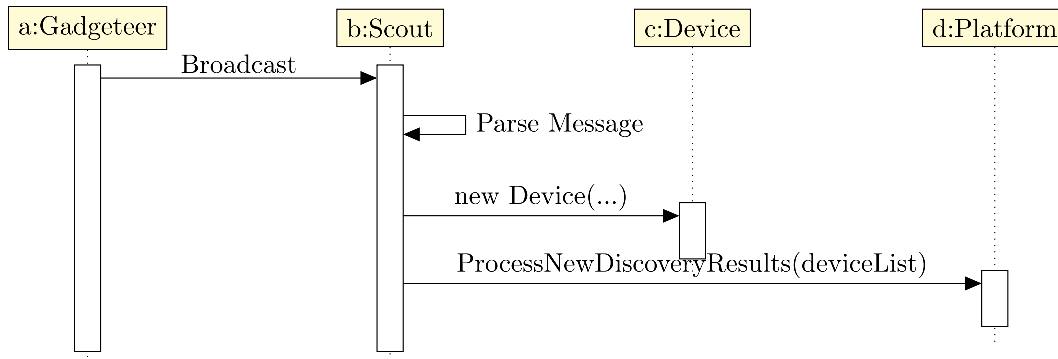


Abbildung 3.2.: Sequenzdiagramm: Gadgeteer erkennen

In einem ersten Schritt verschickt das Gadgeteer in regelmässigen Zeitabständen Beacons. Bisher identifizierte ein Beacon nur einen bestimmten Sensor.

Das Beacon wird vom Scout erkannt und geparsed. Aus den gewonnenen Daten wird dann ein neues Device erstellt. Bei diesem Schritt ist es wichtig, dass das angemeldete Device den Namen des dafür zuständigen Drivers enthält.

Das neu erstellte Device wird nun einer Liste mit Devices hinzugefügt. Diese hat in diesem Fall immer maximal einen Eintrag. Die Liste wird der Plattform übergeben. Die Plattform startet dann den gewünschten Driver anhand der im Device angegebenen Informationen.

3.2. Prototyp 2 - Lab of Things zu Azure

In diesem Prototyp wird eine Applikation für LoT entwickelt, welche ausgewertete Sensordaten von einer Azure SQL-Datenbank liest, das Ziel ist es die Verbindung zwischen LoT und Azure zu testen. Dazu wird die im Prototyp 3 erstellte Datenbank als Datenbasis verwendet. Die Struktur von LoT kann im Kapitel 2.1 nachvollzogen werden.

3.2.1. Lab of Things

Anhand des Beispielcodes, welcher bereits im LoT-Framework vorhanden ist, konnte die Applikation geschrieben werden.

3.2.1.1. Aufbau der Applikation

Die Applikation implementiert die abstrakte Basisklasse ModuleBase. Somit müssen folgende Methoden überschrieben werden:

- Start()
- Stop()
- PortRegistered()
- PortDeregistered()

Die Start-Methode initialisiert einen Webserver, auf dem das User Interface, eine einfache HTML Seite, gehostet wird. Es wird ein ServiceHost gestartet, um die Kommunikation zwischen User Interface und Business-Logik herzustellen und die Verbindung zur Datenbank wird initialisiert. Mehr dazu in Kapitel 3.2.1.2. Die Methode wird sowohl beim Aufstarten der Plattform als auch direkt nach der Installation der Applikation aufgerufen.

Die Stop-Methode schliesst alle gestarteten Services und wird bei der Deinstallation der Applikation aufgerufen.

Die Methoden PortRegistered und PortDeregistered werden benötigt, um die Kommunikation mit den Drivern herzustellen. In diesem Prototyp werden sie nicht benötigt.

Fehlt nur noch die Methode zur Abwicklung der Businesslogik. GetSensorData enthält die SQL-Abfrage, welche die Daten aus der Datenbank liest. Und daraus eine Liste von MeasurementEntities erstellt. MeasurementEntity ist die Modellklasse, welche der Struktur aus der Datenbank entspricht.

3.2.1.2. Verbindung zur Datenbank

Um die Verbindung zur Datenbank herzustellen, wird ein Connection-String benötigt. Dieser setzt sich aus dem Datenbankserver, dem Benutzer, dem Passwort und dem Datenbanknamen zusammen. In der App.config wurde eine Sektion mit dieser Information hinterlegt. Hier das Beispiel dazu:

```
<connectionStrings>
    <add name="ConnectionString"
    connectionString="Server=tcp:SERVER_NAME.database.windows.
        net;Database=DATABASE_NAME;UserID=USER_NAME@SERVER_NAME;
        Password=PASSWORD;Trusted_Connection=False;Encrypt=True;"/
        >
</connectionStrings>
```

Mit Hilfe des SQLConnectionStringBuilder werden die benötigten Information gelesen und daraus die SQLConnection hergestellt.

3.2.1.3. User Interface

Der Code für die HTML Seite konnte praktisch 1:1 vom DummyApp übernommen werden. Für die Anzeige der Sensordaten wurde die Google Charts API verwendet. Diese musste im Code eingebunden werden und die Daten mussten richtig eingespeist werden. Die Webseite greift über die WCF-Schnittstelle vom ServiceHost auf die Sensordaten zu. Als Resultat kommt ein JSON-String mit einem darin enthaltenen JSON Array zurück. Mit JQuery wird der String geparsed und mit einer Schleife werden die Daten richtig aufbereitet, damit sie von der Chart Library benutzt werden können.

Die Daten aus der Datenbank sind die gemessenen Sensordaten der Gadgeteer-Plattform. In der folgenden Abbildung ist die entwickelte Applikation zu sehen, welche ein Diagramm von diesen Sensordaten anzeigt. Es handelt sich um die Lichtstärke, welche in Lux gemessen wurde. Im Verlauf der Messung wurde der Lichtsensor temporär ein wenig abgedeckt und man sieht sehr schön den daraus entstandenen Negativ-Peak.

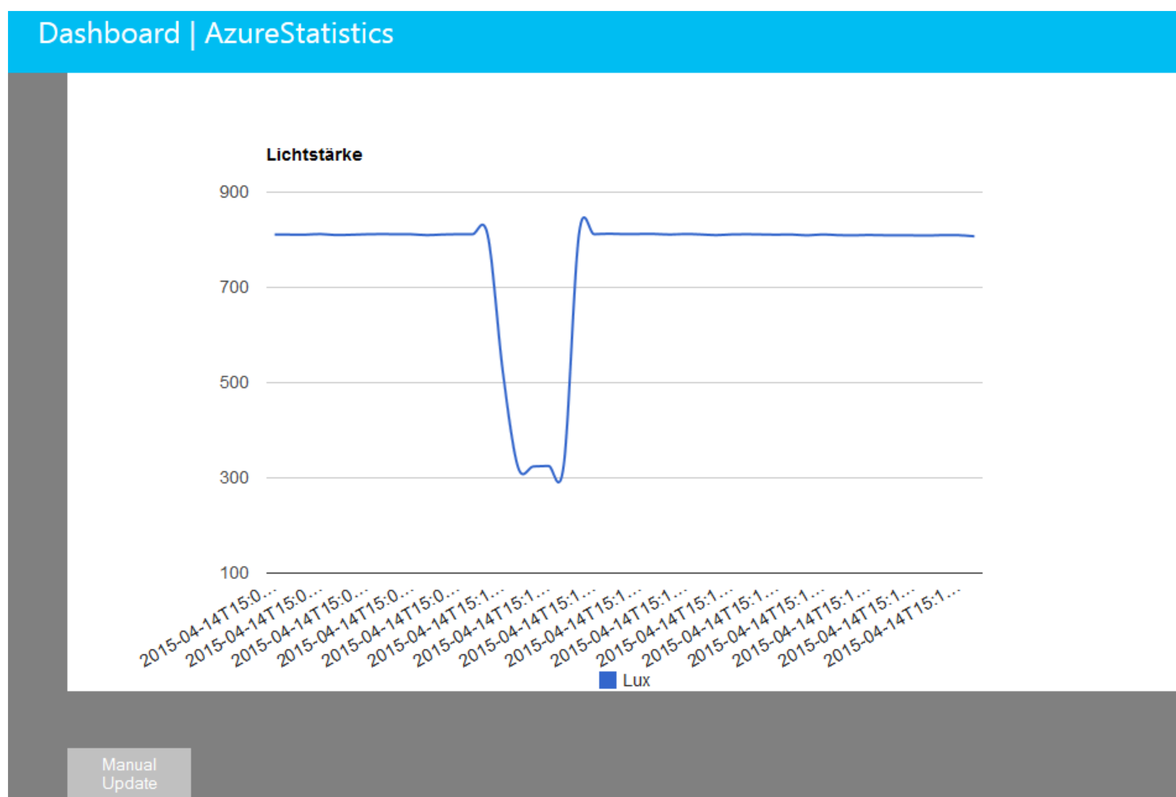


Abbildung 3.3.: Screenshot: Anzeige der Lichtstärke in LoT

3.3. Prototyp 3 - Gadgeteer zu Azure

In diesem Prototyp soll gezeigt werden, wie die von der Gadgeteer Plattform gemessenen Sensordaten mit Hilfe der Cloud Computing-Plattform Microsoft Azure persistiert, analysiert und gesammelt werden. Dabei spielen mehrere Komponenten mit:

- Gadgeteer
- Azure Event Hub
- Azure Stream Analytics
- Azure SQL Database[16]

Folgende Schnittstelle soll also mit dem Prototyp analysiert und getestet werden.

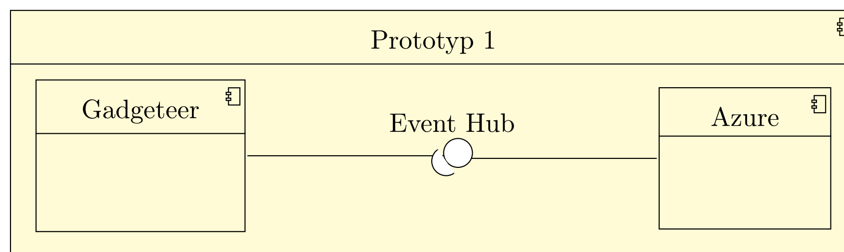


Abbildung 3.4.: Komponentendiagramm: Prototyp 3

Dabei wird neben dem Event Hub zusätzlich die Stream Analytics-Technologie verwendet, welche eine effiziente Auswertung der Sensordaten ermöglicht.

3.3.1. Gadgeteer

In einem ersten Schritt wurden die Sensoren mit dem Gadgeteer-Mainboard verbunden. Wie dies funktioniert, ist im Kapitel 2.2.2.1 beschrieben. Im Visual Studio konnte in einem grafischen Editor die Konfiguration der Gadgeteer Plattform für die Applikation vorgenommen werden. Hierfür wurden die einzelnen Sensoren mit den richtigen Schnittstellen am Mainboard verbunden, so dass im Programm und physikalisch die gleiche Struktur vorliegt. Der Editor initialisiert daraufhin die einzelnen Komponenten und passt dafür die „Program"-Klasse an, welche beim Start des Mainboards ausgeführt wird.

In einem weiteren Schritt wurden die gemessenen Daten der Sensoren ausgelesen. Hier ein kleiner Einblick, wie dies funktioniert. Die verwendete Timer-Instanz ist weiter oben im Code deklariert und tickt im Abstand von einer Sekunde.

```
//Wird beim Start des Mainboards aufgerufen
void ProgramStarted()
    {
        timer.Tick += timer_Tick;
        timer.Start();
    }

    void timer_Tick(GT.Timer timer)
    {
        double temperature = readTemperaturSensor();
        double lux = readLuxSensor();

        SendAMQPMessage(FormatMessage("Temperature", "C",
           temperature));
        SendAMQPMessage(FormatMessage("Lux", "Lux", lux));
    }

    double readTemperaturSensor()
    {
        return tempHumidSI70.TakeMeasurement().Temperature;
    }

    double readLuxSensor()
    {
        return lightSense.GetIlluminance();
    }
```

Auf dem Microcontroller des Gadgeteer-Mainboard wird das .NET Micro Framework 4.3 ausgeführt. In der Analysephase von Azure wurde die Verbindung zum Event Hub nicht mit dem Micro Framework realisiert und deshalb konnte eine offizielle Service Bus Implementation verwendet werden. Leider gibt es noch kein Azure SDK für das .NET Micro Framework. Um mit dem Event Hub zu kommunizieren, bieten sich zwei Möglichkeiten: HTTP POST oder AMQP. Im Vergleich zu HTTP hat AMQP einen kleinen Overhead und ist dafür ausgelegt, um asynchrone Aufrufe zu tätigen. Deshalb wurde eine Implementation von AMQP für das .NET Micro Framework gesucht. Eine Recherche führte zur Implementation Amqp.Net Lite[1] des Azure Service Bus Teams.

Die Amqp.Net Lite Implementation wird bereits in etlichen Projekten verwendet, welche die Anbindung an den Azure Service Bus ermöglichen. Mit Inspiration aus diesen Projekten konnte der Prototyp aus Seite Gadgeteer fertiggestellt werden.

3.3.2. Azure

Für die Entwicklung des gesamten Projekt wird die Cloud Computing-Plattform Microsoft Azure verwendet, welche deshalb auch hier im Prototyp 3 ihren Einsatz findet.

3.3.2.1. Event Hub

Im Kapitel 2.3.1 ist die Funktionsweise des Event Hubs beschrieben. In diesem Kapitel wird gezeigt wie die Konfiguration des Event Hubs vorgenommen wird und wie man eine Verbindung zum Event Hub aufbaut. Im Mannagement Portal[19] von Microsoft Azure lässt sich

der Event Hub erstellen, konfigurieren und überwachen. Dafür wählt man „New“ → „App Services“ → „Service Bus“ → „Event Hub“ → „Custom Create“.

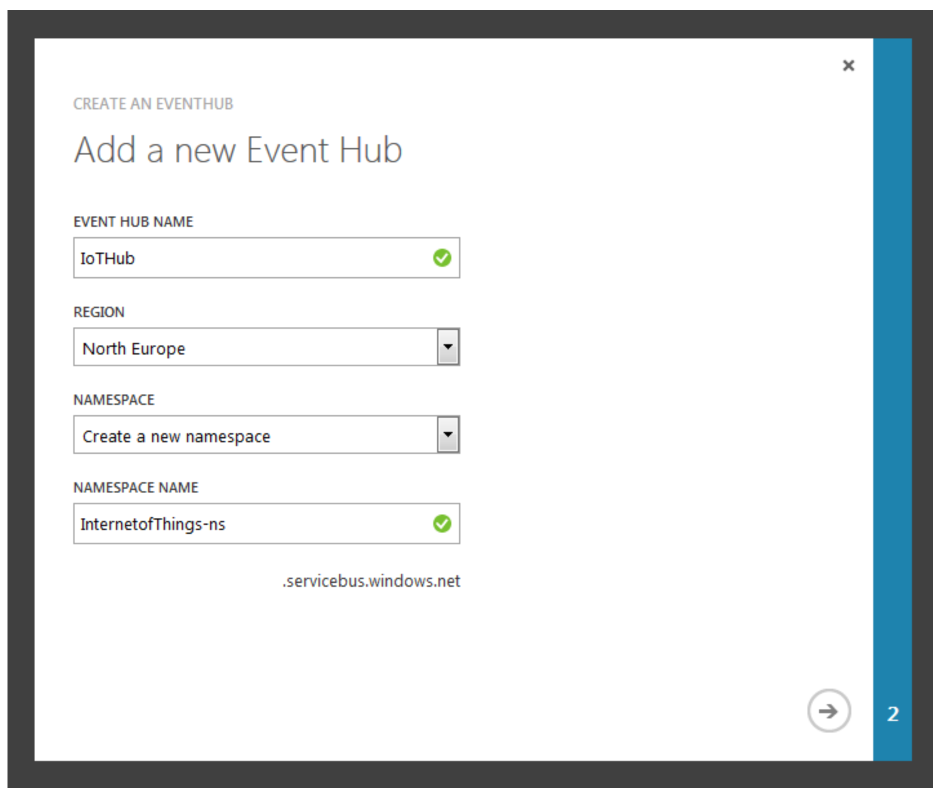


Abbildung 3.5.: Screenshot: Event Hub erstellen

In einem letzten Schritt wird die Konfiguration der Anzahl Partitionen eingestellt und eine Anzahl Tage festgelegt, welche definiert wie lange das Nachrichten im Event Hub Event gespeichert werden bevor sie vom System verworfen werden.

Um Daten in den Event Hub einzuspeisen, braucht man eine Sendeberechtigung. Deshalb wurde eine „send_rule“ im Konfigurations-Tab des Event Hub erstellt. Damit diese Berechtigung geteilt werden kann, ohne dass der Account-Key angeben werden muss, werden Shared Access Signatures(SAS) erstellt und den Regeln zugewiesen. Der Name der Regel und die SAS werden beim Erstellen der AMQP-Verbindung angegeben.

Folgendes Beispiel zeigt den Aufbau der AMQP Adresse:

```
1 const string AMQPAddress = "amqps://RULE_NAME:
    URL_ENCODED_SAS@NAMESPACE.servicebus.windows.net";
```

Mit der Adresse kann ein SenderLink erstellt werden, welcher via der send() Methode eine Nachricht an den Event Hub schickt.

```
1 address = new Address(AMQPAddress);
2 connection = new Connection(address);
3 session = new Session(connection);
4 sender = new SenderLink(session, "send-link", EventHubName);
5
6 //...create a message
7
8 sender.send(message);
```

3.3.2.2. Stream Analytics

In diesem Kapitel wird kurz auf das Erstellen eines Stream Analytics Job eingegangen. Mehr zur Funktionsweise ist im Kapitel 2.3.2 beschrieben. Stream Analytics ist ein Event Consumer vom Event Hub und kann ebenfalls über das Azure Management Portal eingerichtet werden. Allerdings ist Stream Analytics erst in der Preview-Phase, deshalb muss man sich zuerst speziell dafür registrieren. Danach kann der Service unter „New“ → „Data Services“ → „Stream Analytics“ → „Quick Create“ eingerichtet werden.

Sobald ein Job erstellt ist, kann im Interface des Jobs unter „Query“ mittels der Stream Analytics Query Language, eine SQL ähnliche Sprache, definiert werden, wie die gestreamten Daten gefiltert, ausgewertet oder sogar manipuliert werden sollen. Testweise wurde folgender Query eingesetzt, um eine Auswertung der Sensorwerte im 5 Sekunden-Fenster zu berechnen:

```
1     SELECT DateAdd(second,-5,System.TimeStamp) as WinStartTime,
         system.TimeStamp as WinEndTime, DeviceId, Avg(Value) as
         AvgValue, Count(*) as EventCount, Type as Type
2     FROM input
3     GROUP BY TumblingWindow(second, 5), DeviceId, Type
```

Damit die Auswertung nicht einfach verloren geht, wird das Ganze in einer Datenbank gespeichert. Dafür wurde als Output des Stream Analytics Job eine Datenbank angeben, welche die Struktur der Daten aufweist. Im Vorfeld wurde diese erstellt. Im nächsten Kapitel wird beschrieben, wie die SQL-Database eingerichtet wurde.

3.3.2.3. SQL Database

Die Auswertung der Events soll in einer Datenbank gespeichert werden. Zuerst wurde im Management Portal eine Storage Account eingerichtet. Nach der Erstellung wurde die Datanbankstruktur definiert. Dazu wurden mittels des Visual Studios folgende SQL Befehle ausgeführt:

```
1  CREATE TABLE [dbo].[AvgReadings] (
2      [WinStartTime]   DATETIME2 (6) NULL,
3      [WinEndTime]     DATETIME2 (6) NULL,
4      [DeviceId]      BIGINT NULL,
5          [Type] VARCHAR(255) NULL,
6      [AvgValue] FLOAT (53)    NULL,
7      [EventCount] BIGINT null
8  );
9
10 GO
11 CREATE CLUSTERED INDEX [AvgReadings]
12     ON [dbo].[AvgReadings]([DeviceId] ASC);
```

4. Design

4.1. Komponenten

Das Endsystem soll aus den folgenden drei Hauptkomponenten bestehen:

Azure
Azure speichert alle Sensordaten und analysiert sie mit den Analyse Services.

Gadgeteer
Das Gadgeteer liefert zum einen alle Sensordaten (Temperatur, Luftfeuchtigkeit und Lichteinstrahlung). Zum anderen sind Aktoren (LED, Display) vorhanden, welche gesteuert werden können.

Lab of Things
LoT erkennt das Gadgeteer und bindet es im System ein. Auf dem Framework laufen auch die Apps zur Steuerung vom Gadgeteer und zur Anzeige von Sensordaten.

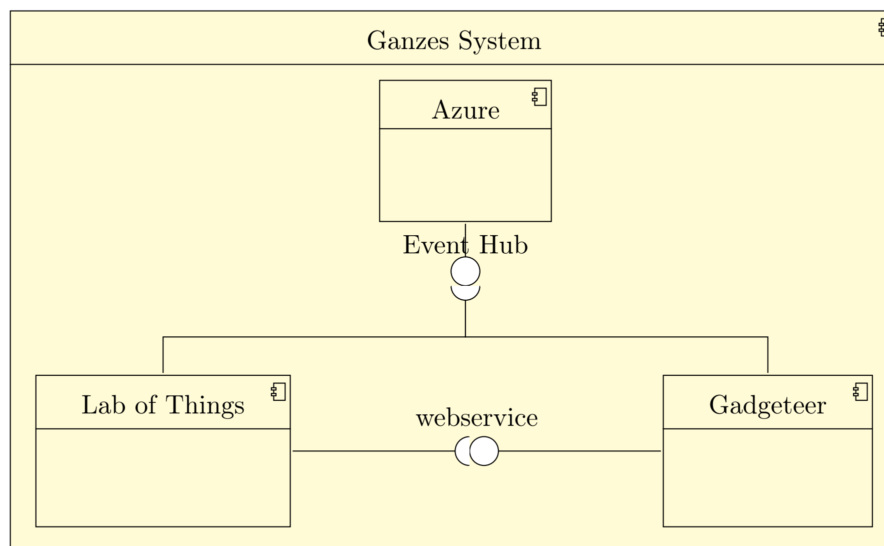


Abbildung 4.1.: Komponentendiagram: Endsystem

Wie auf der Abbildung zu sehen ist, kommunizieren alle Komponenten miteinander. Dabei gibt es nur zwei Interfaces. Azure und das Gadgeteer bieten Interfaces an.

Das LoT benutzt das Interface vom Gadgeteer, um zum einen Daten wie Bilddaten abzuholen und zum andern, um das Gadgeteer zu steuern. Es können Lichter ein- und ausgeschaltet oder bestimmte Informationen auf dem Display angezeigt werden.

Das Gadgeteer benutzt das Interface von Azure nur, um Sensordaten dort zu speichern. Die Daten werden von Azure dann selbstständig behandelt. Das LoT hingegen benutzt das Interface von Azure, um diese Sensordaten und die verarbeiteten Daten abzuholen.

4.2. Architektur

4.2.1. Lab of Things

4.2.1.1. Gadgeteer Workbench

Die Gadgteer Workbench Applikation wird in C# geschrieben und soll folgende Struktur aufweisen:

- **GadgeteerWorkbench:** Startet den WCF Service, Startet den WebFileHoster für die Webseite und verwaltet die Aktionen beim Hinzufügen und Entfernen eines Geräts. Zudem erfolgt hier die Kommunikation mit den Drivern.
- **IManager:** Das Interface bestimmt, welche Methoden von den Manager-Klassen implementiert werden müssen, damit die GadgeteerWorkbench deren Funktionalität einheitlich benutzen kann.
- **Managers:** Ausser dem DatabaseManager und dem RuleManager implementieren alle Manager-Klassen das IManager Interface. Sie wissen, welche Geräte momentan verfügbar sind, verwalten die aktuellen Messwerte und haben über den DatabaseManager den Zugriff auf die Azure Datenbank.
- **GadgeteerWorkbenchService:** Definiert die WCF-Schnittstellen und Ruft die entsprechenden Funktionen in den IManager Klassen auf
- **Rules:** Enthält alle für die Rule-Engine benötigten Klassen.

4.2.1.2. Gadgeteer Workbench Webseite

Ein Teil der Gadgeteer Workbench Applikation ist eine Webseite. Diese wird vom WebFileHoster in der GadgeteerWorkbench Klasse gehostet. In diesem Kapitel wird die Struktur der Webseite aufgezeigt. Wichtig zu verstehen ist es zudem, dass Angular Webseiten ihre Businesslogik in sogenannten Controllern verwalten. Bei dieser Webseite handelt es sich um eine Single Page Application. Es werden Views definiert, welche von Angular dynamisch geladen werden. So kann HTML Code dynamisch in die Seite eingebunden werden.

Die Abbildung 4.2 zeigt die Ordnerstruktur der Webseite, während die Tabelle 4.1 deren Inhalte beschreibt.

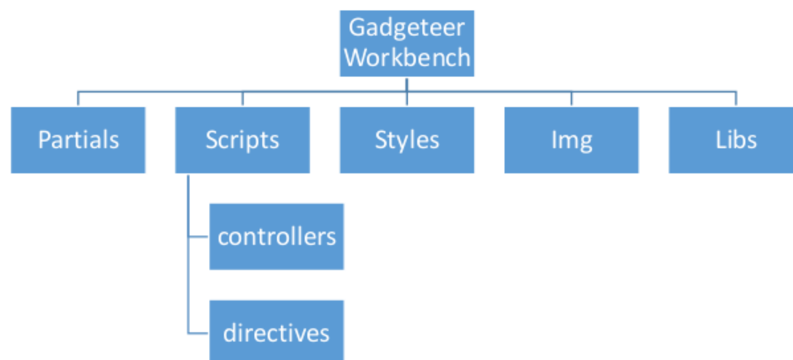


Abbildung 4.2.: Ordnerstruktur: Admin Tool

Partials	Beinhaltet die Views, welche von AngularJS dynamisch in die Webseite geladen werden
Styles	Alle benötigten CSS Dateien sind in diesem Ordner enthalten
Scripts	Beinhaltet alle eigenen Javascript Dateien. Dazu gehören controllers und directives
Libs	Bootstrap [2], spezielle angular Libraries und natürlich alle Corelibraries des AngularJS Frameworks.
Img	Loader GIF-Datei

Tabelle 4.1.: Ordnerstruktur: Webseite

4.2.2. Gadgeeteer

Auch die Gadgeteer Applikation wird in C# geschrieben und soll folgende Struktur aufweisen:

- **Program:** Dies ist eine vorgegebene Klasse. Sie wird automatisch vom Gadgeteer gestartet und bietet den Einstiegspunkt ins Program.
- **Networking:** Beinhaltet alle Klassen, welche Netzwerkaktivitäten anbieten, dazu gehört der ConnectionManager, welcher die Netzwerkverbindung herstellt und die für den Webserver benötigten Klassen.
- **EvenHubManager:** Diese Managerklasse stellt die Verbindung zum Azure Event Hub her und sendet die Sensorwerte per AMQP.
- **WebEventManager:** Diese Klasse definiert die Webschnittstellen und registriert diese beim Webserver.
- **IdentifierSetting:** Ist eine Hilfsklasse, welche die eindeutigen Bezeichner für die einzelnen Sensoren hält.

4.3. Kommunikation

4.3.1. Kommunikation zwischen Lab of Things und Gadgeteer

Die Kommunikation, die im Kapitel 3.1.3.1 beschrieben wird, hat den Nachteil, dass im Gadgeteer nur ein Sensor vom LoT erkannt wird. Nun sollen aber alle Sensoren erkannt werden. Das folgende Sequenzdiagramm zeigt, wie dies funktioniert.

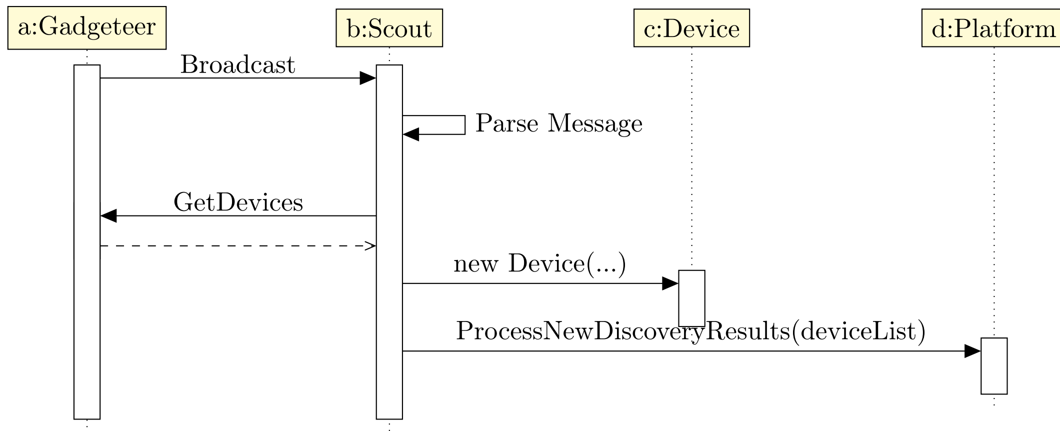


Abbildung 4.3.: Sequenzdiagramm: Gadgeteer erkennen

Ein grosser Unterschied zum Konzept im Prototyp ist der Zwischenschritt „GetDevices“, bei dem das LoT eine Liste von allen Geräten vom Gadgeteer holt. Diese Nachricht wird mit JSON verschickt und sieht folgendermassen aus.

```
Pfad: /devices

Response:
{
    Devices:
        [{
                Type: <String>,
                DeviceId: <String>,
                ViaAzure: <Boolean>,
        }]
}
```

Aus diesen Informationen erstellt der Scout die erkannten Devices und registriert diese im System. Sollte ein Device dann vom User hinzugefügt werden, so kann die Plattform aufgrund der Device Klasse den richtigen Driver instanziieren.

Um die Sensordaten zu erhalten, stellen die Driver in einem vorgegebenen Zeitintervall einen Request an das Gadgeteer und erhalten als Antwort die gewünschten Daten. Als Beispiel hier der Request, um die Temperatur vom Gadgeteer zu erhalten:

```
Request:
Pfad: /temperature

Response:
{
    DeviceId: <String>, //Unique Identifier
    Type: <String>, //Temperature, Light Strength, usw.
    Unit: <String>, //Celsius, Lux, usw.
    Value: <Number>
}
```

4.3.2. Kommunikation Gadgeteer zu Azure Event Hub

Hier wird das Format für die Kommunikation zwischen Gadgeteer und dem Azure Event Hub festgelegt, damit nur einheitliche Nachrichten im Event Hub sind und alle gleichermassen ausgewertet werden können. Die Pfadangabe kann hier nicht so einfach dargestellt werden. Der Aufbau ist jedoch in Kapitel 3.3.2.1 beschrieben.

```
Request:
{
    DeviceId: <String>, //Unique Identifier
    Type: <String>, //Temperature, Light Strength, usw.
    Unit: <String>, //Celsius, Lux, usw.
    Value: <Number>
}
```

4.4. Sequenzdiagramme

Dieses Kapitel soll verdeutlichen, wie die Sensordaten von Gadgeteer auf die Webseite im LoT kommen. Dabei gibt es zwei Wege. Der eine führt direkt ins System der andere geht noch zuerst über Windows Azure. Um den Ablauf besser zu verstehen, wird der Gesamtablauf in einzelnen Teilabläufen aufgezeigt.

4.4.1. Gadgeteer zu Lab of Things

4.4.1.1. Direkter Weg

Die direkte Kommunikation ist sehr simpel gehalten. Wie im Kapitel 4.3.1 beschrieben wird, sendet LoT in einem vorgegebenen Zeitintervall HTTP-Requests an das Gadgeteer Device. Als Beispiel wird hier die Temperatur abgefragt.

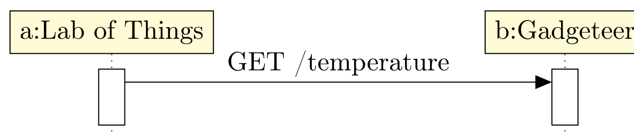


Abbildung 4.4.: Sequenzdiagramm: Requests von LoT zu Gadgeteer

4.4.1.2. Über Windows Azure

Zur Datenanalyse schickt Gadgeteer die Messwerte an die Cloud Computing-Plattform. Die Services Event Hub und Stream Analytics bieten die optimale Funktionalität dafür an. Per AMQP gelangen die Messwerte an den Event Hub und werden schliesslich vom Stream Analytics Service in einer Datenbank gespeichert. Auffällig ist, dass der Stream Analytics Service die Daten selbstständig aus dem Event Hub liest.

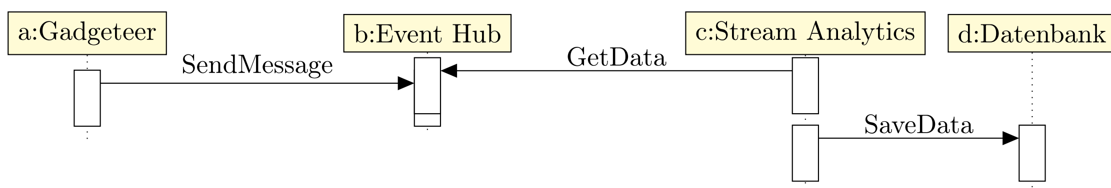


Abbildung 4.5.: Sequenzdiagramm: Messwerte an Azure senden

Im nächsten Schritt wird die Datenbank vom LoT ähnlich wie im Kapitel 4.4.1 angesprochen und die erstellten Statistikdaten werden ausgelesen.

4.4.2. Lab of Things

4.4.2.1. Driver zu Applikation

Beim Hinzufügen eines neuen Geräts in LoT wird eine neue Driver Instanz gestartet. Diese registriert im System einen Port, mit dem das Gerät eindeutig identifiziert werden kann. Das System benachrichtigt daraufhin alle Applikationen darüber, dass ein neuer Port zur Verfügung steht. Die Applikationen haben nun Möglichkeit, sich beim Port zu registrieren. So wird ihnen über alle Neuigkeiten berichtet. Dieser Benachrichtigungsablauf kann aus dem unten stehenden Sequenzdiagramm entnommen werden:

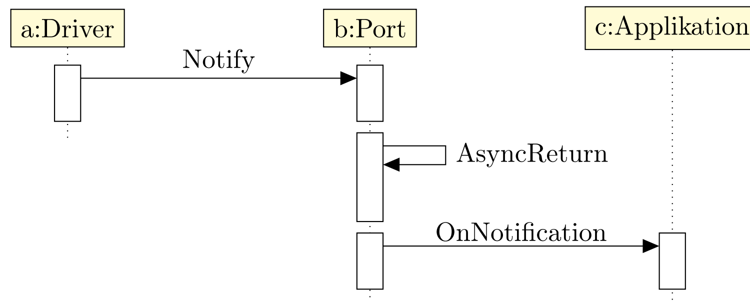


Abbildung 4.6.: Sequenzdiagramm: Benachrichtigungen vom Driver

4.4.2.2. Applikation zu Driver

Wie im Kapitel 4.4.2.1 beschrieben wurde, registriert die Driver-Instanz den Port im System. Zusätzlich wird mittels Delegate festgelegt, welche Methode im Driver aufgerufen werden soll, wenn das System die Invoke-Methode auf dem Port aufruft. Das System gewährt somit der Applikation die Möglichkeit, eine Methode im Driver aufzurufen und so dem Driver Anweisungen zu erteilen. In folgendem Diagramm kann der Ablauf nachvollzogen werden:

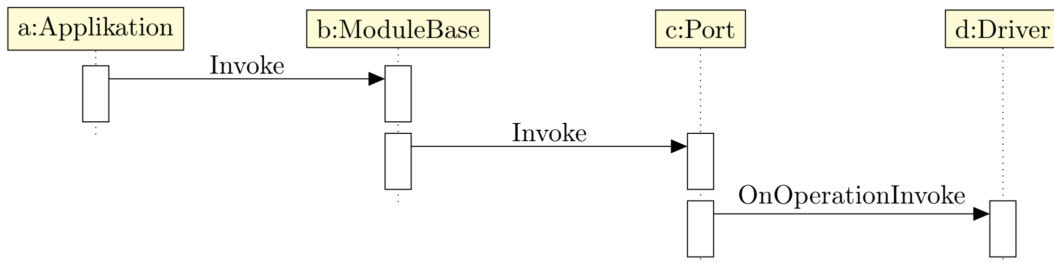


Abbildung 4.7.: Sequenzdiagramm: Driver ansprechen

4.4.2.3. Webseite zu Azure Datenabank

Um die ausgewerteten Daten von Azure zu lesen, wird von der Applikation aus eine SQL-Abfrage auf der Azure Datenbank gemacht. Um diesen Prozess anzustossen, ruft die Webseite über eine WCF-Schnittstelle die entsprechende Methode in der Applikation auf. Sobald die Daten gelesen sind, werden sie zurück an die Webseite gesendet. Dort wird dann die Callback-Methode ausgeführt.

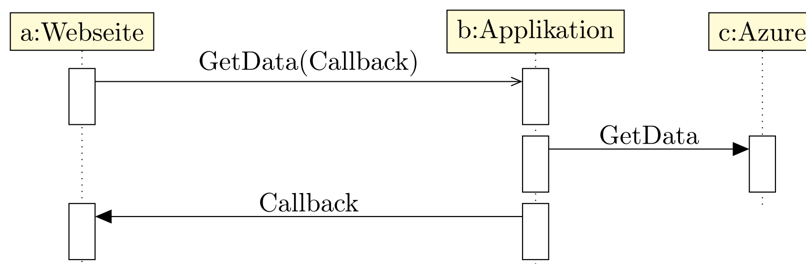


Abbildung 4.8.: Sequenzdiagramm: Webseite zu Azure

4.4.2.4. Webseite zu Applikation

Die Webseite holt die Daten genau gleich wie in Kapitel 4.4.2.3 aus der Applikation. Der Aufruf zu Azure kann hier jedoch weggelassen werden.

4.5. Rules

Das Endsystem soll die Möglichkeit haben, bestimmte Aktionen automatisch durchzuführen wenn vom Benutzer eingestellte Bedingungen erfüllt werden. Dieses Rule-System zeigt das Potential einer Internet of Things Anwendung erneut auf.

Das Rule-System soll einfach und verständlich zu bedienen sein. Rules können vom Benutzer hinzugefügt, angezeigt und gelöscht werden.

Der Benutzer wählt aus einer Liste einen Sensor aus und konfiguriert Vergleichswert und Vergleichsfunktion. Diese Bedingung wird von nun an als Trigger bezeichnet. Zeigt der definierte Sensor einen Wert der mit der Vergleichsfunktion positiv reagiert, so ist der Trigger erfüllt und es wird eine Action ausgelöst.

Eine Action besteht einerseits aus einem Modul, das sichtbare Änderungen erlaubt, andererseits aus einem Text, welcher beschreibt, inwiefern sich das Module verändert. Zu den erwähnten Modulen gehören zwei LEDs und ein Display. Bei den LEDs kann der Text die Farbe, welche eingestellt werden soll ,beschreiben und beim Display wird der Text eins zu eins verwendet.

Eine Rule wird nur einmal ausgeführt, denn es könnten sonst Regeln definiert werden, die immer zutreffen. Dies hätte zur Folge, dass gewisse Funktionen, wie zum Beispiel die Steuerung eines LEDs, nicht mehr richtig funktionieren. Die Regel würde dazu führen, dass das System automatisch immer wieder die gleichen Einstellungen am LED vornimmt. Aus diesem Grund hat eine Rule den Zustand „Aktiviert“ oder „Deaktiviert“.

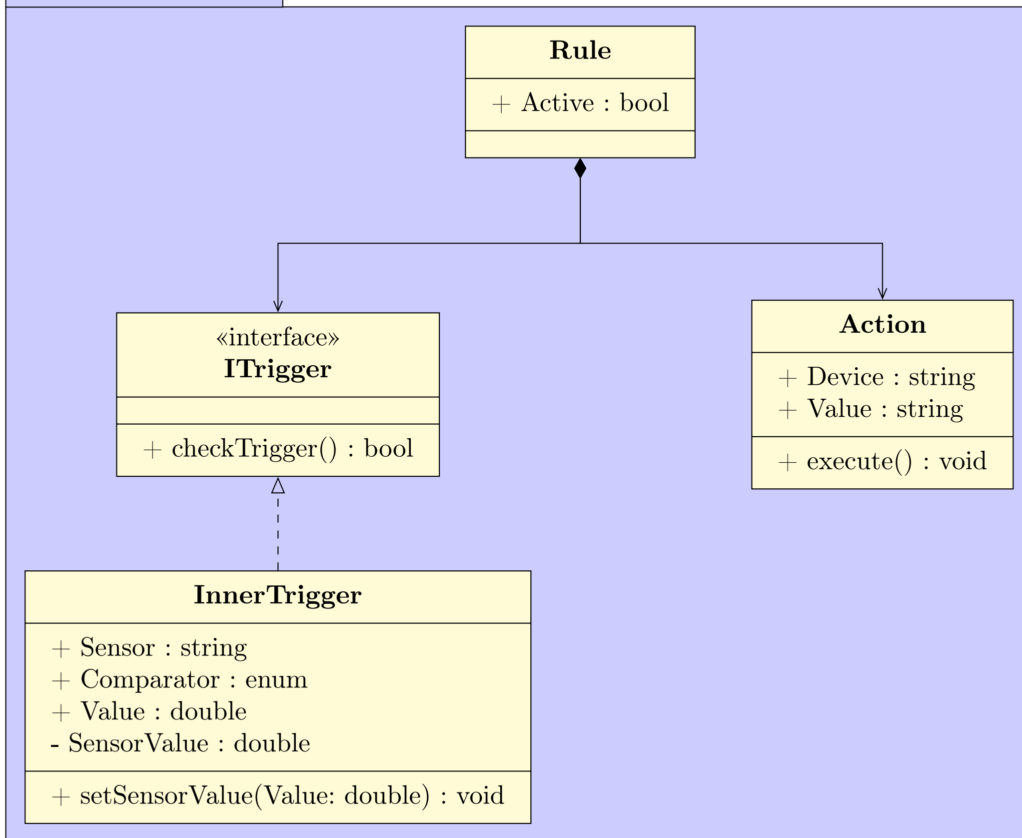
4.5.1. Grammatik

Die Rules wurden mit folgender Grammatik zusammengefasst, um einen Überblick über den Aufbau der Rules zu haben:

$$
\begin{array}{ll}
\text{Rule} & \rightarrow \text{Trigger} + \text{Action} \\
\text{Trigger} & \rightarrow \text{Sensor} + \text{Comparer} + \text{SValue} \\
\text{Comparer} & \rightarrow = \\
 & \rightarrow \neq \\
 & \rightarrow \geq \\
 & \rightarrow \leq \\
 & \rightarrow < \\
 & \rightarrow > \\
\text{Action} & \rightarrow \text{Device} + \text{DValue} \\
\text{SValue} & \rightarrow \text{Number} \\
\text{DValue} & \rightarrow \text{String} \\
\text{Sensor} & \rightarrow \text{String} \\
\text{Device} & \rightarrow \text{String}
\end{array}
$$

4.5.2. Umsetzung

Da die Rules auch nach einem Neustart der Plattform noch vorhanden sein sollen, müssen sie gespeichert werden. Es ist also wichtig, dass die Rule-Objekte vollständig serialisierbar sind. Damit dies funktioniert, müssen Geräte als String repräsentiert werden.



Das Überprüfen der Regel soll im „Rule“ Objekt gemacht werden. Dort kann in einem Thread die „checkTrigger()“ Funktion regelmässig überprüft werden und „execute()“ ausgeführt werden, wenn die Antwort positiv ausfällt.

Um die Sensordaten aktuell zu halten, soll die „setSensorValue()“ Funktion zur Verfügung stehen. Die Sensoren können so die neusten Daten dorthin melden.

4.6. Wireframes

Das Design vom Gadgeteer Workbench ist flach geplant. Mit wenigen Klicks erreicht man also die gesamte Funktionalität der Applikation. Die Kacheln wurden entsprechend dem LoT Layout vorgesehen. Auch die vorgegebenen Farben weiss und blau sollen durchgängig eingesetzt werden. In den folgenden Kapiteln werden die einzelnen Seiten beschrieben. Dabei sind die Wireframes in black and white und weichen deshalb vom geplanten Farbkonzept ab. Bei allen Seiten ist die Navigation am oberen Rand eingebaut. Man kann entweder zurück zur Übersicht oder man kann die Applikation verlassen und zum Dashboard zurück navigieren.

4.6.1. Übersicht

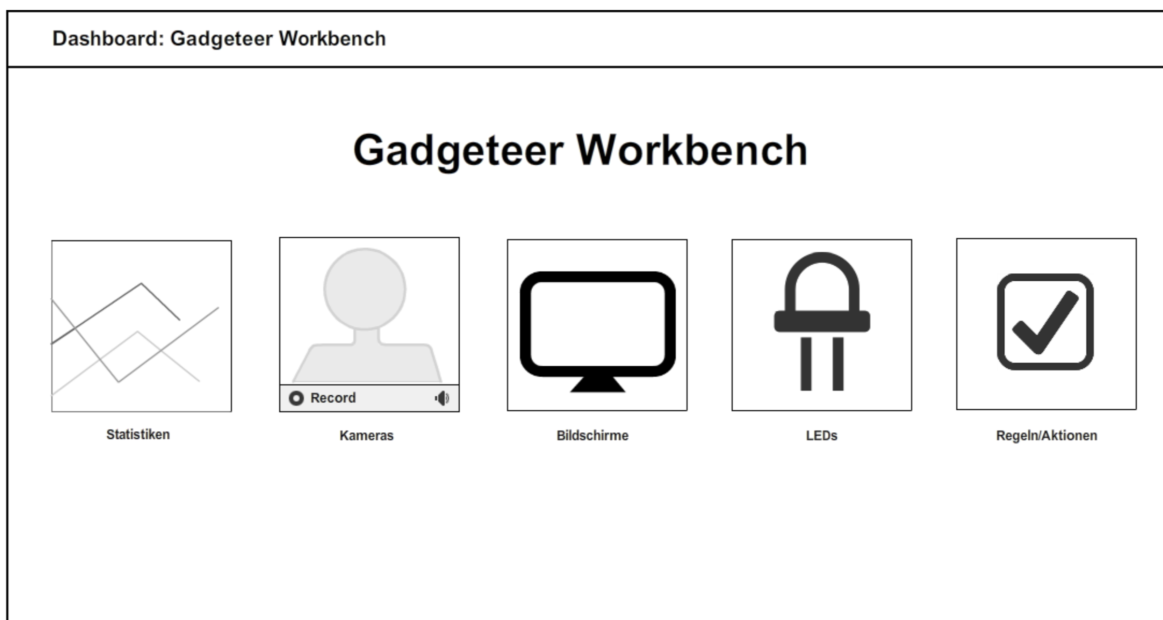


Abbildung 4.9.: Wireframe: Übersicht

Die Übersicht bietet eine verständliche Navigation durch die Funktionalitäten der Applikation. Wie bereits erwähnt, ist das Kachel-Layout dem Framework angepasst. Mit einem einfachen Mausklick auf eine der Kacheln navigiert man zur der entsprechenden Teilfunktion der Applikation.

4.6.2. Statistiken

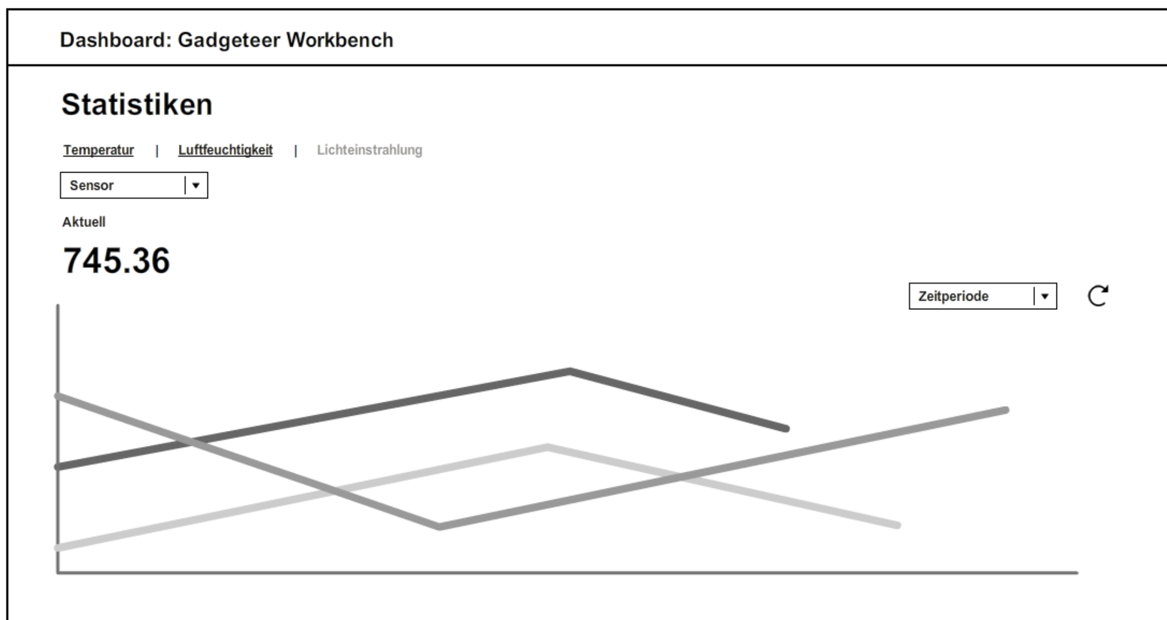


Abbildung 4.10.: Wireframe: Statistiken

Auf dieser Seite kann man sowohl die aktuellen Messwerte der Sensoren als auch eine zeitliche Statistik einsehen. Ganz oben auf der Seite wählt man zuerst aus, welche Kategorie von Sensoren man sehen will. Dazu stehen die Kategorien Temperatur, Luftfeuchtigkeit und Lichteinstrahlung zur Auswahl. Direkt unterhalb kann man nun einen verfügbaren Sensor auswählen. Beim Öffnen der Seite werden standardmässig die erste Kategorie und der erste darin enthaltene Sensor ausgewählt. Unter der Auswahl des Sensors wird der aktuelle Messwert angezeigt. Im unteren Teil der Seite wird die Statistik des Sensors angezeigt. Rechts oberhalb der Statistik kann man dann auswählen, welchen Zeitraum der Statistik man anzeigen möchte und daneben kann man die Statistik neu laden. Es werden dann die neusten Daten aus der Datenbank geladen und angezeigt.

4.6.3. Kameras

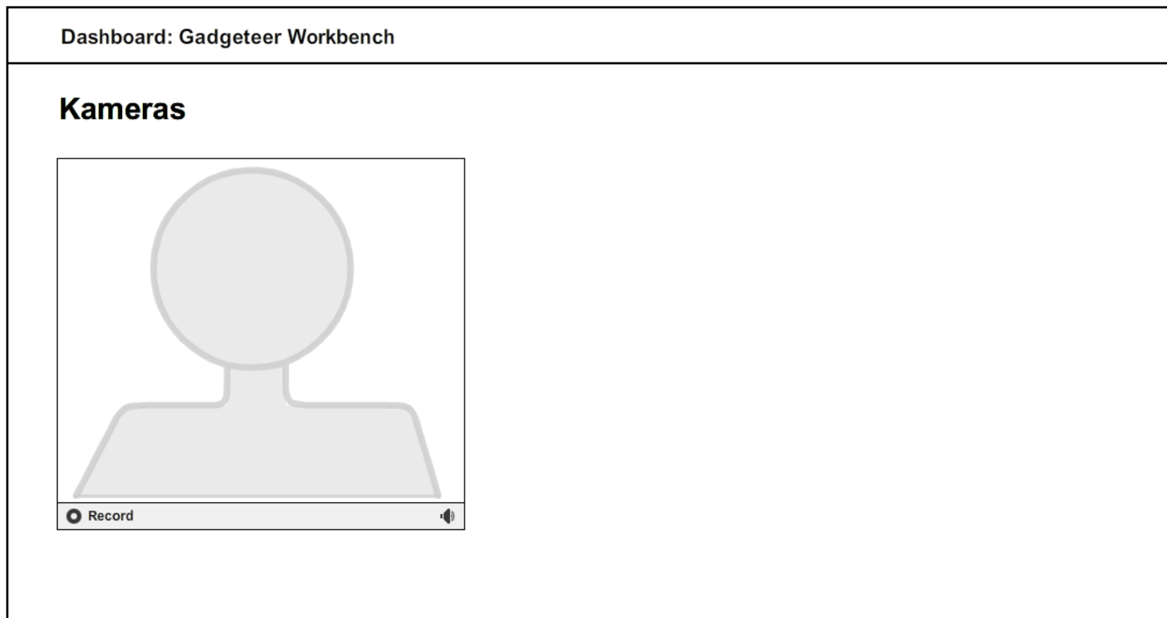


Abbildung 4.11.: Wireframe: Kameras

Die Kamera Seite macht nicht besonders viel. Sie zeigt lediglich das von der Kamera erhaltene Bild an.

4.6.4. Bildschirme

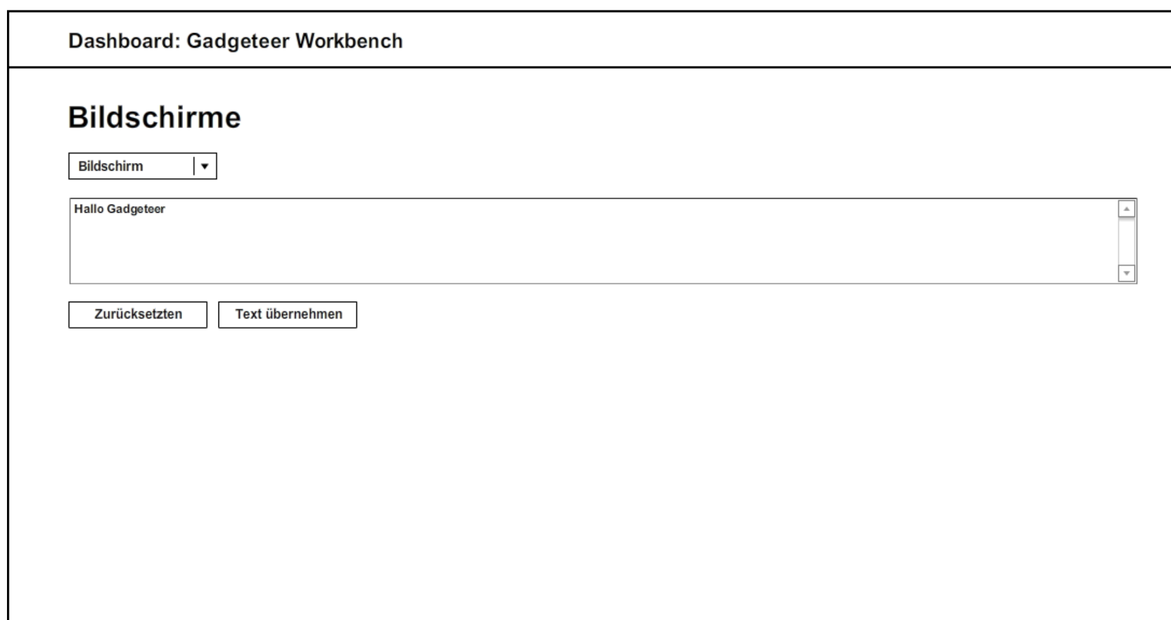


Abbildung 4.12.: Wireframe: Bildschirme

Auf dieser Seite kann man den Text auf einem verfügbaren Bildschirm anzeigen und anpassen. Direkt am Anfang der Seite wählt man den gewünschten Bildschirm aus. Es ist wieder per Default der erste verfügbare Bildschirm ausgewählt. Wenn der Bildschirm schon einen Text anzeigt, wird dieser automatisch in die Textbox geladen und er kann angepasst werden. Ist

man fertig mit Anpassen, kann man den Text auf den Bildschirm übertragen, indem man auf den Button „Text übernehmen“ klickt. Ist man nicht zufrieden mit dem geschriebenen Text, so kann man die Textbox mit einem Klick auf den „Zurücksetzten“-Knopf wieder zurücksetzen. Dies hat jedoch keinen Einfluss auf den Bildschirm.

4.6.5. LEDs

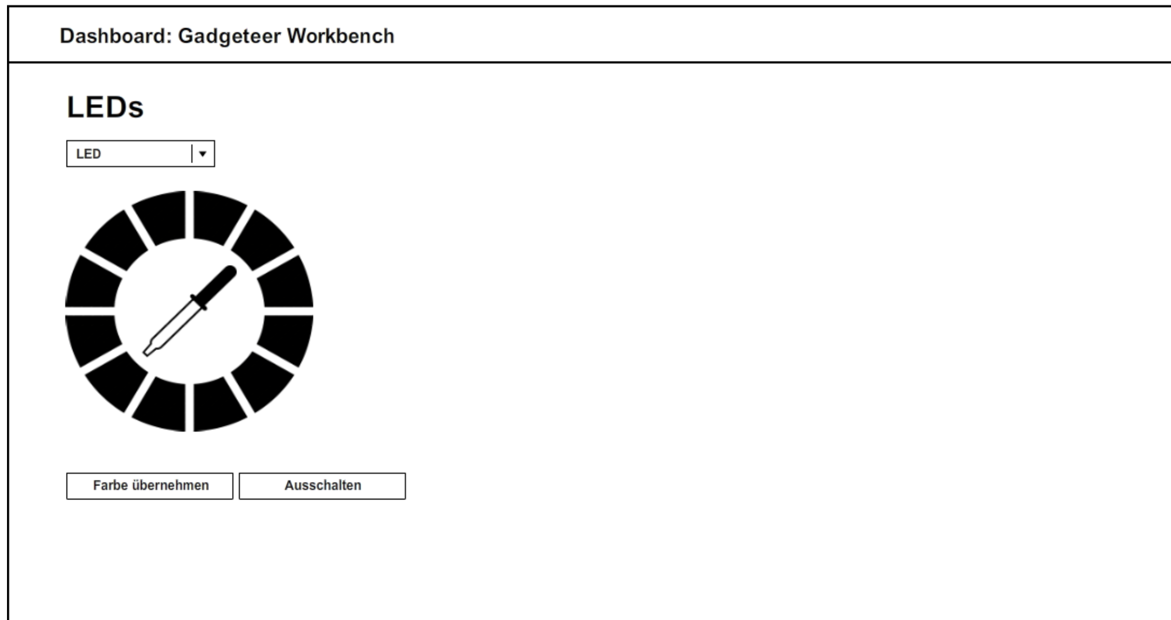


Abbildung 4.13.: Wireframe: LEDs

Wie bei den anderen Seiten kann man auch auf der LED Seite das gewünschte LED auswählen und es ist bereits das erste verfügbare ausgewählt. Mit einem angenehmen Farbwähler kann man bestimmen, in welcher Farbe das selektierte LED leuchten soll. Mit dem „Farbe Übernehmen“-Knopf stellt das System eine Verbindung zum entsprechenden LED her und stellt die ausgewählte Farbe ein. Der Ausschaltknopf stellt das LED ab.

4.6.6. Regeln/Aktionen

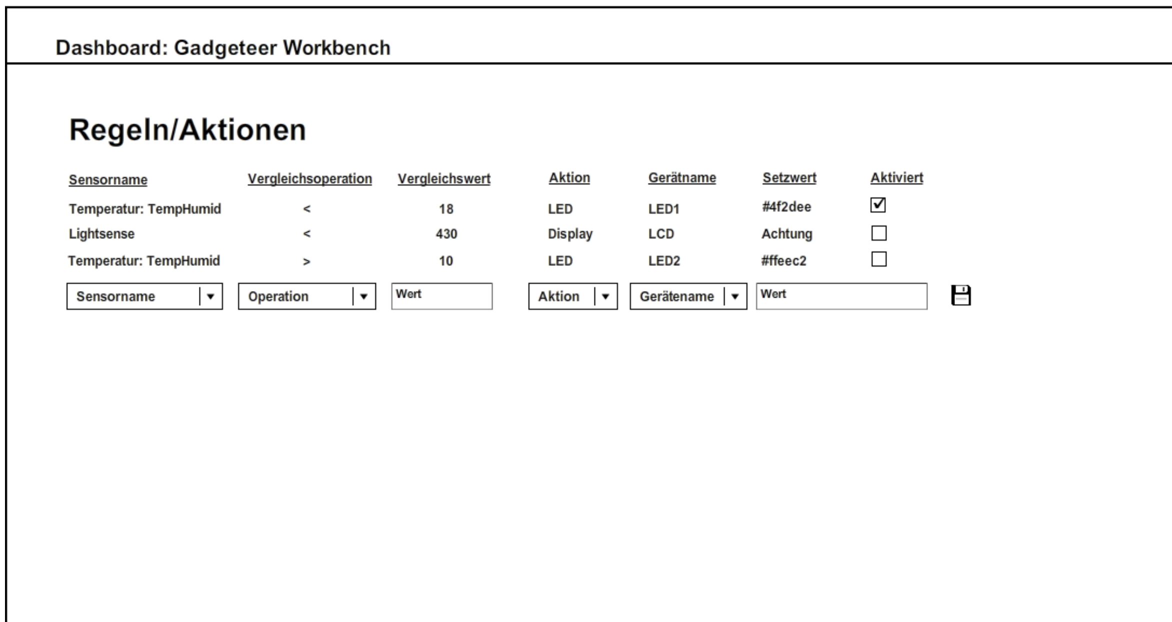


Abbildung 4.14.: Wireframe: Regeln/Aktionen

Die Seite Regeln und Aktionen bietet die Möglichkeit, genau einzustellen, zu welchen Konditionen welche Aktion ausgeführt werden soll. Die bereits konfigurierten Regeln werden direkt unter dem Titel tabellarisch dargestellt. Hier eine kleine Übersicht, was die Spalten zu bedeuten haben:

- Sensorname: Der Name des Sensors, den man betrachten will.
- Vergleichsoperation: Die Vergleichsoperation, welche auf den aktuellen Messwert ausgeführt werden soll.
- Vergleichswert: Der Wert, mit dem der aktuelle Wert verglichen werden soll.
- Aktion: Die Aktion, welche in Kraft tritt, wenn die ausgewählte Bedingung erfüllt wird.
- Gerätname: Das Gerät, welches, diese Aktion ausführen soll.
- Setzwert: Ein Parameter, welcher die Aktion genauer beschreibt. Z.B. eine Farbe, welche mit der Aktion gesetzt werden soll
- Aktiviert: Zeigt an, ob die Regel aktiviert ist oder nicht.

In der untersten Zeile der Tabelle hat man die Möglichkeit, eine neue Regel zu definieren. Dazu geht man von links nach rechts durch die Selektion der einzelnen Einstellungen und speichert am Schluss die Regel mit einem Klick auf das Speicher-Symbol ganz am rechten Ende der Zeile. Bereits vorhandene Regeln können mit einem Klick auf das Löschen-Symbol aus der Liste entfernt werden.

4.7. Testplanung

Das System soll auf Grund der vielen Frameworks und der daher mit automatisierten Unit Test schwierigen Testbarkeit, ausschliesslich mit Systemtests geprüft werden. Ziel dieses Kapitels ist es zusammen mit den Requirements zu definieren, welche Tests ausgeführt werden und welches die erwarteten Resultate sind. Somit kann sichergestellt werden, dass das System richtig funktioniert.

4.7.1. LoT

Nr.	Testbezeichnung	Erwartungswert
1	Temperatur anzeigen	Die aktuelle Temperatur der verfügbaren Temperatursensoren wird auf der Webseite angezeigt.
2	Luftfeuchtigkeit anzeigen	Die aktuelle Luftfeuchtigkeit der verfügbaren Luftfeuchtigkeitsensoren wird auf der Webseite angezeigt.
3	Lichteinstrahlung anzeigen	Die aktuelle Lichteinstrahlung der verfügbaren Lichtsensoren wird auf der Webseite angezeigt.
4	Temperatur-Statistik anzeigen	Die via Microsoft Azure ausgewerterten Temperatur-Messwerte werden gelesen und in einem Diagramm auf der Webseite dargestellt.
5	Luftfeuchtigkeits-Statistik anzeigen	Die via Microsoft Azure ausgewerteten Luftfeuchtigkeits-Messwerte werden gelesen und in einem Diagramm auf der Webseite dargestellt.
6	Lichteinstrahlungs-Statistik anzeigen	Die via Microsoft Azure ausgewerteten Lichteinstrahlungs-Messwerte werden gelesen und in einem Diagramm auf der Webseite dargestellt.
7	Kamerabild anzeigen	Das aktuelle Kamerabild wird auf der Webseite angezeigt.
8	LED Farbe setzten, LED ausschalten	Für die verfügbaren LEDs kann jeweils die Farbe eingestellt oder das LED ausgeschaltet werden.
9	Regeln anzeigen, hinzufügen, löschen	Regeln werden je nach Operation angezeigt, hinzugefügt oder gelöscht.
10	Bildschirm Text setzen, anzeigen	Je nach Operation wird der Text auf dem Display angezeigt oder gesetzt.

Tabelle 4.2.: Testplanung LoT

4.7.2. Gadgeteer

Nr.	Testbezeichnung	Erwartungswert
1	Netzwerkadresse beziehen	Das Gadgeteer Gerät bezieht eine IP-Adresse.
2	Beacon senden	Zur Erkennung des Geräts durch den Scout im LoT werden regelmässig Beacons gesendet.
3	Webserver läuft	Der Webserver ist funktionsbereit und zeigt die Standardseite an.
4	Devices-Schnittstellt	Die Schnittstelle antwortet mit einer Liste der aktuell angeschlossenen Geräte.
5	Temperatur-Schnittstellt	Die Schnittstelle antwortet für jedes verfügbare Gerät mit der aktuellen Temperatur.
6	Luftfeuchtigkeits-Schnittstelle	Die Schnittstelle antwortet für jedes verfügbare Gerät mit der aktuellen Luftfeuchtigkeit.
7	Lichteinstrahlungs-Schnittstelle	Die Schnittstelle antwortet für jedes verfügbare Gerät mit der aktuellen Lichteinstrahlung.
8	Kamera-Schnittstelle	Die Schnittstelle antwortet mit dem aktuellen Kamerabild.
9	LED-Schnittstelle	Die Schnittstelle setzt die Farbe auf den verfügbaren LEDs oder schaltet sie aus.
10	Display-Schnittstelle	Die Display-Schnittstelle setzt den angezeigten Text oder gibt den aktuellen Text zurück.
11	Sensorwerte an Azure senden	Die aktuellen Sensorwerte werden an den Event Hub gesendet.

Tabelle 4.3.: Testplanung Gadgeteer

4.7.3. Microsoft Azure

Nr.	Testbezeichnung	Erwartungswert
1	Event Hub eingerichtet und funktionsbereit	Der Event Hub ist eingerichtet und funktionsbereit.
2	Stream Analytics eingerichtet und funktionsbereit	Der Stream Analytics-Service ist eingerichtet und funktionsbereit.
3	SQL-Database eingerichtet und funktionsbereit	Die SQL-Database ist eingerichtet und funktionsbereit.
4	Event Hub Daten empfangen	Der Event Hub empfängt die aktuellen Sensorwerte vom Gadgeteer.
5	Daten werden mit Stream Analytics ausgewertet	Die empfangenen Daten werden durch den Stream Analytics-Service ausgewertet.
6	Speichern der Resultate der Analyse	Der Stream Analytics-Service speichert seine Resultate in der SQL-Database.

Tabelle 4.4.: Testplanung Microsoft Azure

5. Umsetzung

5.1. Lab of Things

5.1.1. Scout

Bei der Implementierung des Gadgeteer-Scouts wurde der im Prototyp 1 entwickelte Scout (siehe Kapitel 3.1) als Vorlage genommen und den Bedürfnissen angepasst. Der Scout besteht aus einer Hauptklasse, einem Service und einer HTML Seite.

Die HTML-Seite wird beim Konfigurieren eines neuen Geräts angezeigt. Beim Öffnen der Konfigurationsseite wird der AddDeviceFinalDeviceSetup-Call ausgeführt. Darin waren leider noch Fehler zu beheben. Mehr dazu im Kapitel 5.1.4.1.

In der Hauptklasse „GadgeteerScout" wird das Gadgeteer gesucht. Dies wird durch den Empfang von Beacons erreicht. Neben der Info, dass ein Gerät im Netzwerk verfügbar ist, erhält der Scout auch gleich die IP-Adresse des Senders. Dies ist eine wichtige Information, um den Webserver auf dem Gerät anzusprechen. Beim Erhalten eines Beacons wird die „CreateDevices"-Methode aufgerufen. Sie ist dafür zuständig, alle verfügbaren Module vom gefundenen Gadgeteer zu holen. Dafür steht eine Schnittstelle „/devices" bereit. Sie gibt in einer JSON-Nachricht alle Module zurück, die am Gadgeteer angeschlossen sind. Von jedem einzelnen wird ein Objekt mit dem Typ Device erstellt und einer Liste zugefügt. In einem abschliessenden Schritt teilt der Scout die Liste der Plattform mit. Der ganze Verlauf wird in der Methode „ScanNow" ausgeführt, welche im Sekundenintervall durchläuft.

Der GadgeteerScoutService dient der Konfigurationsseite als Schnittstelle zur Hauptklasse. So gelangen Informationen aus dem Programm in die HTML Seite.

5.1.2. Drivers

In den folgenden Kapiteln ist beschrieben, wie die einzelnen Driver umgesetzt wurden. Die Driver-Klassen fangen dabei alle mit DriverGadgeteer an. Um das Ganze ein wenig abzukürzen und verständlicher zu machen, wird das erwähnte Präfix jeweils weggelassen.

5.1.2.1. Base

Die Base-Klasse erbt von der Framework-Klasse ModuleBase und enthält deshalb alle Grundkomponenten eines Modules. Dazu gehören die Start-, Stop-, PortRegistered- und PortDeregistered-Methoden. Die Driver für die einzelnen Geräte erben von ihr, um an die Grundfunktionalitäten zu gelangen. In der selben Datei befindet sich jeweils eine DataContractAttribute-Klasse, welche für das Deserialisieren der HTTP-Responses zuständig ist. Diese wird hauptsächlich bei den Sensor-Drivers eingesetzt.

Weiter enthält die Klasse die Methode „WorkerThread". Sie wird in der Start-Methode als eigener Thread gestartet. Hier findet das Template Pattern seinen Einsatz. Es werden nämlich die beiden abstrakten Methoden „GetUrls" und „HandleGadgeteerResponse" aufgerufen. Die erste gibt die URL der Services zurück, die auf dem Gadgeteer aufgerufen werden sollen, und die zweite dient als Callback für den Aufruf. Das Ganze ist in der Lightsensor-Klasse sehr

gut nachvollziehbar. Wenn eine Klasse dieses Vorgehen nicht unterstützen will, kann sie die „WorkerThread“ Methode einfach überschreiben.

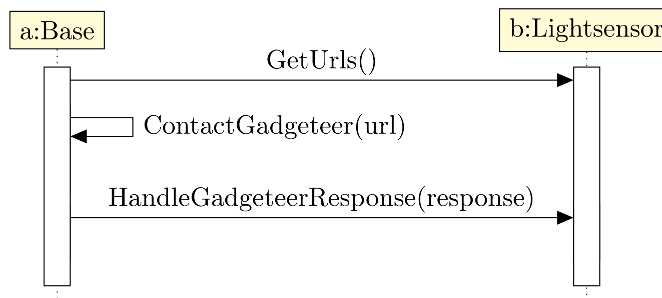


Abbildung 5.1.: Sequenzdiagramm: Template-Pattern im WorkerThread

Ebenfalls in der Start-Methode wird dem System mitgeteilt, welche Roles mit dem Port zur Verfügung stehen.

5.1.2.2. Lightsensor

In der Lightsensor-Klasse werden die Lichtsensordaten vom Gadgeteer geholt. Sie arbeitet am schönsten mit der Base-Klasse zusammen. In der „GetUrls“ Methode wird hier nur eine URL zurückgegeben. Die „HandleGadgeteerResponse“ Methode benachrichtigt nach dem Deserialsieren der HTTP-Response die Subscribers mit den neuen Lichtsensordaten.

Die „OnOperationInvoke“-Methode des Drivers ist dafür zuständig, direkte Aufrufe zu beantworten. Mit geschachtelten Switch-Statements auf die Parameter roleName und opName wird spezifisch nach Get-Aufrufen der RoleSensorMultiLevel-Role gefiltert. Beim Get-Aufruf mit dieser Role gibt die Methode die neusten Lichtsensordaten zurück.

5.1.2.3. Temperaturesensor

Die Klasse Temperatursensor funktioniert sehr ähnlich wie die Lightsensor-Klasse. Da das Gadgeteer Module TempHumid sowohl die Temperatur als auch die Luftfeuchtigkeit misst, werden in „GetUrls“ zwei URLs zurückgegeben. Die „HandleGadgeteerResponse“ benachrichtigt je nach Antwort-Typ die Subscriber mit einer anderen Role(RoleTemperatureSensor oder RoleHumiditySensor).

In der „OnOperationInvoke“-Methode wird gleich wie beim Lightsensor gefiltert. Jedoch wird hier wegen der Doppelfunktionalität des Drivers mit den Parametern roleName und opName, nach den Get-Methoden der beiden oben erwähnten Roles gefiltert.

5.1.2.4. LED

Die Hauptaufgabe des LED-Drivers ist es nicht, wie bei den Sensoren, Werte zurückzugeben, sondern auch Werte zu setzen. Die LEDs müssen ausgeschaltet und umgeschaltet werden können. Aus diesem Grund hat die „OnOperationInvoke“-Methode hier die wichtigste Rolle. Wie beim Lightsensor wird nach den Get- und Set-Operation der RoleLightColor-Role gefiltert. Über die Get-Operation werden die aktuellen LED Einstellungen abgefragt. Über die Set-Operation werden Einstellungen auf dem Gerät gesetzt. Dabei wird die einzustellende Farbe ans Gadgeteer weitergeleitet.

Wegen den verschiedenen Farbwerten einer Farbe konnte beim Deserialisieren nicht die Standard GadgeteerResponse geparsed werden. Deshalb wurde für die LEDs eine separate Response-Klasse erstellt. Da im WorkerThread der Base-Klasse die HandleGadgeteerResponse-Methode nur mit dem GadgeteerResponse-Typ aufgerufen werden kann, musste hier der WorkerThread überschrieben werden. Er benachrichtigt die Subscriber nun selbstständig und verzichtet vollständig auf die Methode HandleGadgeteerResponse.

5.1.2.5. Display

Die Klasse Display ist für das Setzen und Holen des Textes auf dem Display zuständig. Dieser Driver sieht sehr ähnlich aus wie bei den LEDs. Die „OnOperationInvoke“ Methode filtert nach Get-und Set-Operation der RoleDisplay-Role, um dann den aktuellen Text auf das Display zu holen oder den gewünschten Text auf dem Display zu setzen.

5.1.2.6. Camera

Die Kamera-Klasse muss nur Daten holen. Dabei handelt es sich um das aktuelle Bild, der am Gadgeteer angehängten Kamera. Die Bilder sind in Form eines Byte-Arrays abgespeichert. Aus dem gleichen Grund wie schon beim LED-Driver musste der „WorkerThread“ überschrieben werden.

Auch hier wird auf die HandleGadgeteerResponse-Methode verzichtet und die Subscriber werden direkt vom WorkerThread benachrichtigt.

Zur Entlastung des Gadgeteer Device läuft der WorkerThread in einem grösseren Zeitintervall durch.

5.1.3. App

5.1.3.1. GadgeteerWorkbench-Klasse

Die GadgeteerWorkbench-Klasse ist der Kernpunkt der Applikation. Sie erbt wie auch die DriverGadgeteerBase-Klasse von ModuleBase. In der Start-Methode werden die einzelnen Manager-Klassen, welche in Kapitel 5.1.3.4 beschrieben werden, instanziiert. Diese startet zudem einen WebFileHoster, welcher gebraucht wird um die Webseite zu hosten. Sie instanziiert auch den ServiceHost, welcher die WCF-Schnittstellen anbietet.

Sie wird vom System über das Hinzufügen oder das Entfernen eines Geräts benachrichtigt. Wie bereits bekannt ist, werden Geräte über Ports identifiziert. Das System teilt bei jeder Benachrichtigung der App mit, von welchem Port sie stammt. Die GadgeteerWorkbench-Klasse entscheidet, ob der Port der Appplikation hinzugefügt bzw. entfernt wird und ruft dabei über die entsprechende Manager-Klasse die richtige Methode auf.

In der Stop-Methode werden die noch offenen Handles geschlossen.Somit wird die Applikation sauber gestoppt.

5.1.3.2. IManager-Interface

Das IManager-Interface wird verwendet, um den Manager-Klassen einen einheitlichen Aufbau zu geben und somit der GadgeteerWorkbench-Klasse die Sicherheit zu geben, dass die benötigte Funktionalitäten in den Managern vorhanden sind. Folgende Methoden werden von ihm vorgeschrieben:

- Notify: Über diese Methode werden den Managern die neuesten Daten der Driver mitgeteilt.
- RegisterPort: Um zu wissen, welche Geräte zum aktuellen Zeitpunkt am System angemeldet sind, halten die Manager-Klassen eine Liste aller Geräte, welche ihrer Kategorie entsprechen. Mit RegisterPort wird das Gerät in die Liste eingetragen.
- DeregisterPortIfExists: Mit dieser Methode wird geprüft, ob sich das Gerät in der Liste befindet Wenn ja, wird es daraus entfernt.

5.1.3.3. Rules-Klassen

Im Rules-Ordner sind alle Klassen enthalten, die in Kapitel 4.5 geplant wurden. Neben den Klassen „Rules“, „ITrigger“, „InnerTrigger“ und „Action“, die als Platzhalter für die Regeln selber verantwortlich sind, findet man hier die Klassen „RuleFactory“ und „RuleSaver“.

Bei den Actions wurde ein Enum eingeführt, das zwischen Display und Led unterscheidet. Diese Unterscheidung ist nötig, da verschiedene Manager für die verschiedene Devices zuständig sind. In der „Execute“-Methode überprüft ein Switch-Statement, um welche Art von Device es sich handelt.

Die RuleFactory ist, wie es der Name schon sagt, eine Factory-Klasse für die Rules. Jede Rule, die gespeichert wird muss durch diese Klasse gehen. Alle Rules werden hier in einer Liste gespeichert. Weiter werden alle Trigger in einem Dictionary mit dem Devicename als Key gespeichert. Mithilfe dieses Dictionary können die verschiedenen Sensor-Manager alle Triggerwerte aktualisieren. Die Methode „UpdateSensorDatas“, die dafür zuständig ist, iteriert durch alle Trigger, die beim jeweiligen Driver gespeichert sind, und ruft dort die jeweilige „UpdateSensorValue“-Methode auf.

Die Klasse RuleSaver hingegen ist für das Speicher und Laden von Rules auf der Festplatte zuständig. Alle Rules werden von der RuleFactory geschickt, in JSON serialisiert und als String gespeichert. Dieser String wird beim Laden einfach wieder deserialisiert. Die RuleFactory braucht die zweite Funktionalität, um die Rules beim Start des Systems wieder zu laden.

Die Rules konnten im Endprodukt, wie es die Abbildung 5.2 zeigt, eingebaut werden.

Abbildung 5.2.: Regeln/Aktionen-Seite

5.1.3.4. Manager-Klassen

Bis auf die Klassen RuleManager und DatabaseManager implementieren alle Manager-Klassen das IManager-Interface. Die im Kapitel 5.1.3.2 beschriebenen Methoden müssen also von den Managern implementiert werden.

Wie bereits erwähnt wurde, wissen die Manager-Klassen, welche Geräte ihrer Kategorie im System verfügbar sind und speichern die aktuellen Daten. Es wurden Methoden definiert, um diese Informationen der Webseite zur Verfügung zu stellen. Die Methoden werden dafür vom GadgeteerWorkbenchService aufgerufen.

Da Sensordaten zusätzlich über Azure ausgewertet werden, definieren die Manager zusätzliche Methoden, um die Daten mittels dem DatabaseManager aus der Azure Datenbank zu lesen und als Liste zurückzugeben. Die erhaltenen Informationen wurden mittels eines Diagramms auf der Webseite dargestellt (siehe Abbildung 5.3).

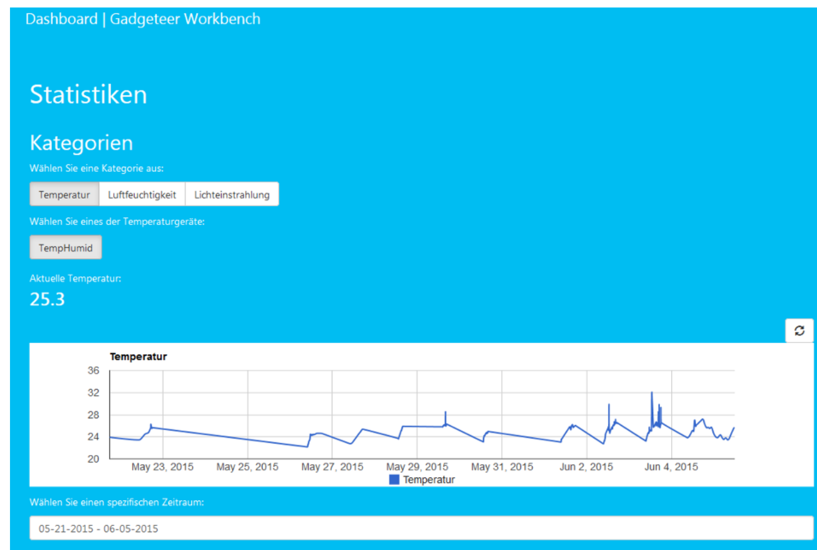


Abbildung 5.3.: Regeln/Aktionen-Seite

Der DatabaseManager hat nur eine Funktion, nämlich ReadMeasurements. Sie wird mit den Argumenten „Type" und „DeviceId" aufgerufen. Eine einfache SQL-Abfrage mit den beiden Argumenten in der Where-Klausel, fragt die Datenbank nach den relevanten Daten ab. Um die Datenbank Einträge im Programm zu repräsentieren, wurde eine Klasse MeasurementEntity erstellt, welche alle Werte eines Datenbank Eintrages beinhaltet. Mit dem SQLDataReader wird auf die Daten zugegriffen und pro Eintrag eine MeasurementEntity erstellt, welche dann in eine Liste eingefügt wird. Die Liste wird am Schluss der Methode zurückgegeben.

Der RuleManager verwaltet die Rules. Die GadgeteerWorkbenchService-Klasse greift auf diesen Manager zu, um entweder eine Liste von Rules zu holen, neue Rules hinzuzufügen, oder bestehende zu löschen. Jede Funktion, die von einem Service gebraucht wird und die Rules betreffen, befindet sicher in dieser Klasse.

5.1.3.5. GadgeteerWorkbenchService-Klasse

Die GadgeteerWorkbenchService-Klasse implementiert ein ServiceContractAttribute-Interface, welches sich in der gleichen Datei befindet, und wird als WCF-Schnittstelle gebraucht. Von ihr werden pro Aufruf die richtigen Methoden in den entsprechenden Manager-Klassen aufgerufen.

Die Schnittstellen werden von der Webseite aufgerufen und ermöglichen so die Kommunikation mit der Applikation.

5.1.3.6. Webseite

Die Webseite wurde mit HTML und AngularJS umgesetzt. Dabei stellt sie eine Single Page Application dar. Das vorgesehene Kachel-Layout wurde wie geplannte realisiert (siehe Abbildung 5.4). Die Seite ist unterteilt in einzelne Views, welche dynamisch eingebunden werden. In AngularJS werden Controller eingesetzt, um die BusinessLogik zu implementieren. Typischerweise wird pro View ein Controller erstellt, deshalb wurde dies hier auch so umgesetzt.

Für jede Teilfunktionalität der Webseite wurden also eine eigene View und ein eigener Controller erstellt. Die Controller fragen die darzustellenden Daten mit einer Polling-Strategie aus der Applikation ab. Sie bereiten die Daten auf und speichern sie im Scope. Der Scope wird als Bindemittel zwischen View und Controller verwendet. In der View kann somit definiert werden, welche Daten aus den Controllern, wie und wo dargestellt werden sollen.

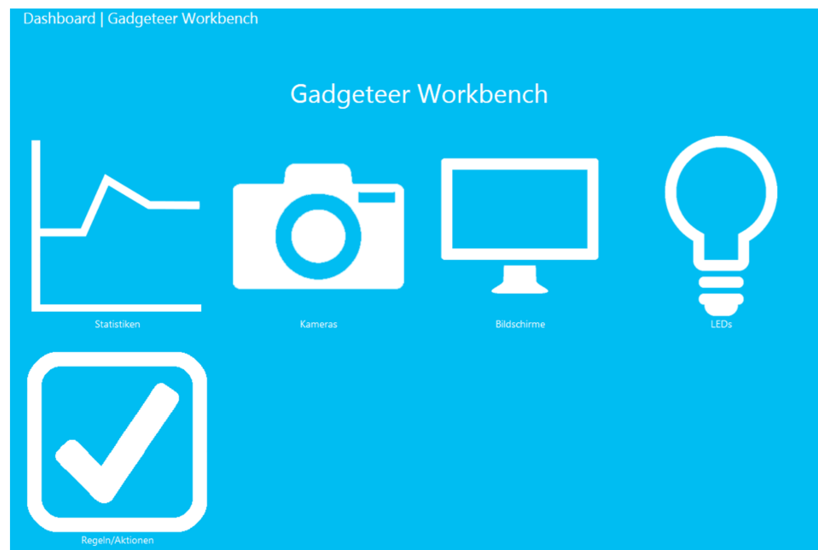


Abbildung 5.4.: Übersichtsseite (Kachel-Layout)

5.1.4. Probleme

5.1.4.1. Devices hinzufügen

Um ein Device hinzuzufügen, muss man unter „Add Devices“ auf eines der Devices in der Liste klicken und dann das Device konfigurieren. Wenn man die Konfigurationsseite jedoch verlässt, ohne das Device zu konfigurieren, dann kann dieses später nicht mehr hinzugefügt werden.

Das ganze Problem liegt leider im Zentrum des Frameworks in der Klasse Plattform. Dort befindet sich die Methode „StartDriverForDevice“. Sie ist für das Starten eines Devices zuständig. Ein Device wird bereits bei seiner Konfiguration gestartet, um zu überprüfen, ob es funktioniert. Dummerweise wird in dieser Methode das Device als konfiguriert abgespeichert. Gespeichert wird es in den Dateien Devices.xml, Modules.xml und Services.xml. Problematisch sind nur die beiden ersten Dateien. Das Device wird mit dem Parameter Configured=true gespeichert, was dem Scout zeigt, dass es schon konfiguriert ist und somit in der Übersicht nicht mehr angezeigt werden muss. Wird das Device als Module in Modules.xml gespeichert, so wird die App vom System über ein neu verfügbares Gerät informiert. Dies darf nicht vor dem Abschluss der Device-Konfiguration gemacht werden.

```
1 config.AddModule(moduleInfo);
2 config.UpdateDeviceDetails(device.UniqueName, true,
     driverFriendlyName);
```

So musste diese Methode verändert werden, damit die Devices beim Öffnen der Konfigurationsseite nicht abgespeichert werden. Zudem wurde eine zusätzliche Methode in GuiService.cs hinzugefügt, die aufgerufen werden kann, wenn die Devices fertig konfiguriert sind. Dort wird dann alles gemacht, was aus StartDriverForDevice gelöscht wurde.

5.1.4.2. Fehlschlagende Web-Callbacks

In der GadgeteerWorkbench App im LoT Framework werden verschiedene Services angeboten, die vom Presentation-Layer benutzt werden können, um auf die Daten in der Applikation zuzugreifen. Um auf diese Services zuzugreifen, benutzt der Presentation Layer eine Funktion, die Asynchrone Ajax-Webcalls durchführt. Die Antworten der Services werden anschliessend einer Callback-Funktion mitgegeben.

Bei der Entwicklung der Seite zur Administration von Rules wurde festgestellt, dass die Callback-Funktionen von mehreren aufeinander geschickten Webcalls nicht richtig aufgerufen wurden. Der Fehler wurde in der Datei common.js gefunden, wo der Ajax-Webcall durchgeführt wird. Dort versuchte der ursprüngliche Entwickler der common.js-Datei, das ganze „this"-Objekt im Call-Context zwischenzuspeichern, wie in folgendem Codeabschnitt zu sehen ist. Leider ist dies auf diese Weise nicht möglich und auch keine schöne Lösung.

```
1  var Url = this.Url;
2  var Data = this.Data;
3  var Callback = this.Callback;
4
5  return $.ajax({
6      url: Url,
7      data: Data,
8      context: this,
9  }).done(function (msg) {//On Successfull service call
10     SucceededServiceCallback(this, msg);
11 });
```

Mithilfe von Stackoverflow [17] wurde für das Problem eine schöne Lösung gefunden. Anstatt die ganze aktuelle Instanz zu speichern, wird ein Objekt mit allen benötigten Variablen im Context gespeichert. Nach der Implementation dieses Lösungsansatzes funktionierten die Callbacks wieder einwandfrei. Die korrekte Umsetzung ist im nächsten Codeabschnitt zu sehen.

```
1  var context = {
2      Url : this.Url,
3      Data : this.Data,
4      Callback : this.Callback
5  };
6
7  return $.ajax({
8      url: Url, // Location of the service
9      data: Data, //Data sent to server
10     context: context,
11 }).done(function (msg) {//On Successfull service call
12     SucceededServiceCallback(this, msg);
13 });
```

5.2. Gadgeteer

5.2.1. Setup

Ziel dieses Kapitels ist es, zu beschreiben, wie die einzelnen Module ans Mainboard angeschlossen sind, damit der Aufbau nachgestellt werden kann.

Das FEZ-Raptor Mainboard hat 18 Anschlüsse. Sie sind jeweils mit einer Nummer gekennzeichnet. Die Tabelle 5.1 beschreibt, welches Modul an welchen Anschluss angeschlossen wurde.

Anschluss	Modulbezeichnung
2	LightSense
3	Ethernet ENC28
4	2 * Smart Multicolor LED (DaisyLink)
6	Camera
8	USB Client DP
12	TempHumid SI70
13	LightSense
14	Display TE35 Touch
15	Display TE35 Red
16	Display TE35 Green
17	Display TE35 Blue

Tabelle 5.1.: Gadgteer Aufbau

5.2.2. DisplayWrapper

Das verwendete Display-Modul hat keine Methode, um den aktuell angezeigten Text zurückzugeben. Diese Funktionalität wurde mit der Display-Klasse erreicht. Über ein Property wird der Text gesetzt bzw. gelesen. Beim Setzten wird der Text auf dem Bildschirm angezeigt.

5.2.3. ConnectionManager

Der ConnectionManager ist das Kernstück des Netzwerkmanagements auf dem Gadgeteer. Der ConnectionManager hat mehrere Aufgaben. Er konfiguriert das Ethernet-Modul so, dass es sich mit dem Netz verbindet. Ausserdem ist er für das Senden der Beacons zuständig welche regelmässig geschickt werden. Mit diesen Beacons kann der Scout im LoT das Gadgeteer-Gerät erkennt. Des Weiteren reagiert der ConnectionManager auf Netzwerkänderungen und zeigt diese auf dem Display an. Zum Beispiel, wenn die IP-Adresse gesetzt oder geändert wird.

Der ConnectionManager stammt ursprünglich vom Prototyp 1, wurde jedoch sehr stark gekürzt und refactored.

5.2.4. EventHubManager

Der EventHubManager ist dafür zuständig in vorgegebenem Intervall die Messwerte der Sensoren zur Auswertung an die Schnittstelle von Windows Azure, dem Event Hub, zu senden. Zeitintervalls werden in Gadgeteer mit der Timer-Klasse realisiert. Die SendToAzure-Methode ist auf den Tick-Event des Timers registriert und wird alle 10 Sekunden ausgeführt. Damit die Authentifizierung beim Event Hub richtig funktioniert, muss beim Senden der Daten eine gültige Zeit mitgeschickt werden. Da Gadgeteer keine Batterie auf dem Mainboard hat,

und vollständig über ein Modul zu Energie kommt, hat es keine Systemzeit. Die Zeit wurde schlussendlich mit einem Zeitserver synchronisiert. In der SendToAzure-Methode wird sichergestellt, dass eine Netzwerkverbindung besteht, damit die Zeit synchronisiert ist und eine AMQP-Verbindung aufgebaut ist. Sind alle Bedingungen erfüllt, werden die Daten in der SendMessage-Methode mittels AMQP an Azure geschickt. Beim Auftreten einer Exception wird sich das System in einen nicht wiederherstellbaren Zustand versetzt und muss deshalb neu gestartet werden. Dies passiert jedoch sehr selten. Ein Neustart dauert kaum 10 Sekunden.

5.2.5. IdentifierSetting

Da die Module selber keine eindeutige Bezeichnung haben, wurden mittels eines GUID-Generator für jedes Modul eindeutige Kennzeichnungen generiert. In der IdentifierSetting-Klasse sind diese abgespeichert, so dass von überall her bequem auf sie zugegriffen werden kann.

5.2.6. WebEventManager

Der WebEventManager kümmert sich, wie es der Name schon andeutet, um das Behandeln der Web-Events. Damit die Webserver-Instanz entscheiden kann, ob ein neu eintreffender Request verarbeitet werden soll, hält er eine Hashtable der verfügbaren WebEvents. Mittels eines Strings definiert ein WebEvent seinen zu behandelnden Pfad. Der Webserver überprüft bei jedem Request, ob ein WebEvent den aufgerufenen Pfad bearbeitet. Wenn ja, wird der _WebEventReceived-Event des entsprechenden WebEvents aufgerufen. Wenn eine Handler-Methode sich auf dem Event registriert hat, wird dieser nun aufgerufen. Ansonsten wird die Standard-Response zurückgeschickt.

Der WebEventManager erstellt die einzelnen WebEvents, registriert die Handler-Methoden und definiert diese zugleich.

5.2.7. Probleme

5.2.7.1. Webserver Deadlock

Die Kommunikation zwischen LoT und Gadgeteer wird mithilfe eines Webservers erledigt, welcher unter Gadgeteer läuft. Dieser wurde von LoT zur Verfügung gestellt und ist im Source Code als „HomeOSGadgeteerLibrary“ enthalten. Leider verklemmt sich dieser manchmal, so dass keine neuen Verbindungen mehr geöffnet werden können und das Gerät neu gestartet werden muss. Es hat sich herausgestellt, dass die Client Sockets nicht richtig geschlossen werden. Darum wurde der Code mittels eines try-finally so angepasst, dass die Sockets richtig geschlossen werden.

5.2.7.2. Exceptions beim Versenden von Bildern

Bei der Abfrage von Kamerabildern wurden im Gadgeteer regelmässig Exceptions geworfen. Diese Exceptions traten beim Durchlaufen des Codes mit einem Debugger allerdings nicht mehr auf, was darauf hinwies, dass etwas zu schnell gemacht worden war.

Es hat sich ergeben, dass die Sockets in der Webserver-Implementation zu früh geschlossen wurden. Jeweils vor dem Senden der Nachrichten wurde ein Flag gesetzt, dass die Nachricht nun geschickt worden war.

Beim Senden von Textnachrichten war das kein Problem, da die Daten gerade noch rechtzeitig über den Socket gesendet werden konnten. Leider war dies bei den grossen Bildnachrichten nicht der Fall.

Der Code wurde darum so angepasst, dass das Flag erst nach dem Senden gesetzt wurde. Das Problem konnte somit behoben werden.

5.2.7.3. Camera - PictureCaptured[5]

Nach Auslösen der TakePicture Methode vom Camera Objekt wird mit der Kamera ein Bild gemacht. Dieses wird, bevor der PictureCaptured-Event ausgelöst wird, intern in eine Bitmap umgewandelt. Diese Konvertierung führt zu einer „System.ArgumentException“ und bringt die Anwendung zum Absturz.

Hier ein Codebeispiel, um den Bug zu reproduzieren drückt man lediglich den Button:

```
1  void ProgramStarted()
2  {
3      Debug.Print("Program Started");
4      button.ButtonReleased += button_ButtonReleased;
5      camera.PictureCaptured += camera_PictureCaptured;
6  }
7  void camera_PictureCaptured(Camera sender, GT.Picture e)
8  {
9      displayTE35.SimpleGraphics.DisplayImage(e, 5,5);
10 }
11 void button_ButtonReleased(Button sender, Button.ButtonState
       state)
12 {
13     camera.TakePicture();
14 }
```

5.3. Azure

5.3.1. Event Hub

Der Event Hub wurde wie geplant als Sammelbecken der Sensordaten eingesetzt. Er wurde genau gleich eingerichtet, wie es im Kapitel 3.3.2.1 beschrieben ist.

5.3.2. Stream Analytics

Der Stream Analytics-Service hat sich im Verlgeich zum Prototyp 3 nur leicht verändert. Nämlich wurde folgender Query verwendet:

```
1 SELECT DateAdd(minute,-30,System.TimeStamp) as WinStartTime,
     system.TimeStamp as WinEndTime, DeviceId, Type, Unit, Avg(
     Value) as AvgValue, Count(*) as EventCount
2 FROM
3     input
4 GROUP BY TumblingWindow(minute, 30), DeviceId, Type, Unit
```

Mit dem TumblingWindow wird im eingestellten Zeitintervall der Durchschnitt über die gesammelten Sensorwerte berechnet. Dabei werden die Sensoren nach DeviceId, Type und Unit gruppiert.

5.3.3. SQL Database

Halbstündlich werden vom Stream Analytics-Service die berechneten Werte in die Datenbank gespeichert. Folgende Befehle wurden für die Konfiguration der Datenbankstruktur verwendet:

```
1  CREATE TABLE [dbo].[AvgMeasurements] (
2      [WinStartTime]   DATETIME2 (6) NULL,
3      [WinEndTime]     DATETIME2 (6) NULL,
4      [DeviceId]      BIGINT NULL,
5          [Type] VARCHAR(255) NULL,
6          [Unit] VARCHAR(255) NULL,
7      [AvgValue] FLOAT (53)    NULL,
8      [EventCount] BIGINT null
9  );
10
11 GO
12 CREATE CLUSTERED INDEX [AvgMeasurements]
13     ON [dbo].[AvgMeasurements]([DeviceId] ASC);
```

6. Abschluss

6.1. Verbesserungen

6.1.1. Rules

Die aktuelle Umsetzung der Rules ermöglicht es dem Benutzer, einen Sensorwert mit einer Zahl zu vergleichen und die Action zu definieren, welche beim Zutreffen der Bedingung ausgelöst wird. Es wäre aber sinnvoll, weitere Triggers einzuführen, bei denen Sensorwerte mit Zeichenketten verglichen werden können oder vielleicht sogar auf das Datum oder die Uhrzeit zu schauen,um den Trigger auszulösen. Eine weitere Verbesserungsmöglichkeit wäre die Einführung von Outer Triggers, damit mehrere Triggers aneinander gekettet werden können. Eine andere Erweiterung wären Action-Listen, damit mehreren Actions ausgelöst werden können. Die entsprechende Grammatik aus 4.5 würde mit folgenden Einträgen erweitert werden:

$$
\begin{array}{ll}
\text{ITrigger} & \rightarrow \text{Sensor} + \text{Comparer} + \text{SValue} \\
 & \rightarrow \text{OTrigger} \\
\text{OTrigger} & \rightarrow \text{ITrigger} + \text{Operation} + \text{ITrigger} \\
\text{Operation} & \rightarrow \wedge \\
 & \rightarrow \vee \\
 & \\
\text{Action} & \rightarrow \text{Device} + \text{DValue} \\
 & \rightarrow \text{Action} + \text{Action}
\end{array}
$$

Weiter ist zu bemängeln, dass der Zustand sich jedes Mal deaktiviert, wenn der Trigger ausgelöst und die Action durchgeführt wurde. Als mögliche Verbesserung könnte man hier noch den Zustand „dauerhaft aktiviert“ einbauen.

6.1.2. GadgeteerResponse-Klasse anpassen

Diese GadgeteerResponse-Klasse dient der Deserialisierung von HTTP-Responses. In einigen Fällen weichen die verschickten JSON-Daten von der Struktur dieser Klasse ab. Daher müssten dann jeweils ganze Codeblöcke überschrieben werden. Die Klasse müsste dringend so angepasst werden, dass verschiedene Datentypen damit geparsed werden können. Für Nummern und Strings könnten jeweils eigene Variablen deklariert werden. Dann müssten nur noch die JSON Strings so angepasst werden, dass diese auf die vorgegebene Struktur passen und schon könnte der WorkerThread von der DriverGadgeteerBase-Klasse verwendet werden.

6.1.3. Temperatur-Driver aufspalten

Das TempHumid Modul von Gadgeteer liefert sowohl die Temperatur als auch die Luftfeuchtigkeit, momentan wird für die Abfrage dieser Daten nur ein Driver verwendet. Eine benötigte Verbesserung wäre es die Elemente für die Luftfeuchtigkeit in einen separaten Driver zu extrahieren.

6.1.4. Gadgeteer Architektur überarbeiten

Die Archtitekur im Gadgeteer-Projekt ist nicht ganz sinnvoll und sollte dringend überarbeitet werden.

6.1.5. Konsequente Verwendung von DeviceIDs

Im Moment werden im System DeviceIDs verwendet, um LEDs und Lichtsensoren zu spezifizieren. Dies wird so gemacht weil mehrere dieser Module am Gadgeteer angeschlossen sind. Um das ganze konsequent zu gestalten, müssten die IDs nicht nur dort verwendet werden sondern bei allen Modulen. Dadurch wäre es dann möglich, mit mehrere Temperatursensoren oder auch mehreren Kameras zu arbeiten.

6.2. Fazit

Das Ziel der Arbeit war es mit LoT und Gadgeteer eine Internet of Things-Anwendung zu entwickeln. Das Projekt konnte erfolgreich zu Ende geführt werden. Herausgesprungen ist dabei ein äusserst dynamisches System, welches es in einem flachen Web-Interface ermöglicht, die Hardware Plattform vollständig zu steuern und Sensordaten anzuzeigen. Die Sensoren können zur Laufzeit hinzugefügt oder entfernt werden, was das System besonders flexibel macht.

Die entwickelte Applikation umfasst zudem die in 4.5 beschriebene Rule-Engine. Damit konnte ein abgespeckter aber durchaus realistischer Show-Case, wie er auch in einem produktiven System eingesetzt würde, veranschaulicht werden.

Mit der Abwicklung dieses Projekts konnten die eingesetzten Technologien untersucht werden und deren Eignung abgeklärt werden. Dabei hat sich herausgestellt, dass Internet of Things-Anwendungen ein riesiges Potential bergen und garantiert in naher Zukunft ihren Einsatz in den Haushalten finden.

Leider musste festgestellt werden, das Lab of Things noch nicht bereit ist, um produktiv auf diesem Gebiet eingesetzt zu werden. Die Dokumentation des Frameworks ist unvollständig und mehrere Komponenten enthalten Bugs oder verwenden veraltete Libraries.

Gadgeteer eignet sich zwar wunderbar für experimentelle Projekte, aber auch dort sind noch Bugs zu beheben. Wegen des modularen Aufbaus der Plattform, war es bei Fehlern sehr schwierig, wenn nicht unmöglich, auf Leute mit gleichen Problemen zu stossen.

Die Microsoft Azure-Technologien sind trotz ihrer Neuheit extrem zuverlässig und mit Abstand die stabilsten Komponenten im System. Sie eignen sich absolut für den Einsatz mit Internet of Things und haben ein sehr grosses Potential.

Abbildungsverzeichnis

2.1. Azure Event Hub: Event Publisher/Consumer Übersicht 12
2.2. Azure Event Hub: Partitionen 13
2.3. Azure Stream Analytics: Übersicht 14
2.4. UML Diagramm: Show cases 15

3.1. Screenshot: Anzeige der Temperaturen 19
3.2. Sequenzdiagramm: Gadgeteer erkennen 20
3.3. Screenshot: Anzeige der Lichtstärke in LoT 22
3.4. Komponentendiagramm: Prototyp 3 23
3.5. Screenshot: Event Hub erstellen 25

4.1. Komponentendiagram: Endsystem 27
4.2. Ordnerstruktur: Admin Tool 28
4.3. Sequenzdiagramm: Gadgeteer erkennen 30
4.4. Sequenzdiagramm: Requests von LoT zu Gadgeteer 32
4.5. Sequenzdiagramm: Messwerte an Azure senden 32
4.6. Sequenzdiagramm: Benachrichtigungen vom Driver 33
4.7. Sequenzdiagramm: Driver ansprechen 33
4.8. Sequenzdiagramm: Webseite zu Azure 33
4.9. Wireframe: Übersicht 37
4.10. Wireframe: Statistiken 38
4.11. Wireframe: Kameras 39
4.12. Wireframe: Bildschirme 39
4.13. Wireframe: LEDs 40
4.14. Wireframe: Regeln/Aktionen 41

5.1. Sequenzdiagramm: Template-Pattern im WorkerThread 46
5.2. Regeln/Aktionen-Seite 48
5.3. Regeln/Aktionen-Seite 49
5.4. Übersichtsseite (Kachel-Layout) 50

Literatur

[1] *Amqp.Net Lite*. URL: http://amqpnetlite.codeplex.com/.

[2] *Bootstrap · The world's most popular mobile-first and responsive front-end framework.* 2014. URL: http://en.wikipedia.org/w/index.php?title=Bootstrap_(front-end_framework)&oldid=637483184.

[3] *Event Hubs - Cloud big data solutions*. URL: http://azure.microsoft.com/en-us/services/event-hubs/.

[4] *Event Hubs Overview*. URL: https://msdn.microsoft.com/en-us/library/azure/dn836025.aspx.

[5] *Forum | camera ConvertToFile exception - GHI Electronics*. URL: https://www.ghielectronics.com/community/forum/topic?id=18615.

[6] *Get started with Stream Analytics: Real-time fraud detection | Microsoft Azure*. URL: https://azure.microsoft.com/en-us/documentation/articles/stream-analytics-get-started/.

[7] *Getting Started with Lab of Things*. 2015. URL: https://labofthings.codeplex.com/wikipage?title=Getting%20Started%20with%20Lab%20of%20Things.

[8] *GHI Electronics*. URL: https://www.ghielectronics.com/.

[9] *Home | AMQP*. URL: https://www.amqp.org/.

[10] *Lab of Things Code*. URL: https://labofthings.codeplex.com/SourceControl/latest.

[11] *Lab of Things Software Architecture*. 2015. URL: https://labofthings.codeplex.com/wikipage?title=HomeOS%20Software%20Architecture.

[12] Microsoft. *Microsoft Research: The Lab of Things*. 2015. URL: http://www.lab-of-things.com/ (besucht am 19. 02. 2015).

[13] *Microsoft Azure: Cloud Computing Platform & Services*. URL: http://azure.microsoft.com/en-us/.

[14] *.NET Gadgeteer Socket Types*. URL: http://gadgeteer.codeplex.com/wikipage?title=.NET%20Gadgeteer%20Socket%20Types&referringTitle=Documentation.

[15] *.NET Micro Framework installieren*. URL: https://www.ghielectronics.com/support/netmf.

[16] *SQL Database - Relational database service*. URL: http://azure.microsoft.com/en-us/services/sql-database/.

[17] *Stackoverflow: Ajax Bug*. URL: http://stackoverflow.com/questions/30578658/what-is-the-right-way-to-use-context-in-an-ajax-call.

[18] *Stream Analytics - Real-time data analytics*. URL: http://azure.microsoft.com/en-us/services/stream-analytics/.

[19] *Windows Azure*. URL: https://manage.windowsazure.com.

Glossar

AMQP Advanced Message Queuing Protocol. 6

Event Hub Der Event Hub ist eine Service-Bus Technologie von Microsoft Azure, welche speziell dafür entwickelt wurde, um grosse Mengen von Daten übers Internet zu sammeln. 12

LoT Lab of Things. 8

Stream Analytics Stream Analytics ist ein Data-Service von Microsoft Azure. Es können damit grosse Datenmengen in Echtzeit analysiert und verarbeitet werden. 13