Herman Hollerith Zentrum

Hochschule Reutlingen
Reutlingen University

UCC@Retail

Project - Dokumentation

Studiengang: Digital Business Engineering

Betreuer: Prof. Dr. Christian Decker
Sebastian Kotstein

Verfasser: Akin Kücük (767948)
Benjamin Traub (767918)
Fethullah Misir (767704)
Hasan Akhuy (767312)

Datum: 27.10.2020

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

1. Problemstellung

Das Projekt „UCC@Retail“ wurde im Rahmen der Module Project 1 und 2 in Zusammenarbeit mit Kaufland/Schwarz IT im Studiengang Digital Business Engineering am Herman-Hollerith-Zentrum bearbeitet. Derzeit existieren für Kaufland-Mitarbeitern in den Filialen keine Möglichkeiten, Informationen über verschiedene Geräte, wie z.B. Waage oder Pfandflaschenautomaten abzurufen. Ist der Behälter eines Pfandflaschenautomaten überfüllt, nimmt dieser keine Flaschen mehr an. Infolgedessen müssen die Kunden darauf warten, bis der Behälter von einem Mitarbeiter geleert wird. Ein anderes Beispiel, bei dem es zu Warteschlangen kommen könnte, ist die Waage in der Obst- und Gemüseabteilung. Ist hier z.B. das Thermopapier leer, müssen Kunden wieder warten, bis das Thermopapier ausgetauscht wird. Diese Beispiele führen zu Verzögerungen und Unzufriedenheit bei den Kunden sowie zusätzlichen Stress für Mitarbeiter, da diese schnell reagieren müssen.

2. Motivation und Zielstellung

Um in Zukunft die oben beschriebenen Situationen zu vermeiden, sollen die Kommunikationsprozesse in der Filiale verbessert werden. Hierfür sollen Geräte in die Kommunikationslandschaft eingebunden werden, um so eine Mensch-zu-Maschine Kommunikation zu ermöglichen. Dadurch soll ermöglicht werden, dass die Waage beispielweise kurz bevor das Thermopapier leer ist, den Mitarbeiter informiert und dieser dementsprechend reagieren kann, indem er die Rolle rechtzeitig austauscht. Außerdem soll für die Kommunikation mit verschiedenen Geräten eine einzige Benutzeroberfläche verwendet werden, anstatt für jedes Gerät eine separate Benutzeroberfläche zu haben. Der Lösungsansatz für diese Zielstellung lautet Unified Communication & Collaboration (UCC). Von besonderer Bedeutung ist in diesem Rahmen die Einbindung von Maschinen in das herkömmliche UCC-Konzept, welches sich stark auf die Mensch-zu-Mensch Kommunikation fokussiert. Die Vision, dass eine Kommunikation zwischen Menschen und Maschine im Rahmen des UCC-Konzepts ermöglicht wird, soll über ein adaptiertes UCC-Produkt realisiert werden.

3. Unified Communication & Collaboration (UCC)

Unter dem Begriff UCC wird die Bündelung verschiedener Kommunikationsdienste zusammengefasst, die dezentrales Arbeiten ermöglichen und den Kommunikationsprozess verbessern (vgl. [UCC18]). Damit wird auf eine steigende, zunehmend komplexe Zahl von Kommunikationsmöglichkeiten reagiert (vgl. [UCC18]). Unified Communications (UC) zeichnet sich im Wesentlichen durch zwei Aspekte aus (vgl. [UCC18]):

1. Anwender müssen einer bereits stattfindenden Kommunikation neue Darstellungsformen hinzufügen können, z.B. wenn aus einem Chat heraus eine Kameraübertragung gestartet wird
2. Kommunikation unabhängig vom Aufenthaltsort, z.B. über einheitliche Rufnummer

Das zusätzliche „Collaboration“ in UCC stellt Erweiterungen dar, die zusätzlich zu den Kommunikationsmöglichkeiten Werkzeuge anbieten, um die Zusammenarbeit zu fördern (vgl. [UCC18]). Beispiele hierfür sind virtuelle Whiteboards oder auch Bildschirmübertragungen.

3.1 UCC – Features

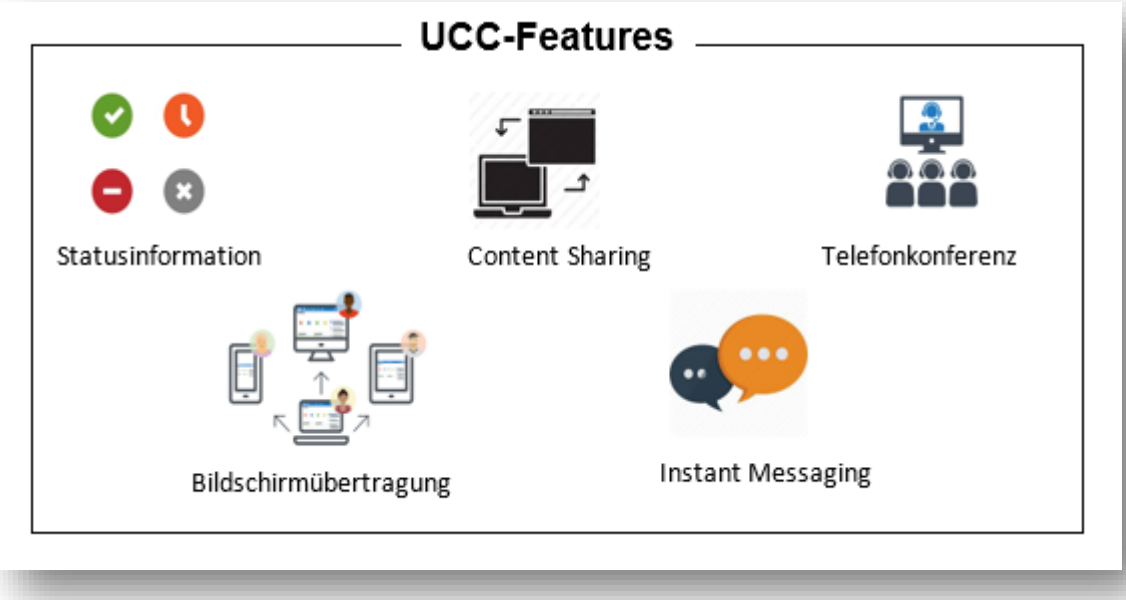


Abbildung 1 UCC Features

- **Statusinformation:** zeigt an, ob eine Person on- oder offline, beschäftigt, in einer Besprechung ist uvm.
- **Bildschirmübertragung:** Übertragung des eigenen Bildschirms, um Inhalte für andere sichtbar zu machen.
- **Content Sharing:** Senden von Dateien, z.B. Excel oder PDF.
- **Instant Messaging:** Austausch von Nachrichten durch Chatten
- **Telefonkonferenz:** Kommunikation von mehreren Personen über IP-Telefonie

3.2 UCC – Produkte

Abbildung 2 Beispiele UCC-Produkte

Mittlerweile existieren einige UCC – Produkte, die verschiedene UCC – Features (siehe 3.1 UCC – Features) kombinieren.

3.3 Grenzen heutiger UCC Lösungen

Die im Vorfeld dargelegten UCC-Lösungen konzentrieren sich in erster Linie auf die Mensch-zu-Mensch Kommunikation. Das bedeutet, dass mehrere Personen mithilfe eines UCC-Produkts (vgl. 3.2 UCC – Produkte) miteinander kommunizieren können, indem sie die oben genannten UCC-Features (vgl.3.1 UCC – Features) verwenden. Durch die zunehmende Digitalisierung in vielen Bereichen des Lebens und die steigende Anzahl an IoT-fähigen Geräten auf dem Markt, rückt die Kommunikation zwischen Menschen und Maschine immer mehr in den Vordergrund. Diese Tatsache

wird jedoch nicht durch heutige UCC-Produkte abgedeckt. Da es gerade in einer Filiale immer mehr solcher digitalen Komponenten wie beispielsweise moderne Waagen gibt, welche in der Lage sind, Daten über das Internet zur Verfügung zu stellen, existiert das Interesse, diese Geräte in das herkömmliche UCC-Konzept miteinzubinden.

4. Anwendungsfall

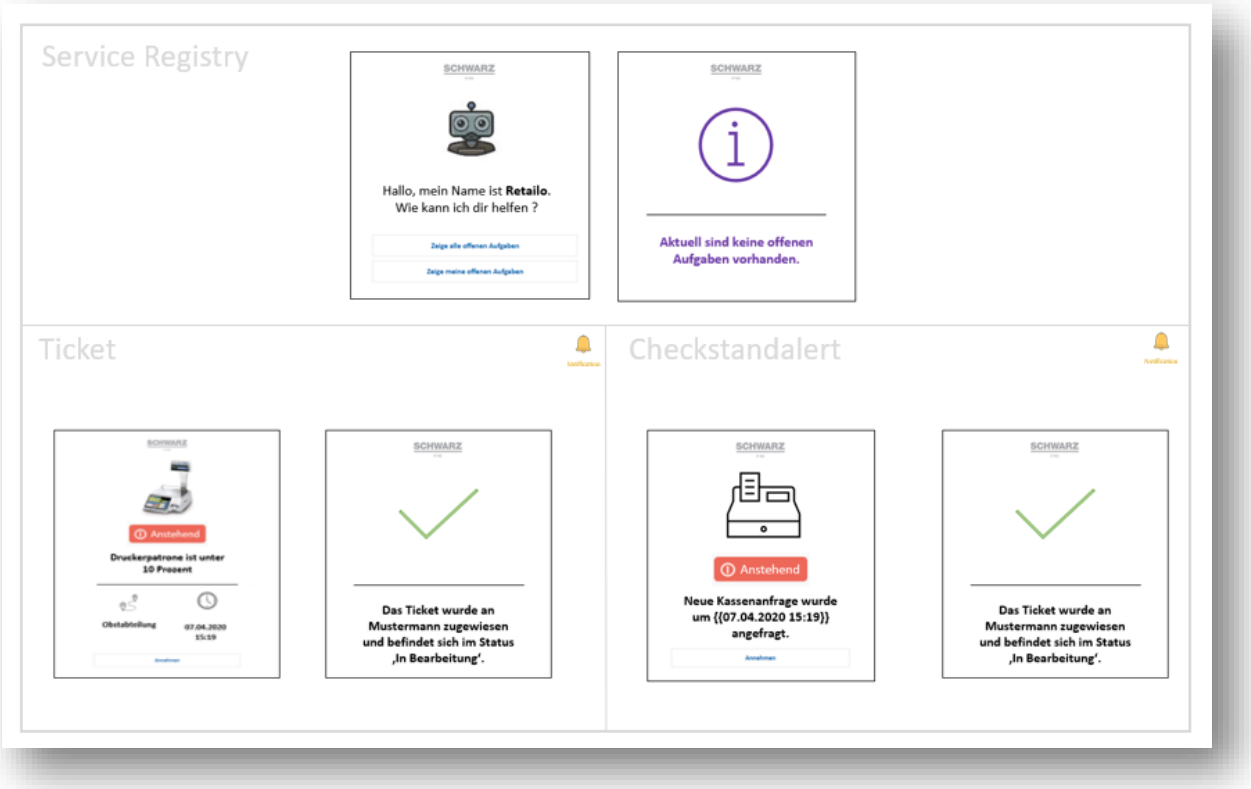


Abbildung 3 Anwendungsfälle

Im Projekt werden unterschiedliche Anwendungsfälle in Betracht gezogen. Dabei soll die Anwendung auf Anwenderinteraktionen reagieren, sowie proaktiv Aufgaben von IoT-Geräten erhalten und diese in Form von Aufgaben einer Microsoft Teams Gruppe bereitstellen.

Der erste Anwendungsfall ist die Registry. Der Anwender kann den Bot mit einer beliebigen Textnachricht anschreiben, dieser reagiert auf die Textnachricht und zeigt alle aktuellen Services in einem Adaptive Card an. Die Registry ist fokussiert auf die offenen Aufgaben für den jeweiligen Service.

Ein weiterer Anwendungsfall ist die proaktive Aufgabenerstellung durch die angeschlossene Waage. Die Waage ist in der Regel mit einem Gateway verbunden,

welches dann die Daten für ThingsBoard bereitstellt. Für das Projekt wird die Anfrage aus der Waage mittels Postman simuliert. Dabei wird eine Aufgabe mit Detailinformationen erstellt und an die Interessenten via Microsoft Teams übermittelt. Die Anwender in der Gruppe können dann auf diese Nachrichten reagieren und bestenfalls die Aufgabe übernehmen. Bei erfolgreicher Zuweisung erhält der Anwender eine Bestätigungsnachricht und kann diese in seinen Aufgaben wiederfinden.

Der letzte Anwendungsfall deckt die Anfrage eines neuen Kassenaufrufs ab. Die Grundidee hierfür ist, mit Hilfe eines Pushbuttons eine Aufgabe an die Filialmitarbeiter zu versenden, wenn eine neue Kasse für den Checkout bereitgestellt werden soll. Wie im vorherigen Anwendungsfall, kann der Mitarbeiter dann die Aufgabe übernehmen und den Prozess abschließen.

5. Konzeption

In diesem Kapitel wird auf die Referenzarchitektur und die Schnittstellen im Detail eingegangen. Es werden Konzepte beschrieben, die in Kapitel 6. Umsetzung für die Implementierung verwendet werden.

5.1 Referenzarchitektur

Die Architektur besteht aus mehreren Komponenten, von denen einige vorgegeben sind, da diese bereits in der Schwarz IT eingesetzt werden. Die vorgegebenen Komponenten sind:

- Microsoft Teams als UCC Lösung
- ThingsBoard als IoT-Plattform
- Kafka für Nachrichtenaustausch

Die Referenzarchitektur bestimmt den technischen Leitfaden, um den Kommunikationsbruch zwischen Menschen und Maschine in Filialen zu eliminieren. Erweiterbarkeit um neue Funktionalitäten und eine lose Kopplung zwischen den Komponenten sind eine primäre Anforderung für das Design der Referenzarchitektur. Die Aufgaben der einzelnen Komponenten müssen klar definiert sein und dürfen Ihren

Rahmen nicht sprengen. Beispielsweise soll ein Bot keinen Workflow kennen. Die Aufgabe des Bots ist es, Benutzeranfragen entgegenzunehmen und zu delegieren.

Die nachfolgende Abbildung stellt die Referenzarchitektur dar:

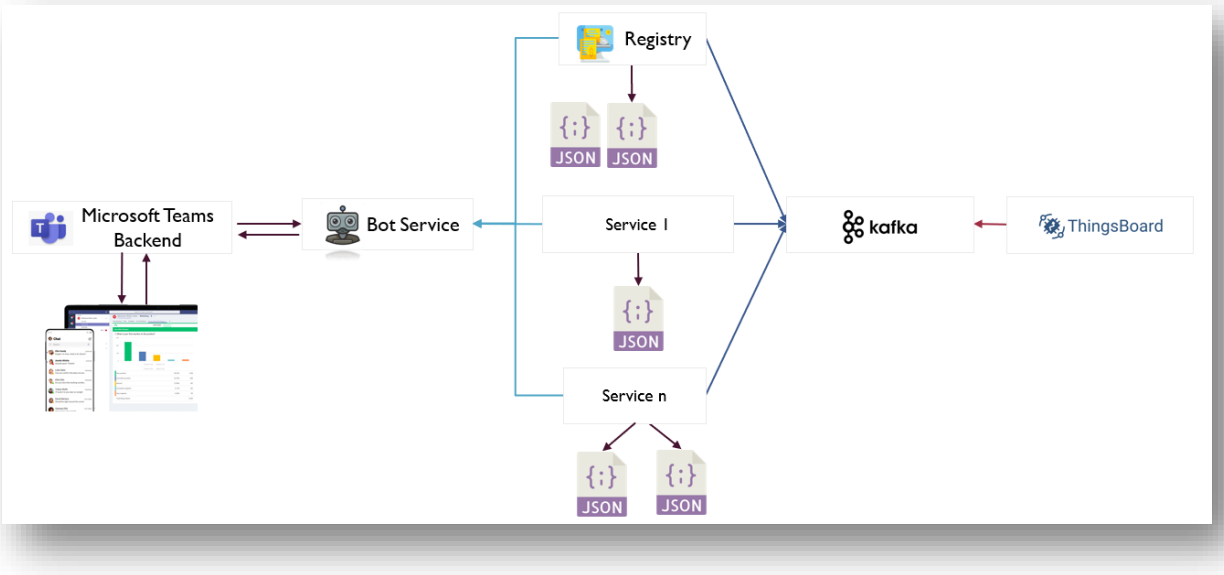


Abbildung 4 Referenzarchitektur

Microsoft Teams dient als Integrationskomponente zwischen den Benutzern und der Architektur in der zugrundeliegenden Lösung. Benutzer interagieren dabei mit einem Chatbot. Alle Interaktionen mit dem Chatbot werden von Microsoft Teams an den Bot Service weitergeleitet. Die Referenzarchitektur setzt auf die von dem Anwendungsfall abgeleiteten Kommunikationswege an. Diese sind die interaktive und benachrichtigungsbasierte Kommunikation. Benutzer können über den Teams Client interaktiv mit dem Bot Service kommunizieren. Informationen können als Benachrichtigung an alle mit dem Chatbot verbundenen Benutzer übermittelt werden.

Die Bot Service Komponente hat als primäre Aufgabe, Anfragen der Benutzer an die zuständigen Komponenten zu delegieren. Außer der Registry kennt der Bot Service keine andere Komponente. Dadurch wird gewährleistet, dass neue Komponenten eingebettet werden können, ohne die Notwendigkeit einer Änderung im Source Code oder in der Konfiguration des Bot Services.

Um die Anforderungen zu erfüllen, müssen zwei Konzepte verwendet werden.

1. Der Bot Service weiß zur Laufzeit, zu welcher Komponente die Anfrage delegiert werden soll

2. Der Bot Service stellt keine Use-Case spezifischen Assets wie Adaptive Cards oder Bilder zur Verfügung

Zum Erfüllen der ersten Anforderung wird das Hypermedia as the Engine of Application State (HATEOAS) Prinzip angewendet. In HATEOAS navigiert der Client ausschließlich über URLs, die vom Server zur Verfügung gestellt werden. Clients brauchen außer einer Einstiegs-URL keine weiteren Informationen (vgl. [HAT20]). Das Prinzip wird durchgängig im Web verwendet. Wird eine Webseite mit dem Browser angefragt, erhält der Browser ein HTML Dokument, in der die URLs zu weiteren Seiten hinterlegt sind. Der Browser muss lediglich über die Links innerhalb des Dokuments navigieren und keine weiteren Details einer spezifischen Seite kennen.

Die Kommunikation zwischen dem Bot Service und den Services basiert auf diesem Konzept. Der Bot Service erhält JSON Dokumente nach dem Adaptive Cards Schema. Die URLs sind in den Adaptive Cards hinterlegt, wodurch eine Navigationsmöglichkeit für den Bot Service entsteht.

Beispielsweise kann ein Service eine Adaptive Card Definition mit einem „Annehmen“-Button zur Verfügung stellen. In der Definition ist hinterlegt, welche URL mit welchen Daten aufgerufen werden soll, wenn auf das „Annehmen“-Button geklickt wird. Mit diesen Informationen weiß der Bot Service, an welche Komponente eine Anfrage weitergeleitet werden muss und mit welchem Inhalt.

Die zweite Anforderung wird erfüllt, in dem die Services dem Bot Service ausschließlich Adaptive Cards senden. Der Bot Service kann die Adaptive Cards direkt an Microsoft Teams weiterleiten, ohne den Inhalt kennen zu müssen. Bilder können als URLs referenziert werden, die Microsoft Teams automatisch auflöst.

ThingsBoard ist die zentrale Komponente für die Integration von IoT-Devices. Mittels ThingsBoard erhalten IoT-Devices eine digitale Repräsentation, an welche die Devices ihre Daten übermitteln können. Über Regelwerke kann in ThingsBoard bestimmt werden, was mit den Daten passieren soll. Um ebenfalls eine Kommunikation zwischen ThingsBoard und den Services zu ermöglichen, werden Nachrichten über Messaging mittels Kafka übertragen. Dadurch muss ThingsBoard die umliegenden Komponenten nicht kennen, sondern nur an Kafka Topics Daten schicken. Services registrieren sich an die Kafka Topics und erhalten die Nachrichten. Neben der losen Kopplung hat dieser Ansatz einen weiteren entscheidenden Vorteil. Bei einer

vorübergehenden Unerreichbarkeit eines Services gehen keine Daten verloren, da sie in Kafka gehalten werden.

Das nachfolgende Sequenz Diagramm stellt die beschriebene Interaktion vom Device bis zum Benutzer dar:

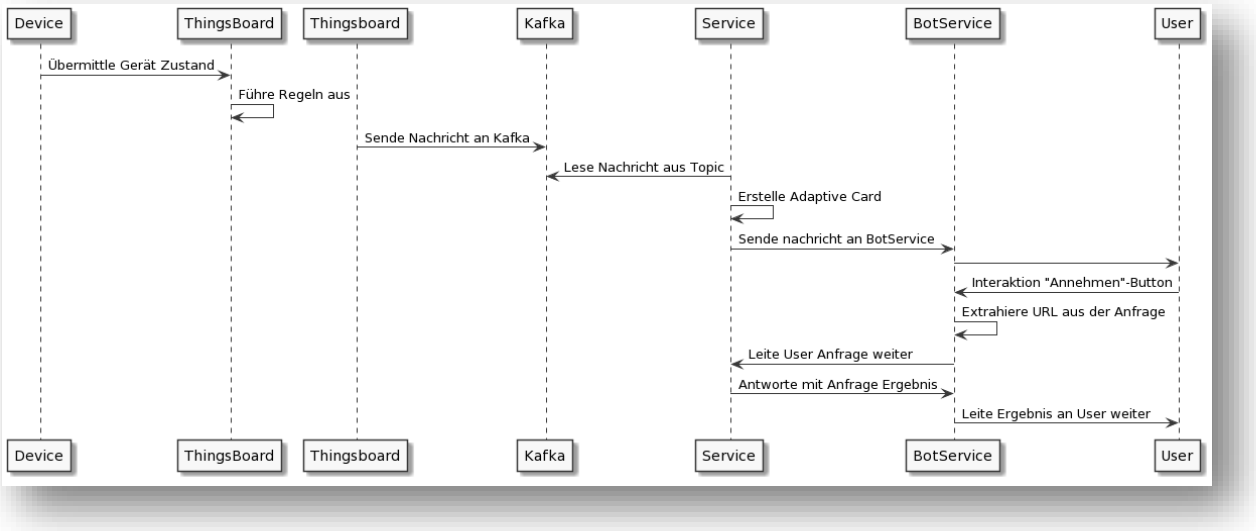


Abbildung 5 Sequenzdiagramm Applikation

Mit den beschriebenen Konzepten und der Referenzarchitektur lassen sich neue Services einbinden, ohne dass in umliegenden Komponenten Änderungen vorgenommen werden müssen. Ausgeschlossen davon sind die Stammdatenpflege wie Devices und Erweiterungen an den Regelwerken in ThingsBoard.

5.2 Schnittstellen

Dieses Kapitel enthält die Spezifikation der Schnittstellen. Die Schnittstellen ermöglichen die Umsetzung der im vorherigen Abschnitt beschriebenen Prinzipien.

5.2.1 Bot Service

Der Bot Service bietet die Microsoft Bot Framework API und eine zusätzliche Schnittstelle für das Erhalten von benachrichtigungsbasierten Anfragen an.

Die Definition der API für Benachrichtigungen ist wie folgt:

Benachrichtigungs API

Request	URL	/api/notify
	Method	POST

	Body	Adaptive Card JSON
		
		Ist ein Interaktionselement wie beispielsweise ein Button enthalten, muss das in dem Adaptive Card Schema definierte „data“ – Attribut für das Interaktionselement folgendes enthalten:
		
		"data": {
		"target" : "",
		"payload" : {}
		}
		
		target beschreibt die URL, an welche der Bot Service bei einer Interaktion des Benutzers die Anfrage mittels POST weiterleitet.
		
		payload ist der Inhalt, der in dem POST Body in die Anfrage hinzugefügt wird. Jegliches valides JSON Format kann als Payload inkludiert werden.

Response	Status Codes	200 - Bei erfolgreicher Anfrage
	Body	-

Tabelle 1 Benachrichtigungs API

Die Microsoft Bot Framework API ist von der SDK implementiert. Sie dient primär dazu, um eine Kommunikation zu und von Microsoft Teams zu ermöglichen. Im Detail wird auf die API nicht weiter eingegangen. Die Dokumentation ist im Literaturverzeichnis verlinkt (vgl. [THI20]).

5.2.2 Services

Services kommunizieren mit dem Bot Service ausschließlich über Adaptive Cards. Die URLs zu den API-Endpunkten und die Daten können vom Service selbst bestimmt werden. Durch die Benachrichtigungs-API des Bot Services wird das in Kapitel 5 Konzeption beschriebene HATEOAS-Prinzip realisiert. Um das Prinzip einzuhalten muss die Service Schnittstelle folgende Kriterien erfüllen.

- Schnittstellen, die an den Bot Service über die Benachrichtigungs-API zur Verfügung gestellt werden, müssen POST Anfragen akzeptieren
- Die Response muss Eine Adaptive Card enthalten. Den Inhalt kann der Service selbst bestimmen. Sind Interaktionselemente wie beispielsweise ein Button vorhanden, muss das in Tabelle 1 Benachritigungs API beschriebene Format eingehalten werden.

Abgesehen von diesen Kriterien, sind die Services frei bei der Benennung der Schnittstellen URLs.

5.2.2.1 Registry

Die Registry stellt ebenfalls Adaptive Cards zur Verfügung. Sie besitzt eine Schnittstelle und ist wie folgt definiert.

Registry API		
Request	URL	https://<HOST>/bot/action
	Method	POST
	Body	-
Response	Status Codes	200 - Bei erfolgreicher Anfrage
	Body	Adaptive Card JSON

Tabelle 2 Registry API

Die Adaptive Card, die die Registry in der Response zur Verfügung stellt, wird dem Benutzer direkt angezeigt. Die visuelle Darstellung ist den Anforderungen auf UX Perspektive überlassen. Inhaltlich sollte eine Navigationsmöglichkeit über beispielsweise Buttons vorgesehen sein. Beispielsweise kann die Registry eine Adaptive Card mit einer Liste von Buttons zur Verfügung stellen, die vom Benutzer angeklickt werden kann. Dadurch wird eine Navigation zu den verschiedenen in der Registry registrierten Services ermöglicht. Um die Navigation zu ermöglichen muss die Bot Service API-Regelung eingehalten werden.

Beispielsweise kann die Registry Response ein Adaptive Card Button mit folgendem *data* Attribut enthalten:

```
"data": {
   "payload": {},
   "target": "https://<host>/list_all_devices"
}
```

Code 1 Registry Response Code

Klickt der Benutzer auf den Button, wird vom Bot Service eine POST Anfrage auf die URL in *target* mit der *payload* abgeschickt.

5.2.4 ThingsBoard

ThingsBoard bietet diverse Schnittstellen an. Die für das Entgegennehmen von Devicedaten im Folgenden beschrieben. Andere ThingsBoard Schnittstellen können

der ThingsBoard Dokumentation entnommen werden und haben zum aktuellen Stand keine Relevanz.

ThingsBoard Device Attributes API		
Request	URL	https://<host>/api/v1/<DEVICE_TOKEN>/attributes DEVICE_TOKEN: Der Token des in ThingsBoard angelegten Devices.
	Method	POST
	Body	Valides JSON Format Der Request Body kann jegliches JSON Objekt entgegennehmen. Die Verarbeitung des Formats kann gerätespezifisch in der Rule Engine durchgeführt werden. ThingsBoard gibt für diesen Endpunkt kein Format vor.
Response	Status Codes	200 - Bei erfolgreicher Anfrage
	Body	-

Tabelle 3 ThingsBoard Device Attrbutes API

5.2.5 Kafka

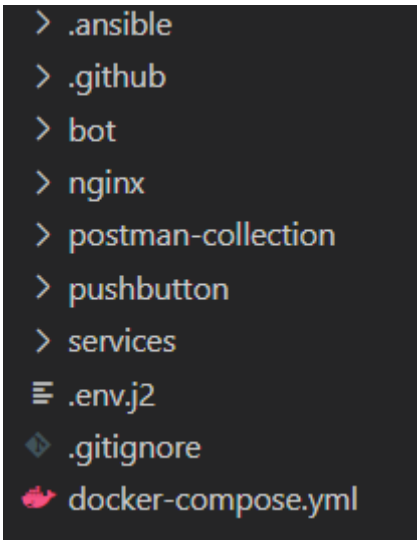
Jeder Service kann ein oder mehrere Kafka Topics besitzen. Die Anzahl der Topics sowie der Inhalt der Daten, die an das Topic gesendet und gelesen werden, sind Service spezifisch. Daher ist an dieser Stelle keine weitere Schnittstellenbeschreibung notwendig. Die Beschreibung der Service spezifischen Schnittstellen für Kafka werden in Kapitel 6 Umsetzung im Detail dargestellt.

6. Umsetzung

Dieses Kapitel umfasst die Details der umgesetzten Komponenten. Die in Kapitel Konzeption beschriebenen Konzepte und Schnittstellen wurden in die Umsetzung einbezogen.

Der gesamte Code des Projektes ist auf GitHub hinterlegt. (https://github.com/HHZ-UCC/ucc)

Nachfolgend ist die Projektstruktur beschrieben.



.ansible: Enthält alle Ansible Automatisierungsskripte für die Installation des Projektes auf einem Server. In Kapitel 7 Installation und Konfiguration ist Ansible im Detail beschrieben.

.github/workflow: Enthält die GitHub Action Definition. In Kapitel 7 Installation und Konfiguration ist GitHub Action im Detail beschrieben.

bot: Enthält den Code für den Bot Service. In Kapitel 6.1.2 Bot Service ist der Bot Service im Detail beschrieben.

nginx: Enthält die Konfiguration für das NGINX gateway. Sie ist in Kapitel 7.3 Nginx & Certbot im Detail beschrieben.

postman-collection: Enthält eine Postman Zusammenstellung, um die ThingsBoard API für Devices zu simulieren.

pushbutton: Enthält den Code für das Pushbutton das in Kapitel 6.5 Pushbutton beschrieben wird.

services: Enthält den Code für die Django basierten Services. Diese sind im Detail in Kapitel 6.2 Services beschrieben.

docker-compose.yml & .env.j2: Enthält die Docker Definition für das Installieren und Starten der Komponenten. In Kapitel 7 Installation und Konfiguration ist diese im Detail beschrieben.

6.1 Microsoft Teams Bot

In diesem Kapitel wird auf die Implementierungsdetails des Bot Services eingegangen, das mit Microsoft Produkten umgesetzt wurde.

6.1.1 Microsoft Teams

Microsoft Teams (abgekürzt MS Teams oder nur Teams) ist eine UCC-Plattform, welche von Microsoft entwickelt wurde (vgl. [MST20]). Sie bündelt verschiedene Features wie beispielsweise Chat, Besprechungen, Notizen und Anhänge (vgl.

[MST20]). Die Anwendung wird als Kollaborations-, Kommunikations- und Videokonferenz-Tool im Unternehmen eingesetzt. In Microsoft Teams können Teams angelegt werden, zu denen Personen innerhalb oder außerhalb des Unternehmens eingeladen werden können. Microsoft Teams ermöglicht es, verschiedene Bots in der täglichen Zusammenarbeit zu nutzen. Die Plattform ist von verschiedenen Geräten wie PCs, Laptops, Smartphones oder Tablets nutzbar.

6.1.2 Adaptive Cards

Adaptive Cards sind ein wesentlicher Bestandteil der in diesem Projekt entwickelten Lösung. Durch Adaptive Cards können Interaktionskonzepte des Benutzers mit einem Chatbot umgesetzt werden. Benutzer müssen dabei keine Textnachrichten eingeben und können beispielsweise über Buttons mit dem Chatbot interagieren.

Adaptive Cards werden im JSON Format erstellt und Microsoft Teams über dem Bot Service zur Verfügung gestellt. Das Erstellen des User Interfaces anhand des JSON übernimmt Microsoft Teams. Unter https://adaptivecards.io/designer/ wird ein Werkzeug angeboten, um Adaptive Cards zu entwickeln. Auch sind diverse Beispiele vorhanden, die als Vorlage verwendet werden können.

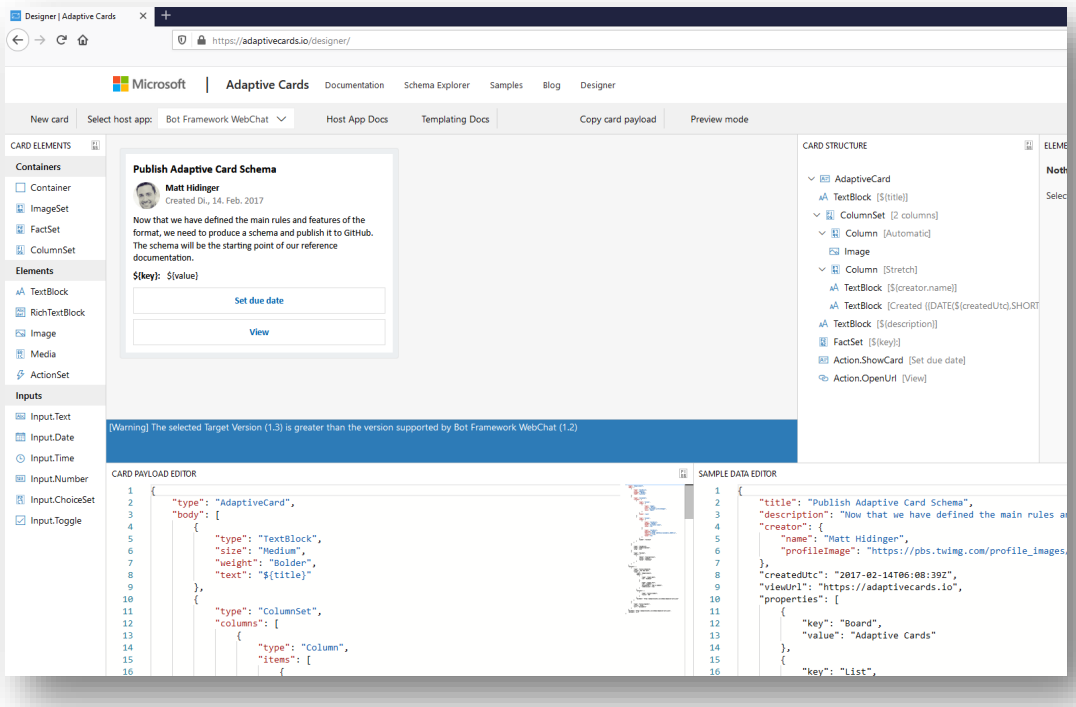


Abbildung 6 Adaptive Cards Designer Webseite

6.1.3 Bot Service

Um einen Chatbot in Microsoft Teams zu integrieren, müssen APIs gemäß der Microsoft Teams Spezifikation implementiert werden. Microsoft stellt das Bot Framework zur Verfügung, um die Realisierung von Chatbots direkt mittels Programmierschnittstellen zu ermöglichen. Dadurch wird die Umsetzung erleichtert, da direkt mit Programmierbibliotheken gearbeitet werden kann.

Der Bot Service wurde mit der JavaScript SDK des Bot Frameworks implementiert. Der Aufbau des Bot Services ist wie folgt:

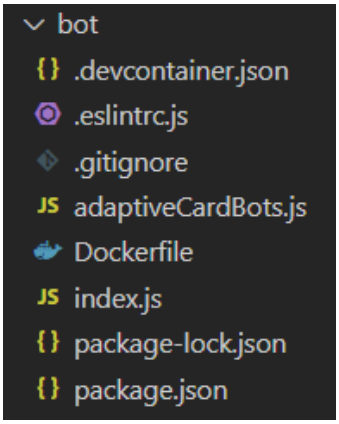


Abbildung 7 Bot Service Struktur

Die *index.js* und die *adaptiveCardBots.js* enthalten die Umsetzung. In der *index.js* werden hauptsächlich die API-Endpunkte gemäß der in 5.2.1 Bot Service beschriebenen Schnittstellen Spezifikation umgesetzt. Die *adaptiveCardBots.js* enthält die Implementierung, die das in der Konzeption beschriebene HATEOAS-Prinzip ermöglicht. Auf den Code wird daher nicht weiter eingegangen.

6.1.4 Microsoft Azure Bot Channel Registration

Azure ist eine Cloud Lösung von Microsoft. Sie erlaubt es, eine Vielzahl an Cloud Services wie beispielsweise Virtuelle Maschinen oder Messaging Queues zu provisionieren und einzusetzen.

Um einen Chatbot in Teams einzubinden, ist es erforderlich, auf Microsoft Azure eine „Bot Channel Registration“ zu erstellen. Im Folgenden werden die einzelnen Schritte, die dafür notwendig sind, beschrieben. Es wird vorausgesetzt, dass bereits ein Azure Account vorhanden ist. Dieser kann über https://portal.azure.com erstellt werden.

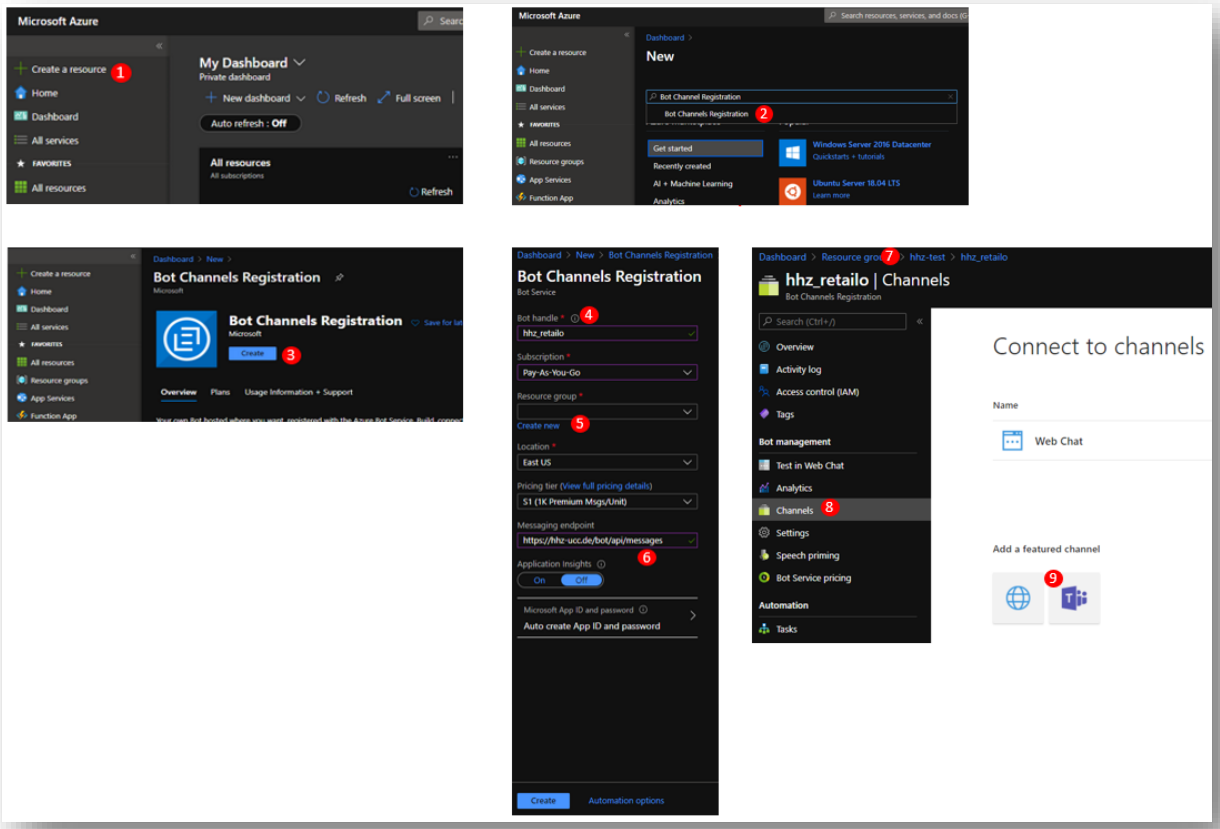


Abbildung 8 Microsoft Konfiguration I

1. Anlage eines neuen Azure Dienstes.

2. Suche nach „Bot Channel Registration“.

3. Erstellen der „Bot Channel Registration“.

4. Vergeben des Bot Namens. Das Bot ist unter diesem Namen später in Teams verfügbar.

5. Über „Create new“ eine neue Ressourcengruppe erstellen. Es erscheint ein Dialog Fenster, in dem der Name der Ressourcengruppe eingegeben werden kann.

6. Angeben der URL des Bot Services https://hhz-ucc.de/bot/api/messages. Das kann nach der Anlage in den Einstellungen ebenfalls geändert werden. Mit Create die Anlage bestätigen.

7. Navigieren zum angelegten Bot Channel Dienst.

8. Öffnen der „Channels“ Einstellungen.

9. Microsoft Teams hinzufügen.

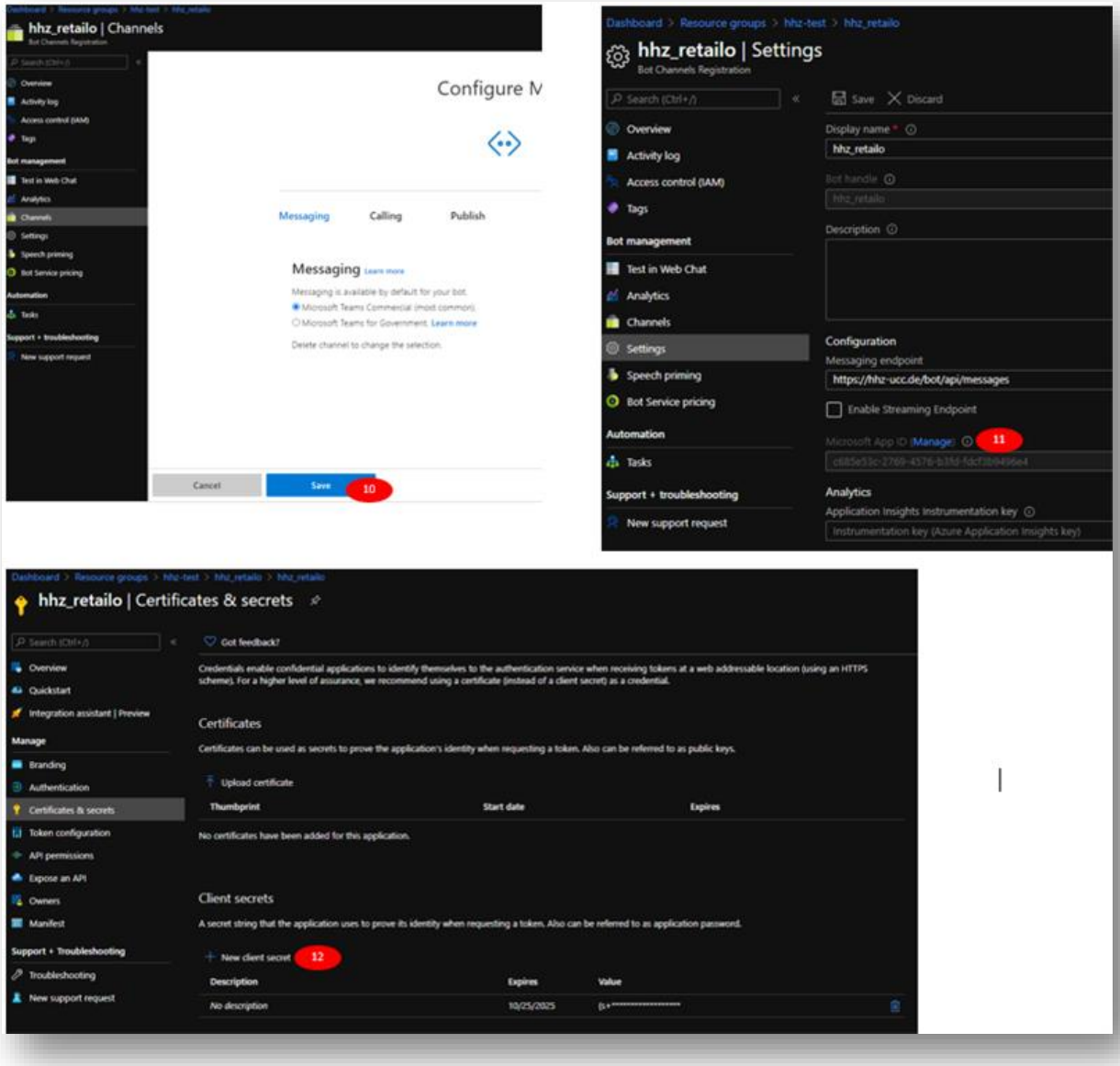


Abbildung 9 Microsoft Azure Konfiguration II

10. Mit „Save“ die Anlage bestätigen.

11. Navigieren zu „Settings“. In diesem Dialog ist die Microsoft App ID dargestellt. Diese wird in Kombination mit dem Password benötigt, um die Authentifizierung zwischen Microsoft Teams und dem Bot Service zu ermöglichen. Zum Erstellen des Passwortes dem Link „Manage“ folgen.

12. Über „New client secret” ein neues Passwort generieren und aufschreiben. Es ist später nicht mehr möglich auf das Passwort zuzugreifen.

Nachdem der Bot Channel angelegt ist, können die Microsoft App ID und das Passwort als Umgebungsvariable im Bot Service gesetzt werden. Die Konfiguration dafür kann aus Kapitel 7.4 Übersicht der Umgebungsvariablen entnommen werden.

6.2 Services

Services implementieren die in Kapitel 4 Anwendungsfall beschriebenen Use Cases. Es sind drei Services implementiert, auf dessen Implementierungsdetails in diesem Kapitel näher eingegangen wird.

6.2.1 Django

Das beschriebene Konzept kann mithilfe von Microservices oder monolithisch umgesetzt werden. In der Microservice basierten Architektur ist jeder Service voneinander isoliert und kann von unterschiedlichen Teams sowie mit unterschiedlichen Programmiersprachen oder Frameworks genutzt werden. Allerdings ist die Komplexität für den Betrieb und die Entwicklung höher. Unter diesen Aspekten wurde für die Entwicklung der Services ein monolithischer Ansatz gewählt. Der Betrieb und die Entwicklungskomplexität sind für die Entscheidung ausschlaggebende Aspekte.

Ein modular, möglichst unabhängiger Aufbau der Services ist dennoch gewünscht. Darum wurden Web Frameworks analysiert und speziell Django als Framework ausgewählt. Django erlaubt hinsichtlich Betriebes und Entwicklungskomplexität einen monolithischen Ansatz, unterstützt aber dennoch einen modularen Aufbau.

Django ist ein Python Web Framework. Es ist in der Python Community weit verbreitet und speziell auf Modularität und Einfachheit ausgelegt. Django besitzt ein Projekt und ein oder mehrere Applikationen (Apps). Das Projekt beschreibt applikationsübergreifende Einstellungen. Beispielsweise wird im Projekt definiert welche Apps im Betrieb verwendet werden sollen. Apps können deshalb über die Einstellungen hinzugefügt oder entfernt werden. Eine App besteht alle notwendigen Konstrukten wie URLs unter denen Schnittstellen erreichbar sind, Datenbank Modelle und HTML Templates, welche für eine Web Applikation notwendig sind.

Django stellt zusätzlich Benutzeraccounts und Logins, sowie ein umfangreiches Berechtigungskonzept zur Verfügung. Im Django Admin User Interface können CRUD Operationen durchgeführt werden.

Die Nachfolgende Abbildung stellt das Django Admin User Interface dar:

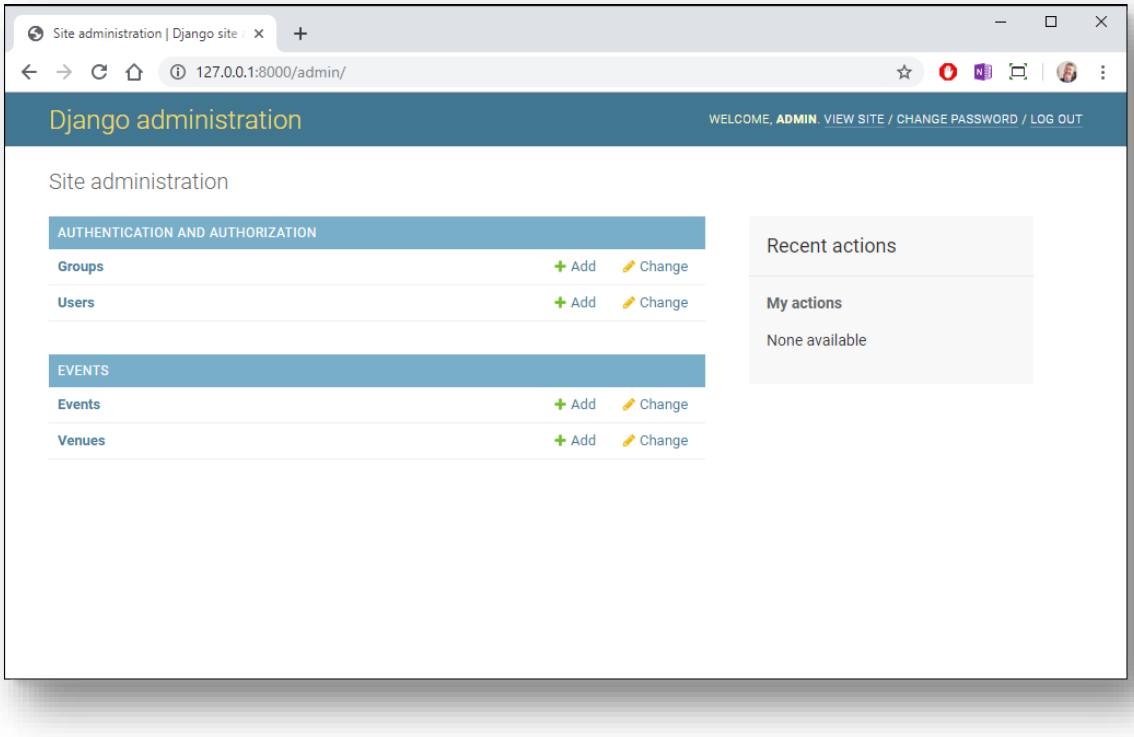


Abbildung 10 Django Administration UI [DA20]

Eine Django App ist wie folgt aufgebaut:

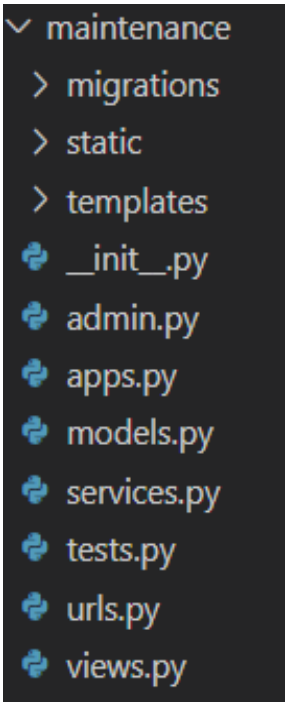


Abbildung 11 Django Struktur

Das Root Verzeichnis ist nach der App benannt und enthält alle für die App relevanten Dateien

migrations: Enthält alle Datenbankmigrationsskripte

static: Enthält alle statischen Dateien wie beispielsweise Bilder

templates: Enthält alle HTML oder JSON Templates. Diese Dateien können in Views referenziert und mit zusätzlichen Daten angereichert werden

admin.py: Erlaubt die Konfiguration der App für das Django Admin User Interface. Beispielsweise können hier die Datenbankobjekte angegeben werden, die im Admin User Interface zur Verfügung stehen sollen

models.py: Enthält die Datenbankmodelle.

services.py: Klassen und Businesslogik können in dieser Datei implementiert und an beliebiger Stelle der App eingesetzt werden

tests.py: Unit Tests können in dieser Datei implementiert werden

urls.py: Enthält die Konfiguration der http Endpunkte, welche die App zur Verfügung stellt

views.py: Enthält die Klassen, die auf URLs registriert werden können. Diese werden aufgerufen sobald eine URL angefragt wird.

Nach diesem Konzept sind die Apps für das UCC Projekt umgesetzt. Jeder Service repräsentiert im Projekt eine App. Die Struktur eines Services sieht wie folgt aus:

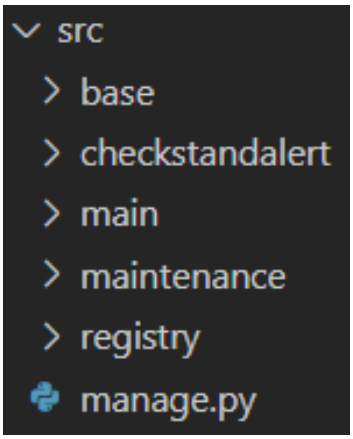


Abbildung 12 Django Projekt Struktur

src: Root Verzeichnis des Projektes

base: Die base App enthält wiederverwendbaren Code. Beispielsweise Datenbankobjekte wie Device und Employee, die in mehreren Apps vorhanden sind. Dadurch wird eine Coderedundanz vermieden. Die base App stellt keine *views* oder *urls* zur Verfügung.

checkstandalert: Repräsentiert die App für den Anwendungsfall „Kassenalert“.

main: Enthält die Projektkonfiguration. Hier werden projektübergreifende Einstellungen gesetzt. Beispielsweise kann eingestellt werden, welche Apps im Deployment zur Verfügung gestellt werden sollen.

maintenance: Repräsentiert die App für den Anwendungsfall „Tickets“

Registry: Repräsentiert die App für die Registry

manage.py: Ist ein von Django zur Verfügung gestelltes Skript, welches das Ausführen von Django Befehlen wie beispielsweise das Generieren von Datenbankmigrationen ermöglicht.

In den nachfolgenden Abschnitten werden die Implementierungsdetails der einzelnen Services beschrieben.

6.2.2 Registry Service

Die Registry verfügt über kein Datenbankmodell. Die Einträge der einzelnen Services werden dynamisch zur Laufzeit von der Registry abgefragt. Hierzu wird ein Django Feature verwendet. Django erlaubt es, auf alle Apps über eine Django interne API zuzugreifen. Die Registry liest anhand dieser Möglichkeit alle verfügbaren Apps aus und überprüft, ob diese für sie relevant sind.

Die Relevanz wird über eine Methode in den jeweiligen Apps bestimmt. Ist die Methode vorhanden wird die App in der Registry registriert. Der nachfolgende Ausschnitt stellt die Implementierung in der Maintenance App dar.

```python
# maintenance/apps.py
def get_registry_info(self):
    host = settings.HOST
    return  {
        "app"         : self.verbose_name,
        "type"        : "list_tickets",
        "displayText" : "Zeige alle Tickets",
        "targetUrl"   : '{host}/services/{name}/bot/list_tickets'.format(host=host, name=self.name)
    }
```

Code 2 Registry Info Implementierung

Anhand von diesen Attributen generiert die Registry über die in Kapitel 5 Konzeption beschriebenen Schnittstellen eine Adaptive Card, an welcher sich der Bot Service bedient. In Teams wird die Adaptive Card der Registry wie folgt dargestellt:

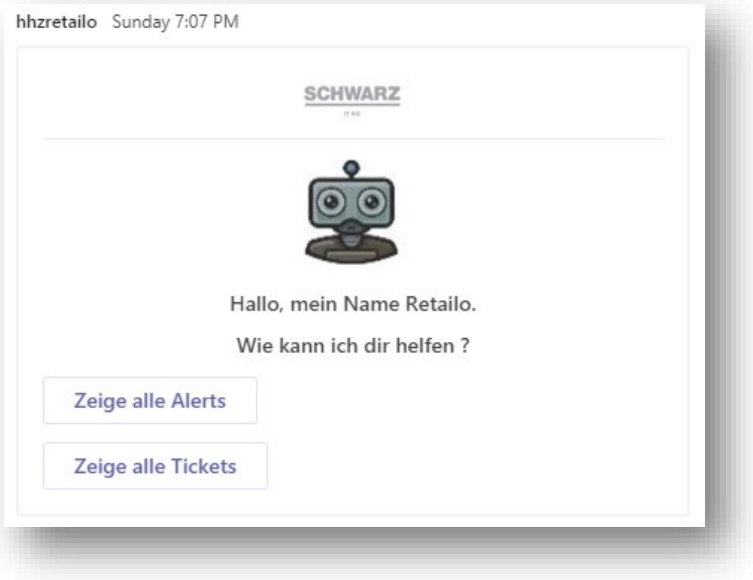


Abbildung 13 Registry Adaptive Card

Die Registry erstellt eine Adaptive Card mit einem Button pro Registry-Eintrag. Die Texte für die Buttons sowie die aufzurufenden URL‘s werden von den Services bestimmt.

Durch dieses Konstrukt können beliebige Apps integriert werden sofern die notwendige Methode implementiert wird.

Die Registry unterstützt aktuell keine Registrierung von Services, die nicht im Django Projekt zur Verfügung stehen. Ist dies in Zukunft eine Anforderung, kann dies über die Verfügungstellung einer API ermöglicht werden.

6.2.3 Maintenance Service

Der Maintenance Service verwaltet erforderliche Wartungen an Devices. Hierzu implementiert der Maintenance Service folgendes Datenbank Model.

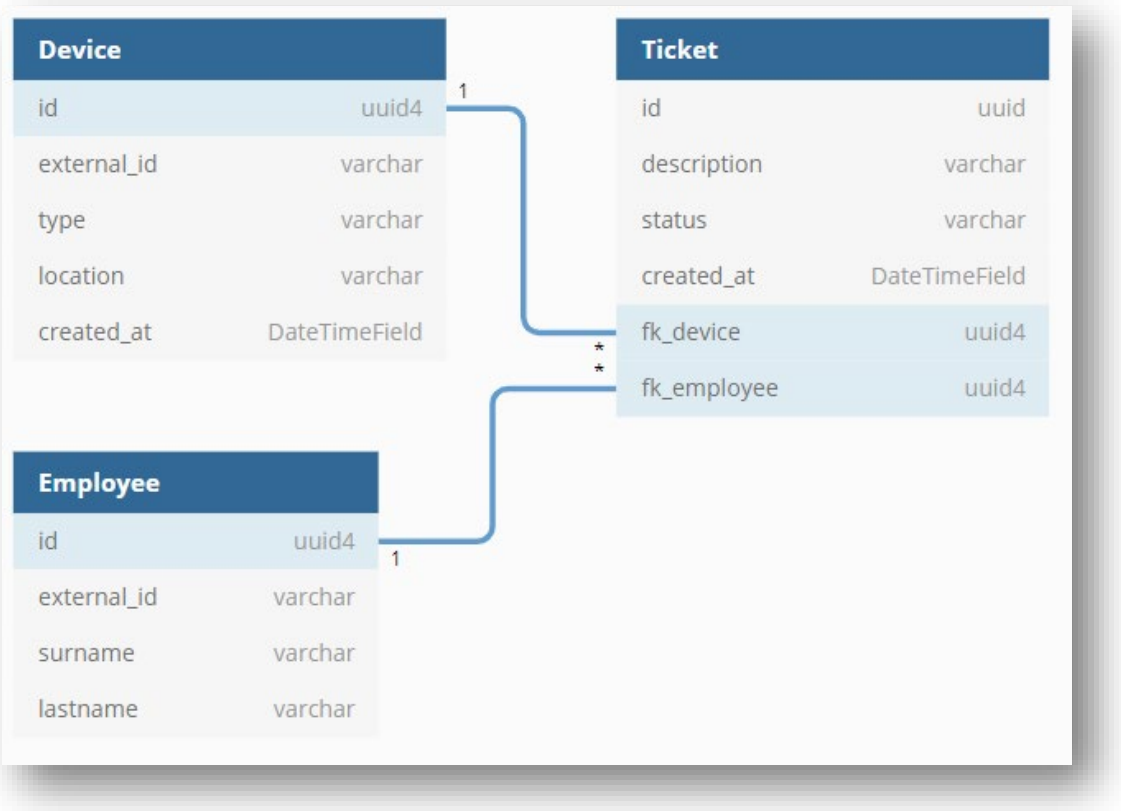


Abbildung 14 Datenmodell Maintenance Service

Zu einem Device können mehrere Tickets im System hinterlegt werden. Jedes Ticket kann einem Employee zugewiesen werden. Ein Employee kann mehrere Tickets zugewiesen bekommen. Dieses Datenbankmodell ermöglicht die Umsetzung des Anwendungsfalls.

Der Service stellt folgende APIs zur Verfügung.

API: list_tickets		
Beschreibung		Liefert eine Adaptive Card mit einer Liste von Tickets zurück.
Request	URL	https://<host>/bot/list_tickets
	Method	POST
	Body	-
Response	Status Codes	200 - Bei erfolgreicher Anfrage
	Body	Adaptive Card JSON
API: assign_tickets		
Beschreibung		Weist einem Benutzer ein Ticket zu und bestätigt die erfolgreiche Zuweisung mit einer Adaptive Card.
Request	URL	https://<host>/bot/assign_tickets
	Method	POST

<table>
<tr><td></td><td>Body</td><td>
<table>
<tr><th>Attribut</th><th>Beschreibung</th></tr>
<tr><td>payload.ticketId</td><td>ID des Tickets an welches der Benutzer verknüpft wird</td></tr>
<tr><td>user.id</td><td>ID des Benutzers, das dem Ticket hinzugefügt wird. Gibt es den Benutzer nicht in der Datenbank, wird ein Eintrag angelegt.</td></tr>
<tr><td>user.name</td><td>Name des Benutzers</td></tr>
</table>

Beispiel:
<pre>{
  "payload": {
     "deviceId" : "",
     "ticketId" : ""
  }
  "user": {
     "id" : "",
     "name" : ""
  }
}</pre>
</td></tr>
<tr><td rowspan="2">Response</td><td>Status Codes</td><td>200 - Bei erfolgreicher Anfrage</td></tr>
<tr><td>Body</td><td>Adaptive Card mit einer Bestätigungsnachricht</td></tr>
</table>

Tabelle 4 Maintenance Service APIs

Der Maintenance Service konsumiert ein Kafka *topic,* um die Device Daten, welche von ThingsBoard an Kafka übertragen werden, zu erhalten. Die nachfolgende Tabelle stellt die Details dar.

<table>
<tr><th colspan="2">Kafka Consumer: Tickets</th></tr>
<tr><td>Beschreibung</td><td>Eine Nachricht in diesem Topic wird als Ticket im Maintenance Service angelegt. Der Maintenance Service hört auf Nachrichten dieses Topics und erstellt sobald eine neue Nachricht eintrifft ein Ticket in der Datenbank. Zusätzlich wird eine Adaptive Card generiert und an den Bot Service weitergeleitet.</td></tr>
<tr><td>Topic</td><td>tickets</td></tr>
<tr><td>Payload</td><td>
<table>
<tr><th>Attribut</th><th>Beschreibung</th></tr>
<tr><td>device.id</td><td>Die ID des Devices aus ThingsBoard</td></tr>
<tr><td>device.deviceType</td><td>Der Type des Devices. Beispielsweise Waage.</td></tr>
<tr><td>device.shared_location</td><td>Die Lokation des Devices. Beispielsweise Obstabteilung.</td></tr>
</table>
</td></tr>
</table>

	Beispiel:
	{
	"device": {
	"id" : "",
	"deviceType" : "",
	"shared_location" : ""
	}
	}

Tabelle 5 Kafka Consumer API

Der Maintenance Service sendet die Nachfolgende Adaptive Card an den Bot Service sobald ein Ticket aus dem Kafka *topic* konsumiert wurde.

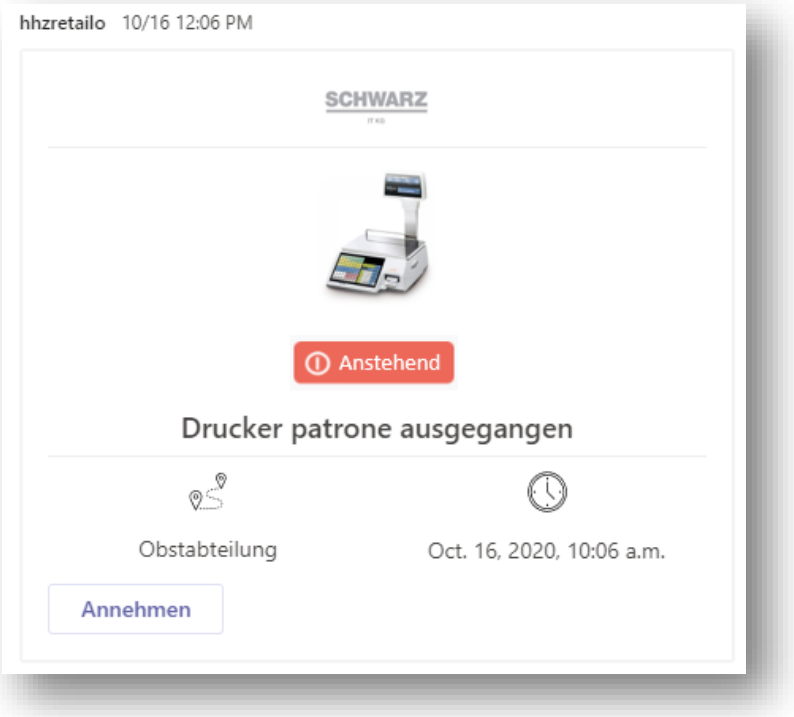


Abbildung 15 Adaptive Card Maintenance Service

Der „Annehmen“ - Button referenziert dabei die *assign_tickets* API. Alle Daten, die die API benötigt, sind in der Adaptive Card enthalten und ermöglichen dadurch dem Bot Service einen korrekten Aufruf der *assign_tickets* API. Klickt der Benutzer auf den Annehmen Button wird er dem Ticket hinzugefügt und mit einer Adaptive Card bestätigt. Diese ist folgendermaßen aufgebaut.

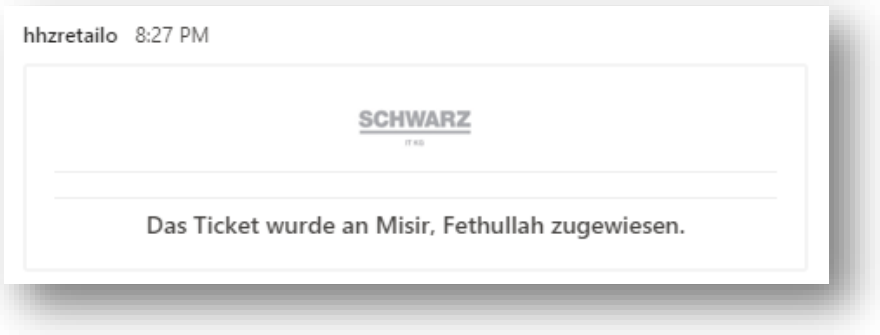


Abbildung 16 Adaptive Card Bestätigung

6.2.4 Checkstandalert Service

Der Checkstandalert Service verwaltet Kassenrufe in einer Filiale. Hierzu implementiert der Checkstandalert Service folgendes Datenbank Model.

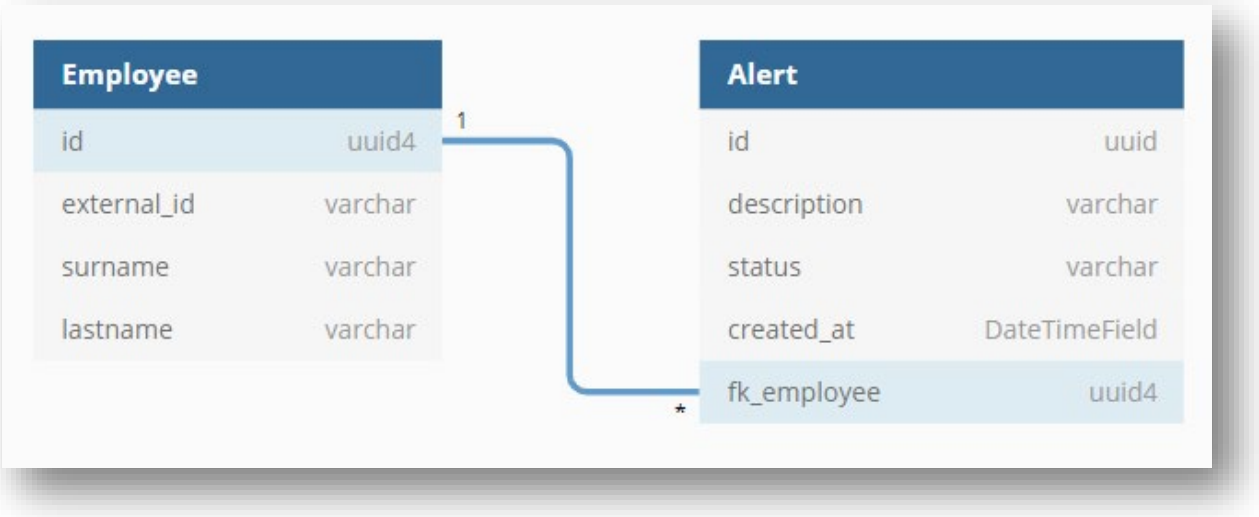


Abbildung 17 Datenmodell Checkstandalert Service

Ein Employee kann mehrere Alerts zugewiesen bekommen. Ein Alert Repräsentiert einen Ruf zur Kasse. Dieses Datenbankmodel ermöglicht die Umsetzung des Anwendungsfalls.

Der Service stellt folgende APIs zur Verfügung.

API: list_alerts		
Beschreibung		Liefert eine Adaptive Card mit einer Liste von Alerts zurück.
Request	URL	https://<host>/bot/list_alerts
	Method	POST
	Body	-

<table>
<tr><td rowspan="2">Response</td><td>Status Codes</td><td>200 - Bei erfolgreicher Anfrage</td></tr>
<tr><td>Body</td><td>Adaptive Card JSON</td></tr>
<tr><td colspan="3">API: assign_alerts</td></tr>
<tr><td colspan="2">Beschreibung</td><td>Weist einem Benutzer ein Alert zu und bestätigt die erfolgreiche Zuweisung mit einer Adaptive Card.</td></tr>
<tr><td rowspan="3">Request</td><td>URL</td><td>https://<host>/bot/assign_alerts</td></tr>
<tr><td>Method</td><td>POST</td></tr>
<tr><td>Body</td><td>
<table>
<tr><th>Attribut</th><th>Beschreibung</th></tr>
<tr><td>payload.alertId</td><td>ID des Alerts zu dem der Benutzer verknüpft wird</td></tr>
<tr><td>user.id</td><td>ID des Benutzers das dem Ticket hinzugefügt wird. Gibt es den Benutzer nicht in der Datenbank wird ein Eintrag angelegt.</td></tr>
<tr><td>user.name</td><td>Name des Benutzers</td></tr>
</table>
Beispiel:
{
"payload": {
"alertId" : ""
}
"user": {
"id" : "",
"name" : ""
}
}
</td></tr>
<tr><td rowspan="2">Response</td><td>Status Codes</td><td>200 - Bei erfolgreicher Anfrage</td></tr>
<tr><td>Body</td><td>Adaptive Card mit einer Bestätigungsnachricht</td></tr>
</table>

Tabelle 6 Maintenance Service APIs

Der Checkstandalert Service konsumiert ein Kafka *topic* um die Kassenrufe, welche von ThingsBoard an Kafka übertragen werden, zu erhalten. Die nachfolgende Tabelle beschreibt die Details.

Kafka Consumer: Checkstandalert	
Beschreibung	Eine Nachricht in diesem Topic wird als Alert im Checkstandalert Service angelegt. Der Checkstandalert Service hört auf Nachrichten dieses Topics und erstellt sobald eine neue Nachricht eintrifft ein Alert in der Datenbank. Zusätzlich wird eine Adaptive Card generiert und an den Bot Service weitergeleitet.
Topic	checkstandalert
Payload	Erfordert keine Payload. Da keine Informationen aus ThingsBoard für den Anwendungsfall notwendig sind.

Tabelle 7 Kafka Consumer Checkstandalert

Der Checkstandalert Service sendet die nachfolgende Adaptive Card an den Bot Service sobald ein Alert aus dem Kafka *topic* konsumiert wurde.

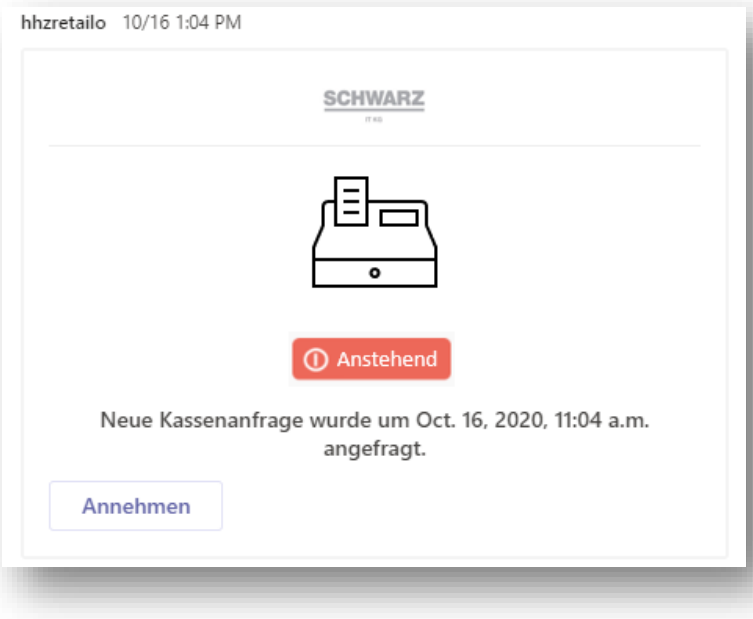


Abbildung 18 Adaptive Card Checkstandalert

Der „Annehmen“ - Button referenziert dabei die *assign_alerts* API. Alle Daten, die die API benötigt, sind in der Adaptive Card enthalten und ermöglichen dadurch dem Bot Service einen korrekten Aufruf der *assign_alerts* API. Klickt der Benutzer auf den Annehmen Button wird er dem Alert zugewiesen, und erhält eine Bestätigung, die Adaptive Card dargestellt wird. Diese ist wie folgt aufgebaut.

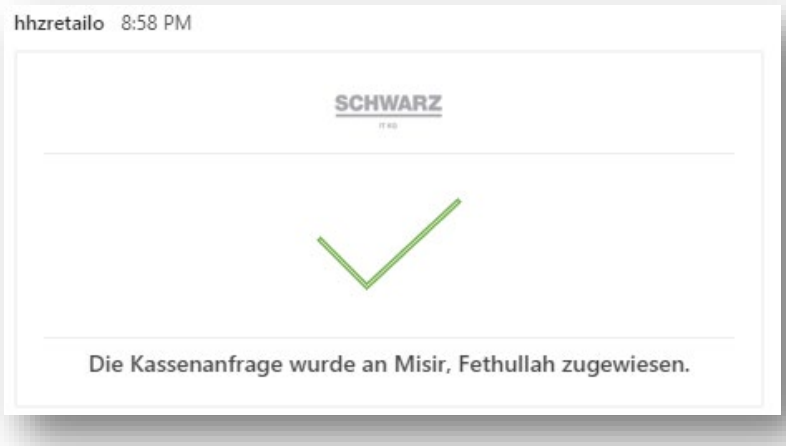


Abbildung 19 Bestätigung Checkstandalert

6.3 Kafka & Zookeeper

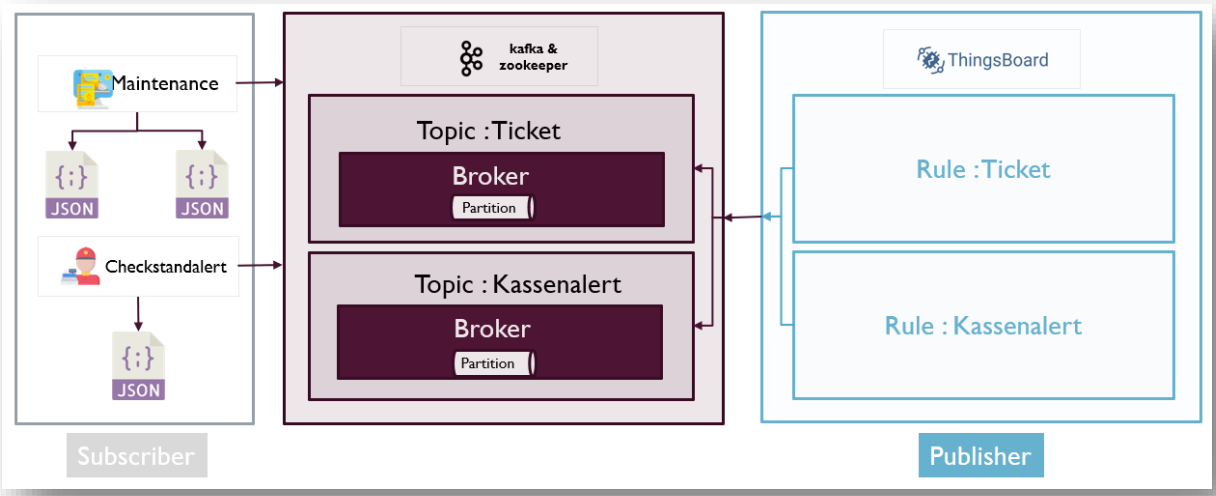


Abbildung 20 Apache Kafka mit Zookeeper

Apache Kafka ist ein verteiltes Messaging-System, dessen Kernfunktion auf Skalierbarkeit, Performance und Fehlertoleranz ausgerichtet ist (vgl. [KAF17]). Es wurde von LinkedIn entwickelt und ist seit 2011 Open Source. Es ermöglicht eine entkoppelte, asynchrone Kommunikation zwischen mehreren Komponenten einer Anwendung. Dabei basiert Kafka auf einem verteilten Public-Subscribe System (vgl. [KAF17]).

Kafka organisiert die einzelnen Nachrichten über Topics. Jede Nachricht erhält einen Eintrag mit einem Schlüssel, Wert und Zeitstempel. Diese Einträge werden persistiert und in eine sogenannte Partition abgelegt. Eine Partition ist eine Menge von Records, die über eine Sequenznummer (Offset) nummeriert ist. Die Vergabe von Offsets und die Konfiguration von Brokern wird durch den Zookeeper sichergestellt.

Abbildung 20 Apache Kafka mit Zookeeper zeigt den Aufbau für das UCC Projekt. Dabei wird die Kafka Instanz mit zwei Topics, die jeweils zu einem Broker gehören konfiguriert. Der Publisher, in diesem Fall das Thingsboard versorgt die Kafka-Anwendung mit den Nachrichten aus der Waage und aus dem physischen Knopf. Jedes dieser Geräte ist mit einem Topic verknüpft und sendet innerhalb dieses Topics. Die Consumer wiederum sind die registrierten Services, Maintenance und Checkstandalert Service innerhalb der Django Applikation. Dabei kann der Service sich für das jeweilige Topic registrieren und Nachrichten erhalten.

6.4 Thingsboard

ThingsBoard ist eine IoT-Lösung für die Datenerfassung, Datenverarbeitung, Datenvisualisierung und Geräteverwaltung.

Wie im Kapitel 4 Anwendungsfall beschrieben, wird der Fokus auf zwei Geräte festgelegt. Dabei handelt es sich um eine internetfähige Obst- und Gemüsewaage , sowie einen physischen Knopf für einen weiteren Kassenaufruf.

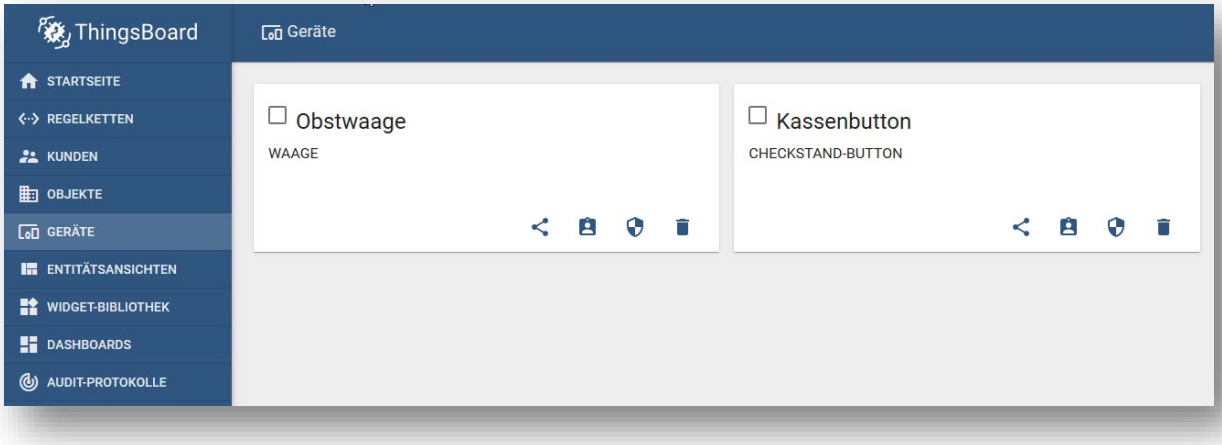


Abbildung 21 Thingsboard Geräteverwaltung

Die beiden Geräte werden im Thingsboard mit ihren jeweiligen Stammdaten als virtuelle Geräte registriert. Für eine Waage wird als Attribute die Bezeichnung, ID, Standort und Beschreibung festgelegt. Bis auf das Attribut Standard sind alle aufgezählten Attribute im Thingsboard vorhanden. Das Thingsboard unterstützt das Hinzufügen von weiteren gerätespezifischen Attributen. Folglich wurde das Standort Attribut in die Waage als gemeinsame Attribute[1] aufgenommen. Der physische Knopf hingegen wird mit denselben Attributen bis auf den Standort angelegt, hierbei spielt das Standort Attribut keine Rolle. Jedes Gerät besitzt ein eindeutiges Token für den Zugriff von außen. Dieser Token kann durch eine bis zu 20 Stellen selbstdefinierte Zeichenkette ersetzt werden.

```
http(s)://host:port/api/v1/$ACCESS_TOKEN/attributes
```

Abbildung 22 Geräte - HTTP Endpunkt [TI20]

[1] Siehe: https://thingsboard.io/docs/user-guide/ui/devices/

Die angelegten Geräte können mit Hilfe ihrer Token Daten empfangen. Dieser Endpunkt ist das Überführen von physischen Gräten in die virtuelle Umgebung. Für die Integration bietet Thingsboard eine Standard Schnittstelle an. Hier wird anhand des Tokens (siehe Abbildung 2 Geräte - HTTP Endpunkt) das Gerät ermittelt und die Daten abgelegt. Für die Berechtigung wird das übermittelte Token in der URL validiert.

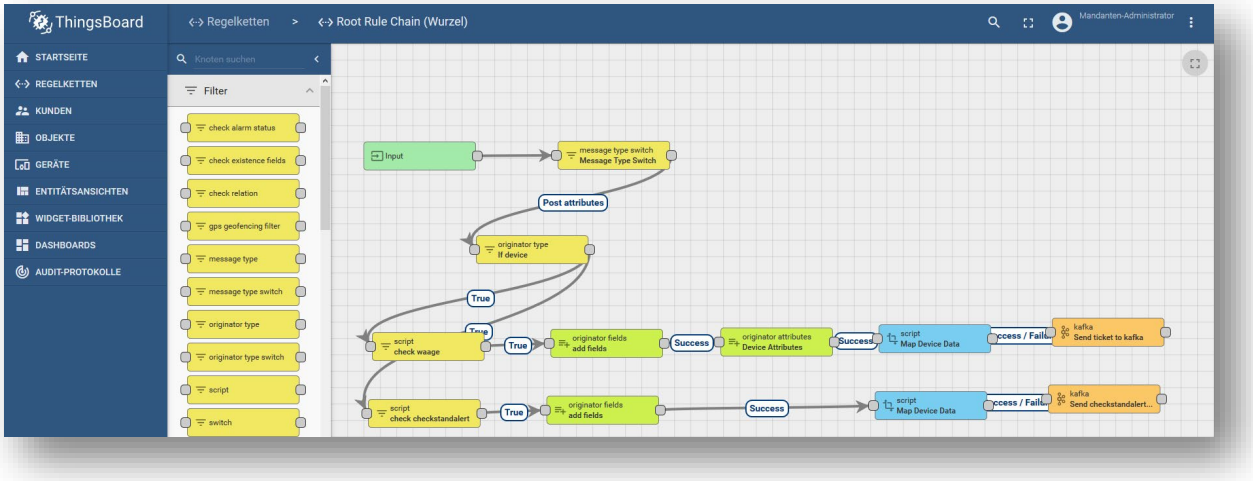


Abbildung 23 Thingsboard Regelketten

Die eingebaute Regelverwaltung[2] ist eine der Kernfunktionalitäten von Thingsboard. Für das UCC Projekt ist ein Regelfluss definiert, dabei wird die HTTP (Hypertext Transfer Protocol) Anfrage durch Thingsboard angenommen und an das Regelwerk weitergegeben. Das Regelwerk hat einen Eingang und einen Ausgang. Die Hauptaufgabe des Regelwerks besteht darin, eine kontrollierte Ausgabe bereitzustellen. Das Regelwerk kann anhand von definierten Regeln entscheiden, ob beispielsweise ein kritischer Punkt erreicht oder eine Kondition erfüllt ist und somit die Anfrage weitergegeben werden kann. Die Abbildung 23 Thingsboard Regelketten zeigt das Regelwerk für das UCC Projekt. Hier werden als Voraussetzung die Waage bzw. der Knopf erwartet. Wenn die Bedingung erfüllt ist, werden die gerätespezifischen Daten ermittelt und eine neue Payload erstellt. Diese Nachricht wird anschließend an das Kafka Plugin übermittelt.

[2] Siehe: https://thingsboard.io/docs/user-guide/rule-engine-2-0/re-getting-started/

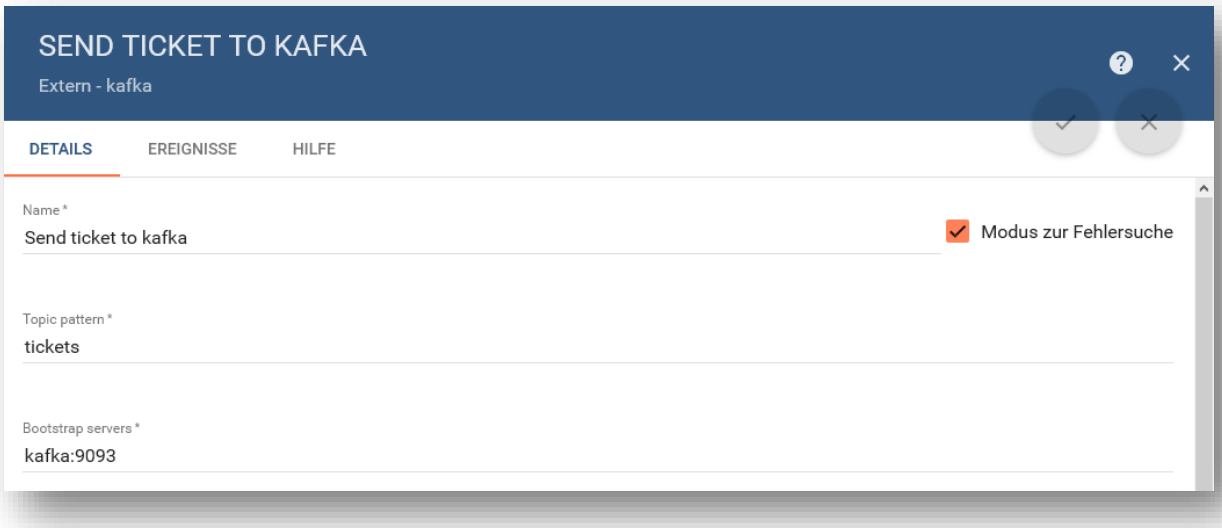


Abbildung 24 Thingsboard Kafka Plugin

Das Kafka Plugin[3] ist standardmäßig in jeder Thingsboard Version vorhanden. Innerhalb des Regelwerks kann auf diesen Kafka-Knoten zugegriffen werden. Damit die Verbindung zu Kafka funktioniert muss mindestens das Topic Name und der Port angegeben sein (siehe Abbildung 24 Thingsboard Kafka Plugin).

6.5 Pushbutton

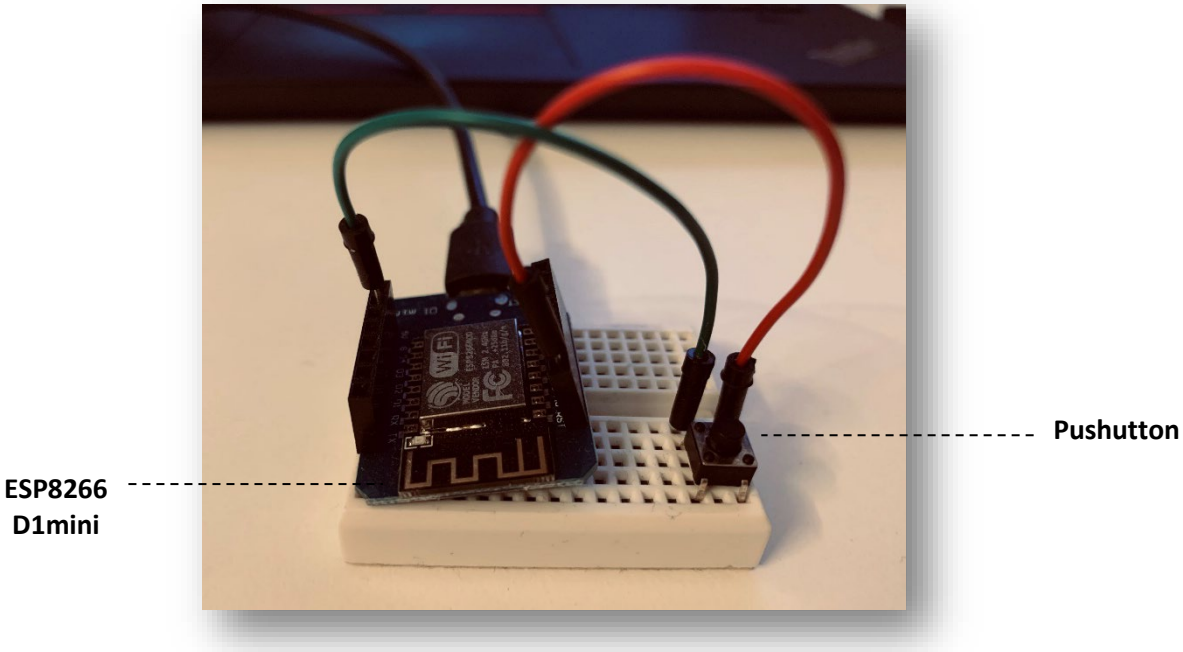


Abbildung 25 Physischer Knopf

[3] Siehe: https://thingsboard.io/docs/reference/plugins/kafka/

Die Hardware des Pushbuttons besteht aus einer ESP8266 D1 mini, einem Knopf, einer Lochrasterleiterplatte, zwei Kabel Steckbrücken und einem USB-Kabel. Abbildung 25 Aufbau physischer Knopf zeigt den Aufbau dieser Komponenten. ESP8266 ist mit einem Wifi-Board[4] ausgestattet, das heißt jede URL kann damit erreicht werden. Durch das Anschließen des Knopfs kann der Trigger ausgelöst und mit Hilfe der geflashten Software die Thingsboard URL Kassenalert aufgerufen werden.

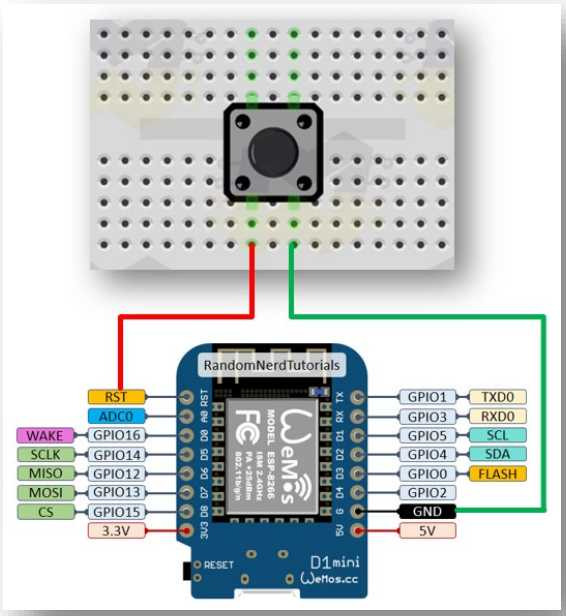


Abbildung 26 Aufbau physischer Knopf

Die Arduino IDE bietet eine Entwicklungsumgebung für die Entwicklung des Scripts an. Anschließend kann das ESP8266 Modul via USB-Kabel an den Computer angebunden und mit der Arduino Software geflasht werden. Die Entwicklungsumgebung erkennt das Modul und schreibt diese Datei auf die Komponente. Damit die Verbindung zu einem WLAN-Gerät funktioniert muss die SSID und das Passwort in der geflashten Datei hinterlegt sein. Das Skript befindet sich im Projektordner.

[4] WLAN-Standard 802.11b/g/n

7. Installation und Konfiguration

Die Architektur erfordert das Zusammenspiel mehrerer Software Komponenten. Die Konfiguration und Installation sind dadurch umfangreich. Eine manuelle Konfiguration ist daher sehr fehleranfällig. Darum wurden die gesamte Konfiguration und Installation automatisiert, sodass keine manuellen Eingriffe notwendig sind.

In diesem Abschnitt wird die bestehende Automatisierung und die dafür eingesetzten Tools beschrieben.

7.1 GitHub Actions & Ansible

Das gesamte Projekt wird auf GitHub gehostet. Als Contionous Integration und Deployment Lösung bietet GitHub das Produkt Actions an. Actions erlauben auf Interaktionen zu reagieren und einen Workflow zu starten.

Das Projekt verwendet Actions, um die Installation auf einem Server vorzunehmen. Der Trigger für den Start der Action ist ein git push auf das Repository.

Actions können komplexe Installationen vornehmen. Allerdings ist die Möglichkeit für komplexere Deployments sehr umständlich. Darum wird zusätzlich Ansible verwendet. Ansible ist ein, mittlerweile Industriestandard, für Konfigurations- und Installationsverwaltung. Die nachfolgende Abbildung stellt den gesamten Installationsvorgang dar.

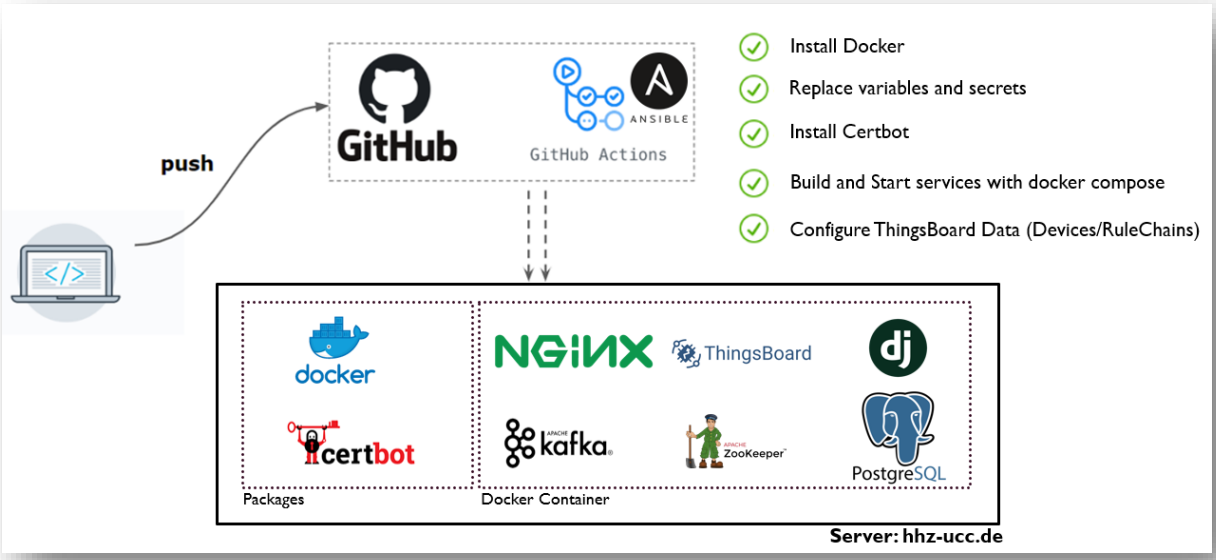


Abbildung 27 Automatisierung Installation

Die GitHub Action startet Ansible. Die Ansible Skripte liegen auch im selben Repository. Ansible installiert auf dem Ziel System Docker, führt Variablen Ersetzungen durch, installiert Certbot und startet mittels docker compose die auf der Abbildung 27 dargestellten Docker Container.

Auf Certbot, Nginx sowie Docker wird in den nachfolgenden Abschnitten detaillierter eingegangen.

Ansible hat ein Inventory Konzept. Ein Inventory ist eine Beschreibung des Zielsystems mit allen notwendigen Informationen wie beispielsweise der Hostname. Ansible verbindet sich mit dem Ziel System über SSH. Für die Authentifizierung ist es daher notwendig, dass im Zielsystem der Public Key des SSH Private Keys hinterlegt ist. Mit dem Private Key, das Ansible bei der Ausführung übergeben wird, findet dann die Authentifizierung statt.

Es werden diverse Passwörter für die Installation benötigt. Passwörter als Plain Text in das Repository zu übertragen ist aus Sicherheitsgründen nicht empfehlenswert. Darum bietet Ansible eine Verschlüsslungsfunktionalität an. Die Passwortdatei wird dabei vor dem push in das Repository mit einem Password verschlüsselt und erst dann übertragen. Damit Ansible die Passwortdatei bei der Ausführung in der GitHub Action wieder entschlüsseln kann, wird das Passwort als Secret in GitHub hinterlegt. Die nachfolgende Abbildung zeigt an welcher Stelle in GitHub das Passwort hinterlegt wird.

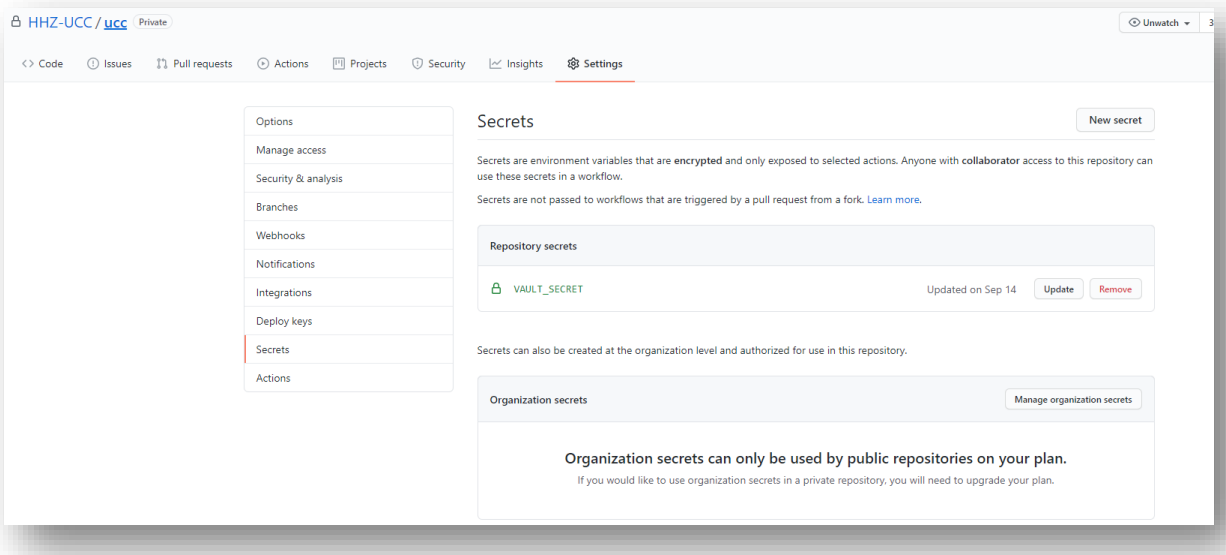


Abbildung 28 GitHub Secrets

Grundsätzlich könnten alle Passwörter als Secret in GitHub hinterlegt werden.

7.2 Docker & Docker Compose

Anwendungen erfordern mehrere Voraussetzungen, die ein System erfüllen muss, damit die Anwendung installiert werden und verwendet werden kann. Beispielsweise erfordert eine Django basierte Applikation, das Python in einer bestimmten Version auf dem Server installiert ist. Es können dadurch Konflikte entstehen, da andere Anwendungen auf dem System wiederum eine andere Python Version benötigen. Docker ist eine Technologie zur Isolierung von Software mittels Container und beugt dieser Problematik vor.

Docker erlaubt es Images zu erstellen und als Container auszuführen. Ein Image kann dabei jede Software beinhalten. Softwarehersteller können Images erstellen und über Docker Hub zur Verfügung stellen.

Für das Projekt werden die in der nachfolgenden Tabelle dargestellten Images verwendet.

Image und Version	Beschreibung
nginx:1.19.1	NGINX wird als Gateway verwendet
thingsboard/tb-postgres:2.5.2	ThingsBoard als IOT-Plattform
postgres:12.3-alpine	Als Datenbank für ThingsBoard
node:12.16.0-stretch-slim	Als Base Image für den Bot Service, da dieser auf node.js basiert
python:3.8.3-alpine	Als Base Image für Django Services

Tabelle 8 Docker Images

Da der Bot Service und die Django Services eine eigene Entwicklung sind, werden die Images hierfür im Rahmen der Installation gebaut. Für das Bauen eines Images wird ein Dockerfile benötigt. Das Dockerfile enthält Installationsanweisungen, welche im Rahmen des Image Builds ausgeführt werden.

Das nachfolgende Dockerfile ist das des Bot Services. Das Dockerfile der Django Services ist ähnlich und ebenfalls im Repository vorhanden.

```
# 1. NodeJS als Base Image
FROM node:12.16.0-stretch-slim


#2. Erstellt das Arbeitsverzeichnis innerhalb des Images
WORKDIR /usr/src/app

#2. Installiert alle Libraries die für den Bot Service notwendig sind
COPY package*.json ./
RUN npm install
COPY . .

#3. Setz den Port, auf dem der Bot Service erreichbar sein soll
ENV PORT=8080
EXPOSE 8080

#4. Startet den Bot Service
CMD [ "node", "index.js" ]
```

Code 3 Automatisierungsskript

Docker Container können mittels Kommando Zeile gestartet werden. Bei mehreren Containern ist das Fehleranfällig und umständlich. Hierfür eignet sich Docker Compose. Im YAML Format werden alle Container und die jeweils notwendigen Konfiguration angegeben können mittels Docker Compose, ein Kommandozeilen Programm, gestartet werden.

7.3 Nginx & Certbot

Alle Komponenten des Projektes werden auf einem Server installiert. Für die Adressierung eines bestimmten Service ist immer die IP-Adresse oder Hostname und der Port erforderlich. Eine portbasierte Weiterleitung auf die entsprechenden Anwendungen ist nicht gängig und hat den Nachteil, dass der Benutzer oder die Clients die Ports kennen müssen.

Nginx ist ein Proxy Server und erlaubt pfadbasierte Weiterleitungen von Anfragen. Die Ports müssen nur noch intern für Nginx bekannt sein.

Folgende Pfade sind in Nginx für das Projekt definiert. Nginx Leitet Anfragen auf einen dieser Pfade an die entsprechende Anwendung weiter.

Pfad	**Anwendung**
https://hhz-ucc.de	ThingsBoard
https://hhz-ucc.de/bot	Bot Service
https://hhz-ucc.de/services/ https://hhz-ucc.de/services/registry https://hhz-ucc.de/services/maintenance https://hhz-ucc.de/services/checkstandalert	Django Services Registry Service Maintenance Service Checkstandalert Service

Tabelle 9 Nginx Applikationspfade

Aus Sicherheitsgründen ist es erforderlich Anfragen und Antworten mittels SSL zu verschlüsseln. Zusätzlich erlaubt Microsoft Teams keine unverschlüsselte Kommunikation. Langjährige gültige SSL Zertifikate sind kostenpflichtig. LetsEncrypt, ein SSL Zertifikat Provider, stellt freie SSL Zertifikate mit einer kurzen Laufzeit zur Verfügung. Um zu vermeiden, dass die SSL Zertifikate von LetsEncrypt ablaufen, müssen diese jeden Tag erneuert werden. CertBot ist eine Software, welche eine automatische Verlängerung der Zertifikate erlaubt. Dazu muss CertBot jeden Tag gestartet werden und kümmert sich um den kompletten Aktualisierungsvorgang. Mittels Ansible wird ein CronJob erstellt das CertBot jeden Tag ausführt. Dadurch wird ein manueller Schritt vermieden.

7.4 Übersicht der Umgebungsvariablen

Es sind mehrere Umgebungsvariablen im Einsatz. Die nachfolgende Tabelle stellt diese im Detail dar.

Allgemeine Variablen		
Variable	**Wert**	**Beschreibung**
HOST	https://hhz-ucc.de	Protokoll und Hostname
HOSTNAME	hhz-ucc.de	Hostname

Postgres		
POSTGRES_USER	tbadmin	Datenbank Username
POSTGRES_PASSWORD	******	Datenbank Password
POSTGRES_DB	public	Datenbank Schema
ThingsBoard		
SPRING_DATASOURCE_URL	jdbc:postgresql://postgres-thingsboard:5432/public	Datenbank Connection String
SPRING_DATASOURCE_USERNAME	tbadmin	Datenbank Username
SPRING_DATASOURCE_PASSWORD	******	Datenbank Password
THINGSBOARD_USER	tenant@thingsboard.org	Login für das ThingsBoard UI
THINGSBOARD_PASSWORD	******	Password des Users
Django Services		
SECRET_KEY	******	Verschlüsselung von Web Sessions
ENABLE_KAFKA_CONSUMER	True	Bestimmt ob Kafka Consumer aktiviert werden soll. Ausschaltbar in der Entwicklung.
Bot Service		

MICROSOFT_APP_ID	1d3cc081-d909-486b-b581-8a704cfd63b8	Die ID der in Microsoft Azure hinterlegten Bot App. Dadurch wird die Authentifizierung mit Microsoft Teams ermöglicht.
MICROSOFT_APP_PASSWORD	******	Passwort der Microsoft App.
Kafka		
KAFKA_CREATE_TOPICS	tickets:1:1,checkstandalert:1:1	Topics die beim Start von Kafka angelegt werden.

Tabelle 10 Umgebungsvariablen

7.5 Wichtige Links für die Entwicklung

In der nachfolgenden Tabelle sind Links zu den eingesetzten Werkzeugen. Im Rahmen der Entwicklung und Einarbeitung wurden primär auf die verlinkten Seiten zugegriffen.

Werkzeug / Technologie	**Beschreibung & Links**
Visual Studio Code	Editor mit Unterstützung für Dev Container, die eine Entwicklung innerhalb eines Containers durchzuführen. Der Bot Service und die Django Services wurden damit entwickelt und haben bereits im Ordner die Konfigurationsdatei. In der Verlinkten Dokumentation ist das Konzept der Dev Container beschrieben. https://code.visualstudio.com https://code.visualstudio.com/docs/remote/containers

Docker	Docker ist für die Entwicklung sowie die Installation notwendig. Über den Nachfolgenden Link kann Docker heruntergeladen und installiert werden. https://www.docker.com/get-started
Django	Die Services wurden mit Django implementiert. Unter Nahfolgendem Link sind die Ersten Schritte mit Django sehr gut erklärt. https://www.djangoproject.com/start/
Microsoft Bot Framework / Adaptive Cards	Der Bot Service wurde mit der JavaScript SDK des Bot Frameworks implantiert. Unter dem Nachfolgende Link ist bietet einen guten Einstieg. https://docs.microsoft.com/en-us/azure/bot-service/javascript/bot-builder-javascript-quickstart?view=azure-bot-service-4.0 https://adaptivecards.io/designer/
Ansible	Mit Ansible wird die Installation und Konfiguration des Projektes auf einem Ziel Sever vorgenommen. Der nachfolgende Link gewährt einen Einstieg in die Entwicklung mit Ansible. https://docs.ansible.com/ansible/latest/user_guide/intro_getting_started.html

Tabelle 11 Wichtige Links für die Entwicklung

7.6 Server und Repository

Die nachfolgende Tabelle enthält Informationen zu dem GitHub Repository und dem verwendeten Server.

GitHub Repository	https://github.com/HHZ-UCC/ucc
Server	https://hhz-ucc.de/

Tabelle 12 Server und Repository

8. Fazit und Ausblick

8.1 Erreicht

In diesem Projekt wurde eine Grundlage für den Einsatz von Bots mit Hilfe von Microsoft Teams als Client untersucht, protypisch entwickelt und anschließend eine Referenzarchitektur erstellt. Als Schwerpunkt der Untersuchung wurde die Kommunikation von Menschen und Maschinen in einem Retail Store simuliert. Für das Proof of Concept wurde eine simulierte internetfähige Waage und ein ebenfalls internetfähiger Push-Button eingesetzt. Die Geräte können per Push Nachrichten Aufgaben erstellen und diese in einem Microsoftkanal veröffentlichen. Die Endanwender können sich die erstellten Aufgaben zuweisen und nach offenen Aufgaben Ausschau halten. Die Referenzarchitektur bietet eine leichte Erweiterbarkeit von weiteren Diensten an, hierzu ist der Ansatz von Plug and Play und Hateoas (vgl. 5.1 Referenzarchitektur) verfolgt worden. Das Projekt hat gezeigt, dass ein sehr großes Potential in diesem Segment vorhanden ist.

8.2 Ausblick

Im Folgenden Abschnitt werden drei Empfehlungen für ein Folgeprojekt beschrieben.

- Integrationsmöglichkeiten von Spracheingaben

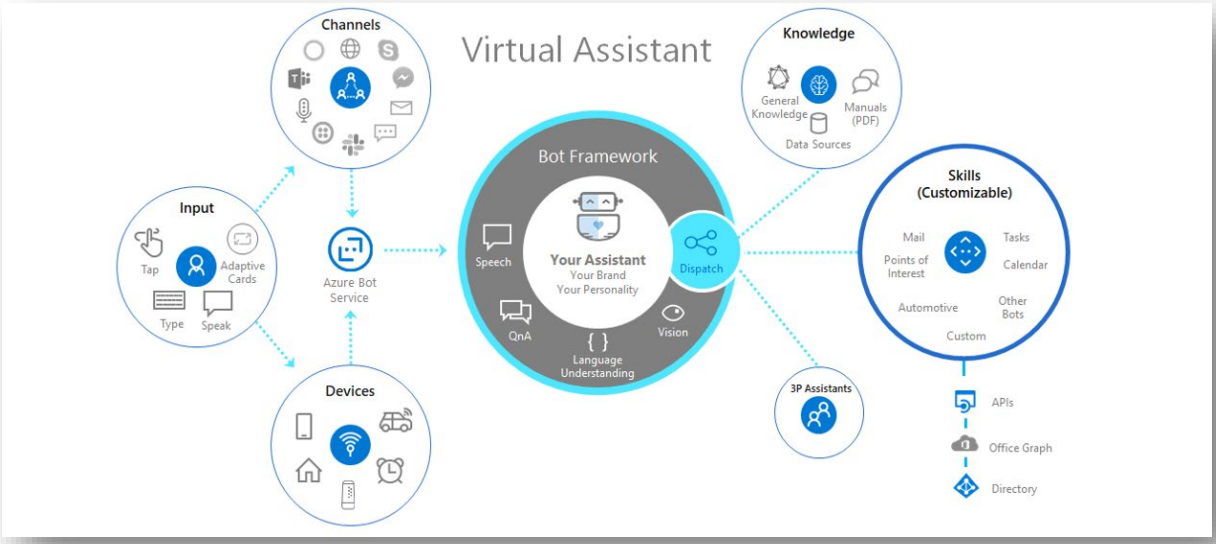


Abbildung 29 Microsoft Virtueller Assistent [MIC20]

In einer Filiale ist die Kommunikation via Spracheingabe gesetzt, dass bedeutet die technischen Voraussetzungen sind mittels Headsets bereits vorhanden. Das eröffnet die Türen einer sprachgesteuerten Kommunikation mit einem Bot. Microsoft unterstützt mit dem Speech-Dienst[5] die Integration zu einem Bot. Für das Einbinden des Bots ist eine Registrierung beim Direct Line Speech-Kanal[6] notwendig. Dieser Kanal erstellt eine Verbindung zwischen dem Bot und der Client-App Microsoft Teams. Als nächstes muss in der Bot Applikation die vorhandene Speech SDK integriert werden. Anschließend kann die Funktion durch einen Schlüsselworterkennungsfähigkeit[7] erweitert werden. Mit Hilfe der Spracherkennung kann der Bot durch einen Schlüsselbefehl gestartet und eine Anfrage mittels Spracheingabe erstellt und versendet werden. Für eine ausführliche Kontextanalyse wird eine KI basierte Textanalyse empfohlen, Abschnitt Kontextbasierte Textanalyse für Bot („Intelligent Bot“) gibt einen Einblick in die Thematik. Weitere Anwendungsfälle sind in der Abbildung 9 Microsoft Virtueller Assistent für den Virtuellen Assistent Bot abgebildet.

- UX Konzepte für den Bot

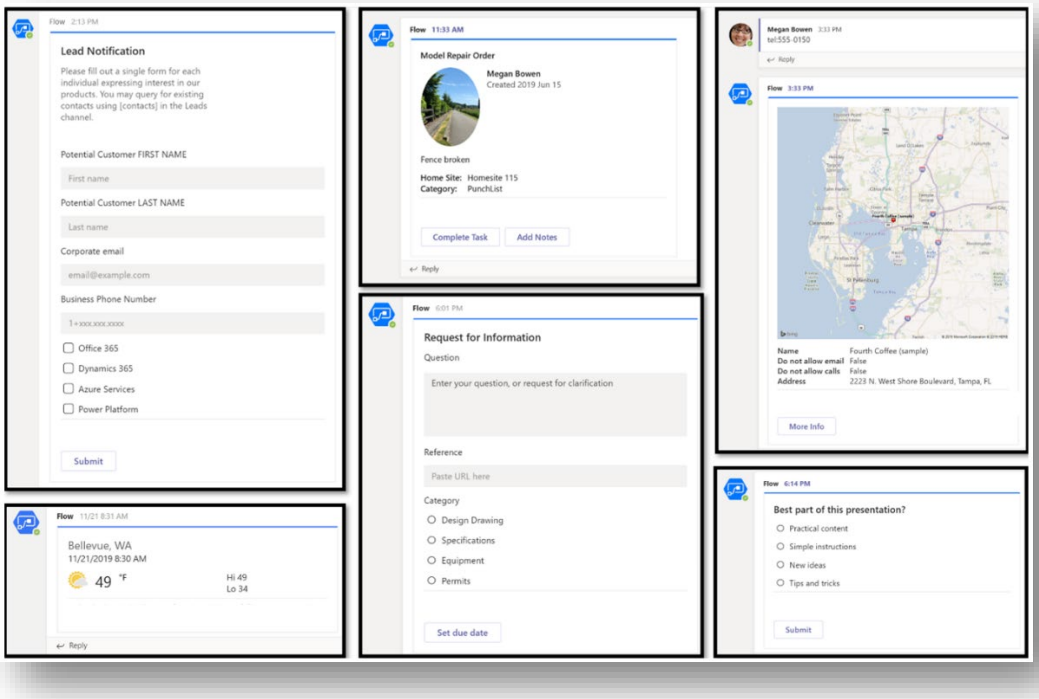


Abbildung 30 Microsoft Adaptive Cards Beispiele [MAC20]

[5] Siehe: https://docs.microsoft.com/de-de/azure/cognitive-services/speech-service/tutorial-voice-enable-your-bot-speech-sdk

[6] Siehe: https://docs.microsoft.com/de-de/azure/bot-service/bot-service-channel-connect-directlinespeech?view=azure-bot-service-4.0

[7] Siehe: https://docs.microsoft.com/de-de/azure/cognitive-services/speech-service/custom-keyword-basics

Microsoft Adaptive Cards ist technisch und visuell eine sehr elegante Lösung Informationen und Aufgaben darzustellen. Adaptive Cards eignen sich hervorragend für Bots. Um eine gezielte UX zu erreichen, wird empfohlen einen Discovery Workshop mit den Key Usern durchzuführen[8]. Dabei ist es wichtig auf die Bedürfnisse und Anforderungen der Anwender einzugehen und diese bestmöglich in Adaptive Cards umzusetzen. Abbildung 30 Microsoft Adaptive Cards Beispiele zeigt einen kleinen Ausschnitt von möglichen Design Layouts. Alle Cards werden in Form von JSON Objekten erstellt und zur Laufzeit im Bot abgelegt. Dieser rendert den Anhang und zeigt diese in der Endanwendung. Da sich adaptive Karten an ihren Host anpassen, sind sie perfekte Instrumente für den Informationsaustausch zwischen Microsoft Teams und anderen Diensten. Die adaptive Card Designer[9] Webseite bietet einen sehr anwenderfreundlichen Service für die Erstellung der Cards an. Hier kann der Designer entscheiden, ob eine Vorlage als Grundlage verwendet oder die Card initial erstellt wird. Mit Hilfe von Drag und Drop können die Elemente im Designer platziert und anschließend kann die generierte JSON Datei dem Bot anhängt werden.

- Kontextbasierte Textanalyse für Bot („Intelligent Bot“)

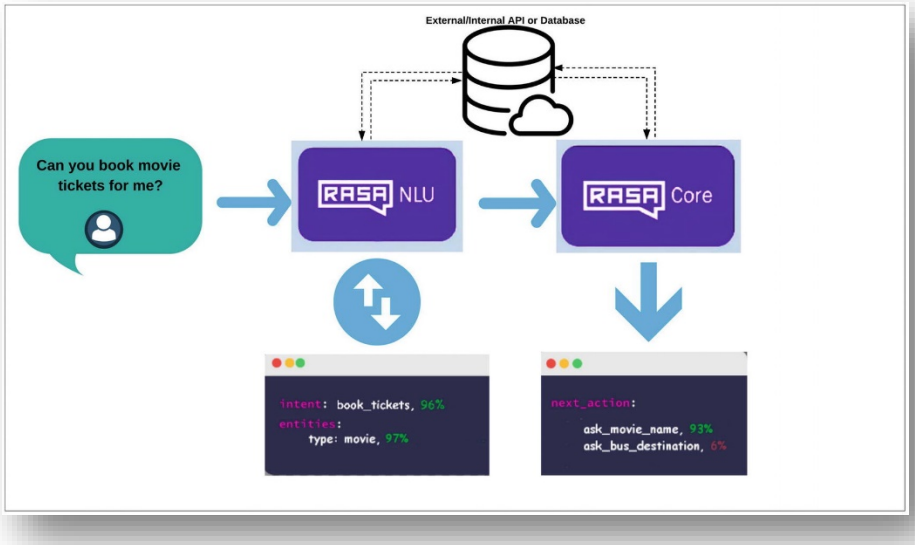


Abbildung 31 Rasa AI Überblick [RAI20]

[8] Siehe: https://www.nngroup.com/articles/5-ux-workshops/

[9] Siehe: https://adaptivecards.io/designer/

In der aktuellen Version ist das Bot nicht in der Lage eine Konversation durchzuführen. Die Textnachricht wird zurzeit nicht analysiert, es dient nur zum Starten des Bots. Dabei ist egal was als Nachricht eingegeben wird. Jede Anfrage wird mit einer Willkommen Adaptive Card erwidert. Der Anwender kann sich dann durch die Adaptive Cards durchklicken, das bedeutet, der Anwender ist nicht in der Lage eine konkrete Abfrage abzuschicken. Dieser Prozess kann durch den Einsatz von KI-basierten Textanalyse Plattformen[10], wie beispielsweise einer Opensource Lösung von Rasa.ai, Microsoft Luis oder Watson Assistent verbessert werden. Abbildung 9 Rasa AI Überblick zeigt den Aufbau von Rasa.ai. Dabei wird die Bearbeitung in zweite Schritte unterteilt. Zunächst wird mit Hilfe des NLU (Natural Language Understanding) Prozesses der Kontext ermittelt. Im zweiten Schritt wird dann eine wahrscheinliche Antwort generiert, in der Rasa Plattform ist der Rasa Core für diesen Schritt verantwortlich. Dabei ist es wichtig im ersten Schritt ein gut trainiertes KI-Modell für einen bestimmten Einsatz zu erstellen. Für das Folgeprojekte muss zunächst entschieden werden, welchen Einsatzzweck man mit dem BOT plant. Es ist aktuell sehr unwahrscheinlich, dass ein Bot alle Anfragen abdecken kann. Der Bot ist nur so schlau, wie er auch trainiert ist.

[10] Siehe: https://www.bigdata.fraunhofer.de/content/dam/bigdata/de/documents/Publikationen/BMBF_Fraunhofer_ML-Ergebnisbericht_Gesamt.pdf

9. Literaturverzeichnis

[DA20] https://djangobook.com/wp-content/uploads/admin_index_models.png (zuletzt abgerufen am 13.10.2020)

[HAT20] HATEOAS. 2018. Von https://en.wikipedia.org/wiki/HATEOAS (zuletzt abgerufen am 26.09.2020)

[KAF17] Jacobs, S.: Apache Kafka. 2017. Von https://blog.oio.de/2017/11/21/apache-kafka/ (zuletzt abgerufen am 26.10.2020)

[MAC20] https://docs.microsoft.com/en-us/power-automate/media/adaptive-cards/multi-adaptive-cards.png (zuletzt abgerufen am 05.10.2020)

[MIC20] Virtueller Assistent für Microsoft Teams. 2020. Von https://docs.microsoft.com/de-de/microsoftteams/platform/samples/virtual-assistant (zuletzt abgerufen am 11.10.2020)

[MST20] Microsoft Teams. 2020. Von https://de.wikipedia.org/wiki/Microsoft_Teams (zuletzt abgerufen am 23.08.2020)

[RAI20] Rasa. 2020. Von https://rasa.com/ (zuletzt abgerufen am 07.08.2020)

[THI20] Working with IoT device attributes. 2020. Von https://thingsboard.io/docs/user-guide/attributes/ (zuletzt abgerufen am 01.10.2020)

[TI20] HTTP Device API Reference. 2020. Von https://thingsboard.io/docs/reference/http-api/ (zuletzt abgerufen am 03.10.2020)

[UCC18] Srocke, D.; Donner, A.: Was ist UCC? 2018. Von https://www.ip-insider.de/was-ist-ucc-unified-communications-collaboration-a-581310/ (zuletzt abgerufen am 05.09.2020)