

Masterarbeit

**Entwicklung von intelligenten Türschildern
auf der Basis von IoT in einem dezentralen
Netzwerk**

Tolgay Usul
Juni.2019

Gutachter:
Dr. Lars Hildebrand
Prof. Dr. Jakob Rehof

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhlbezeichnung (LS14)
<http://ls14-www.cs.tu-dortmund.de>

In Kooperation mit:
Fakultät für Informatik
Lehrstuhl 14 für Software Engineering

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation und Hintergrund	1
1.2	Ziel	1
1.3	Aufbau der Arbeit	2
2	Grundlagen	3
2.1	IoT-Hardware	3
2.1.1	Raspberry Pi 3	3
2.1.2	Uno Rev 3	4
2.1.3	Esp-Wrover-Kit-VB	5
2.2	Netzwerke	7
2.2.1	Zentrales Netzwerk	7
2.2.2	Dezentrales Netzwerk	8
2.2.3	Distribuiertes Netzwerk	8
2.3	IoT Betriebssysteme	9
2.3.1	Windows 10 IoT	9
2.3.2	Raspbian Linux-Distribution	12
2.3.3	Android Things	13
2.3.4	Weitere IoT Betriebssysteme	14
2.4	SOAP und REST	16
2.4.1	SOAP	16
2.4.2	REST	19
2.4.3	Vergleich zwischen REST and SOAP	20
2.5	Sprachen für die Modellierung grafischer Benutzeroberflächen	22
2.5.1	XAML	22
2.5.2	XML für Android Things	24
2.6	Programmierumgebung	26
2.6.1	C#	27
2.6.2	Microsoft Visual Studio 2017	28
2.6.3	Universelle Windows Plattform (UWP)	29

2.6.4 Windows Runtime (WinRT)	30
3 Anforderungen und Konzept	33
3.1 Anforderungsanalyse	33
3.2 Konzept	34
3.2.1 Auswahl der Hilfsmittel	34
3.2.2 Grobe Planung der einzelnen Anwendungen	35
4 Anwendungsentwurf und -implementierung	49
4.1 Architektur der Anwendung	49
4.1.1 Kommunikationswege der Anwendung	49
4.1.2 Aufbau der Anwendungen	50
4.2 Frameworks und Erweiterungen	52
4.3 Serverseitige Implementierung	55
4.3.1 Layout der Anwendung (Foreground-APP)	55
4.3.2 REST-Schnittstelle (Background-APP)	63
4.3.3 Nachrichtenkanal (UWP-MessageRelay)	68
4.3.4 Prüfungsapplikation (ForeGround-Prufung)	74
4.4 Clientseitige Implementierung	74
4.4.1 Layout der Anwendung	74
4.4.2 Implementierung der Anwendung	77
4.4.3 Speicher- und Profilsystem	83
5 Benutzerhandbuch	85
5.1 Installation	85
5.1.1 Installationsvoraussetzungen	85
5.1.2 Ausführen der Anwendung	85
5.2 Verbindungsvorgang	85
5.2.1 Verbindung hinzufügen	86
5.2.2 Verbindung löschen	86
5.2.3 Verbindungen überprüfen	86
5.2.4 Eigenschaftsfenster öffnen	87
5.3 Eigenschaftsfenster	87
5.3.1 Gerät	87
5.3.2 Ein- / und Ausschalten von Kategorien	89
5.3.3 Infoleiste verändern	89
5.3.4 Profile erstellen/verändern	89
5.3.5 Textnachricht bereitstellen	90
5.3.6 Veranstaltung hinzufügen	90
5.3.7 Belegungsplan	91

5.3.8 Wetter anzeigen lassen	91
5.4 Prüfungsalarm aufrufen	92
5.5 Abrufen der Kategoriedaten von der IoT-Anwendung	92
5.6 Daten senden	93
6 Zusammenfassung und Ausblick	95
6.1 Zusammenfassung	95
6.2 Ausblick	96
Abbildungsverzeichnis	101
Algorithmenverzeichnis	103
Literaturverzeichnis	107
Erklärung	107

Kapitel 1

Einleitung

Das Thema „Internet of Things“ (kurz: IoT) leitet ein neues Zeitalter der Digitalisierung ein. Durch den Ausbau des globalen Internets ergeben sich mehrere Möglichkeiten Anwendungen zu entwickeln, die den Alltag des Menschen erleichtern. Das Abrufen und Anzeigen von relevanten Daten ohne menschliche Interaktionen, ist der nächste Schritt der technischen Revolution. Diese Hilfestellungen können in mehreren unterschiedlichen Bereichen stattfinden. Beispielsweise können seismische Aktivitäten an wissenschaftliche Fakultäten in Echtzeit weitergeleitet werden. Eventuelle Schäden durch Naturkatastrophen können vorher erkannt und minimiert werden.

1.1 Motivation und Hintergrund

Informationen über Veranstaltungen und mögliche Sprechstunden sind häufig nur über Umwege abrufbar. Diese können in anderen Räumen stattfinden oder sogar ausfallen. Dozenten und wissenschaftliche Mitarbeiter müssen des Öfteren Termine und Veranstaltungen verschieben oder verlegen. Die Teilnehmer erkennen dies meist sehr spät. Sie sind über die derzeitige Lage nicht informiert und können aufgrund stetigen negativen Erfahrungen das Interesse am Themengebiet verlieren. Im schlimmsten Fall können Sie auf weitere Besuche von zukünftigen Veranstaltungen und Sprechstunden verzichten. Um solchen Situationen entgegenzuwirken, sollen intelligente Türschilder aktuelle Informationen über Räume beinhalten, die den Teilnehmern der Veranstaltung in Echtzeit vermittelt werden. Kurzfristige Änderungen über Veranstaltungen sind sofort erkennbar.

1.2 Ziel

In dieser Ausarbeitung soll ein IoT System implementiert werden. Mit der Eingabe von Daten soll die implementierte Software in der Lage sein mehrere Profile, Ankündigungen,

einen Belegungsplan, sowie Veranstaltungen übersichtlich anzuzeigen. Die Eingaben bleiben dabei schlicht. Es sollen keine Vorkenntnisse für die Benutzung der Anwendung notwendig sein. Die Installation der Anwendung auf dem IoT Gerät und auf dem Computer werden auf ein Minimum beschränkt. Das schließt die Verwendung von zusätzlicher Software wie einer Datenbank aus. Das Ziel ist es eine funktionsfähige und anwenderfreundliche Gesamtapplikation zu erstellen.

1.3 Aufbau der Arbeit

Diese Arbeit besteht aus mehreren Kapiteln. Die Arbeit ist auf einem festen Schema aufgebaut. Erklärungen von Begriffen und Thematiken in vorherigen Abschnitten werden in den danach folgenden Kapiteln nicht mehr thematisiert. Es wird eine Referenz zu den vorherigen Abschnitten eingebettet.

Kapitel 2 befasst sich mit den technischen Grundlagen für die Anwendungsentwicklung. Hier wird eine Anzahl von möglichen Hilfsmittel vorgestellt und erläutert. Die Erklärungen sind dabei objektiv aufgebaut und frei von jeglicher eigener Meinung.

In Kapitel 3 werden die primären und optionalen Anforderungen, die für eine erfolgreiche Abnahme der Anwendung Voraussetzung sind, vorgestellt. Dem folgt eine begründete Auswahl der Hilfsmittel aus dem Grundlagenkapitel 2. Diese stehen in Bezug zu den Anforderungen und der Architektur, der zu erstellenden Anwendungen. Zusätzlich enthält das Kapitel Skizzen mit den Oberflächen und den Prozeduren der Anwendungen.

Die Umsetzung der Konzeption und die Realisierung der Anforderungen findet in Kapitel 4 statt. Dies ist der Hauptteil dieser Ausarbeitung. Mithilfe von eingebetteten Quellcodes aus den Prozeduren werden Funktionsweise und Oberfläche der Anwendungen erläutert. Die Architektur der Anwendungen werden in diesem Kapitel finalisiert. Zusätzliche Hilfsmittel, die für die Implementierung notwendig sind, werden aufgezeigt.

Anschließend wird in Kapitel 5 ein Benutzerhandbuch zur Verfügung gestellt. Dieses Handbuch erklärt die Verwendung aller implementierten Funktionen. Die Erklärungen sind bebildert und in Schlagwörter unterteilt.

Das letzte Kapitel 6 beinhaltet eine kurze Zusammenfassung dieser Ausarbeitung. Weiterhin sind mögliche Erweiterungen aufgeführt.

Kapitel 2

Grundlagen

In diesem Abschnitt werden die notwendigen Grundlagen vermittelt, die für das Verständnis der weiteren Abschnitte notwendig sind. Es werden grundlegende Informationen über die genutzte Hardware, die Architektur und das Betriebssystem erläutert.

2.1 IoT-Hardware

Für das Projekt müssen verschiedene kompatible IoT Boards ausgewertet und analysiert werden. In diesem Unterabschnitt werden drei IoT Boards aufgeführt und miteinander verglichen.

2.1.1 Raspberry Pi 3

Ein Raspberry Pi ist ein Computer mit minimalisierter Hardware. Es ist darauf ausgelegt Personen die Programmierung von Hardware kostengünstig nahezubringen.

Der Hintergrund der Entwicklung des Raspberry Pis war die sinkende Anzahl der Informatikstudenten und der Absolventen mit Programmierkenntnissen. Durch einen günstigen PC wollte man Informatiker das Experimentieren mit diversen Programmiersprachen ermöglichen. Studenten sind nicht mehr gezwungen sich für das Studium teure Heimcomputer anschaffen und können für eine geringe Gebühr ein Raspberry Pi erwerben [21].

Es besteht aus einer einzigen Platine, die etwa die Größe einer Bankkarte besitzt. Anfangs war es nur mit einem ARM-Mikroprozessor ausgestattet, später wurden sie mit Zubehör, wie einer HDMI-Schnittstelle, mehreren USB 2.0 Anschlüssen und einer Videoausgabe erweitert.

Der Unterschied zwischen den Raspberry Pi 3 und dessen Vorgängermodelle ist die CPU, die mit einem 64-bit Quadcore Prozessor und ein 1024 MB Arbeitsspeicher ausgestattet ist. Weiterhin besitzt es ein WLAN- und ein Bluetooth-Modul [11].

Abbildung 2.1 zeigt den äußerlichen Unterschied zwischen den einzelnen Modellen.



Abbildung 2.1: Links Raspberry Pi 1, Mitte Raspberry Pi 2, Rechts: Raspberry Pi 3. Entnommen und leicht verändert aus Quelle [25]

Das Anschließen weiterer Platinen ist mit den eingebauten GPIO-Pins möglich. Die GPIO-Pins dienen sowohl als Eingabe als auch aus Ausgabe. Die Funktion der einzelnen Pins wird durch die Programmierung festgelegt. Ein Raspberry Pi 3 besitzt genau 26 GPIO-Pins, die es ermöglichen das Gerät mit Zubehör zu erweitern und zusätzliche Erweiterungen zu ermöglichen [11].

Abbildung 2.2 zeigt eine Erweiterung des Raspberry Pi 3 mit einem sieben Zoll Display und einem Piface 2 Modul.



Abbildung 2.2: a) Ein Raspberry Pi 3 mit Bildschirm von vorne; b) Ein Raspberry Pi 3 mit Bildschirm von hinten. [Quelle: Selbst erstellt]

2.1.2 Uno Rev 3

Der **Uno Rev 3** von Arduino ist ein kompaktes IoT-Board siehe Abbildung 2.3. Es gilt als Einsteigerboard für die Anwendungsentwicklung.

Es besteht aus einer einzigen Platine, die im Vergleich kleiner und leichter ist als das Raspberry Pi 3.

Der „Uno Rev 3“ verfügt über einen Typ-B USB-Anschluss und einen 32 KB Flashspeicher



Abbildung 2.3: Der „Uno Rev 3“ von Arduino [12]

[23].

Es besitzt 14 digitale und 16 analoge Pins. Ein digitaler Pin kann die Eigenschaften 0 und 1 annehmen. Der analoge Pin nimmt beide Eigenschaften gleichzeitig an. Folgendes Beispiel soll zum Verständnis beitragen:

Wenn 0 schwarz und 1 weiß ist, dann gibt es beim digitalen Pin die Ausgangsstufen Schwarz und Weiß. Beim analogen Fall gibt es eine Mischung von Schwarz und Weiß, also ein Grauton.

Der „Uno Rev 3“ ist ein **Mikrocontroller**. Ein **Mikrocontroller** ist gut geeignet für die schnelle Abwicklung eines Prozesses. Im Gegensatz zu einem **Single Board Computer** werden Anwendungen direkt ohne zusätzliche Einstellungen vom Board ausgeführt [23].

Wegen dieser Eigenschaft ist es nicht in der Lage mehrere Prozesse auf einmal zu koordinieren. Das führt zu einer schlechteren Performance [23].

Das „Uno Rev 3“ wird standardmäßig mit einem Raspberry Pi 3 gekoppelt, um die Nachteile von beiden Seiten auszugleichen.

2.1.3 Esp-Wrover-Kit-VB

Das „Esp-Wrover-Kit-VB“ von Expressif ist ein Entwicklerboard, welches einen integrierten ESP32 Chip mit mehreren Funktionen erweitert. Der ESP32 ist ein 32 Bit Mikrocontroller, der durch seine offene Bauweise eine Vernetzung mit Zusatzbauteile ermöglicht [9].

Bei den Zusatzbauteilen kann es sich um Sensoren und Aktuatoren handeln. Aktuatoren sind mechanische Bauteile, welche elektrische Spannung in maschinelle Bewegungen umleiten. Das Entwickler Kit verbessert den ESP32 mit einem integrierten 3.2 Farbdisplay, sowie einer JTAG, einer UART Schnittstelle, einen USB 2.0 Port, einer Schnittstelle für die Verbindung mit einer Kamera und weiteren Extras.

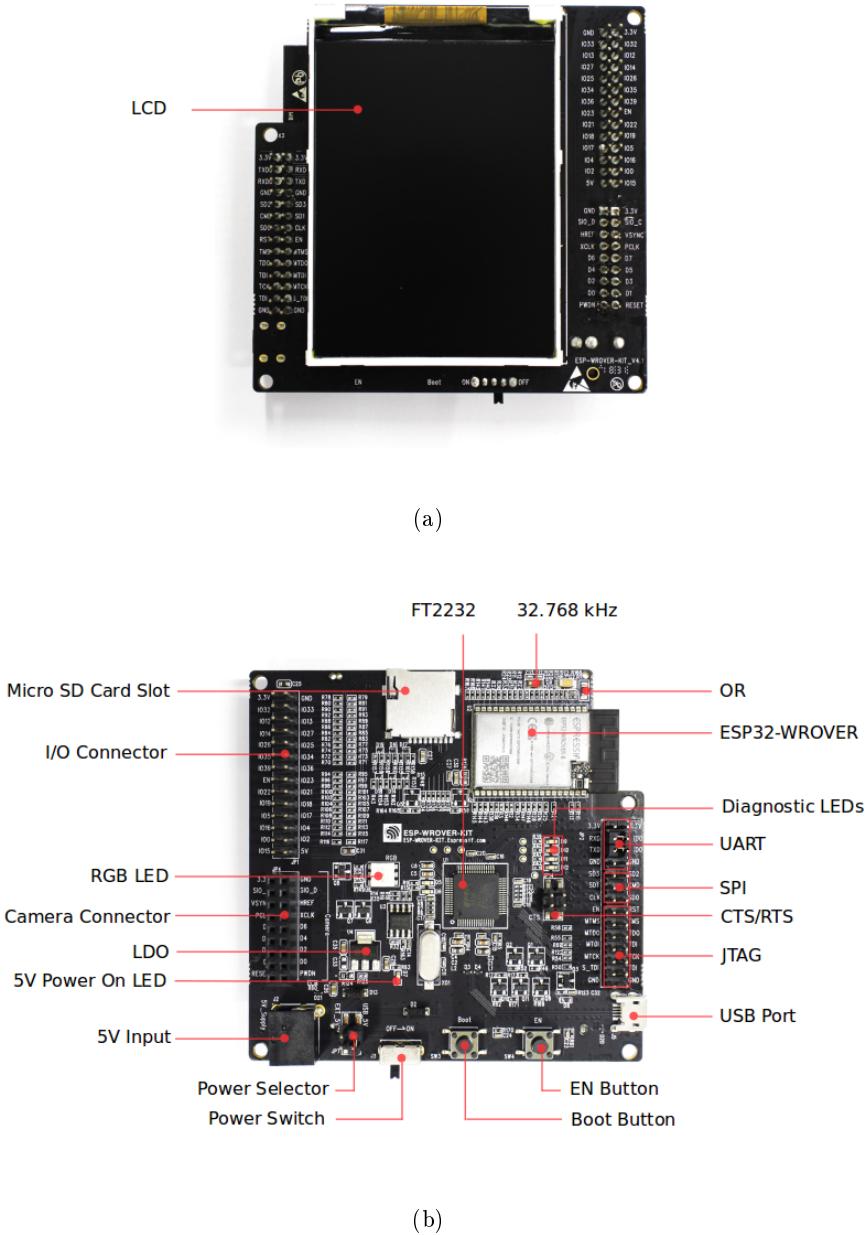


Abbildung 2.4: a) Ein ESP-Wrover-Kit von hinten; b) Ein ESP-Wrover-Kit von vorne [9]

Die Realisierung der Schnittstellen erfolgt durch die 36 GPIO Pins [9].

Abbildung 2.4 bildet das Gerät mit den entsprechenden Schnittstellen auf den Board ab. Trotz des umfangreichen Zubehörs ist der Esp-Wrover-Kit mit dem Esp32 Chip ähnlich dem Uno Rev 3 in Unterabschnitt 2.1.2 ein Massenspeichercontroller und kein selbständiger Computer [9]. Die Performanz ist besser als der Uno Rev 3 von Arduino.

2.2 Netzwerke

In dieser Ausarbeitung werden drei besondere Arten von Netzwerken behandelt. Ein zentrales Netzwerk, ein dezentrales Netzwerk und ein verteiltes (distribuiertes) Netzwerk. In diesem Abschnitt werden die einzelnen Netzwerke mit deren Vor- und Nachteilen erläutert.

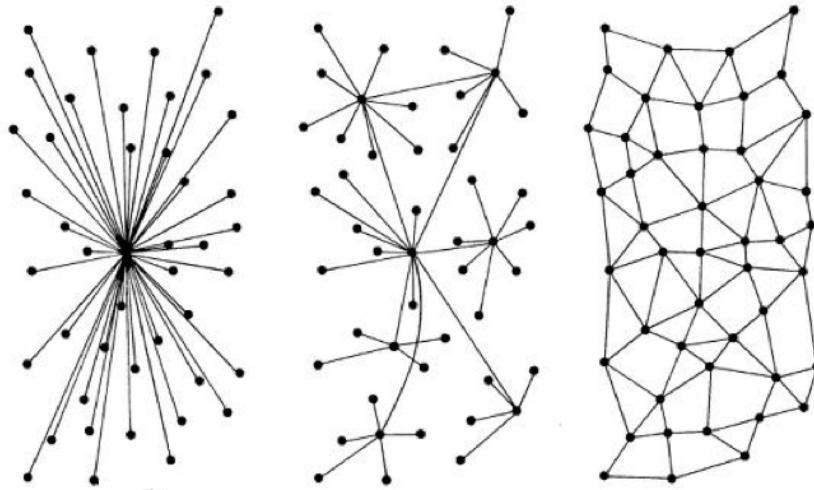


Abbildung 2.5: Links: Zentrales Netzwerk; Mitte: Dezentrales Netzwerk; Rechts: Verteiltes Netzwerk [13]

2.2.1 Zentrales Netzwerk

Bei einem zentralen Netzwerk gibt es einen „Single Point of Control“ zu dem die anderen Computer im Netz verbunden sind. Abbildung 2.5 Links zeigt solch ein Netzwerk.

Vorteil dieses Netzwerkes ist es, dass die Kommunikation koordiniert über einen Punkt ablaufen und es zu keiner Überschneidung mit anderen Servern kommt. Durch dieser Eigenschaft steigt die Kontrolle über die einzelnen Geräte im Netzwerk. Das zentrale Netzwerk gilt zudem als wartungsfreundlich wegen des simplen Hinzufügens weiterer Geräte als Knotenpunkte. Sollte einer der verbundenen Endgeräte im Netz ausfallen, hat das keine Auswirkungen auf das Gesamtnetzwerk.

Ein Kritikpunkt stellt die erhöhte Auslastung auf dem Server dar. Da jedes Gerät abhängig von einem Kontrollpunkt ist, müssen entweder viele Anfragen synchron bearbeitet werden oder eine Warteschlange wird hinzugefügt. Dies führt zu einer entsprechenden Auslastung und zu einer längeren Wartezeit, die mit jedem weiteren Gerät im Netz ansteigt. Schwerwiegender ist der Ausfall des Hauptkontrollpunktes, dadurch wird das Netz aufgelöst und die Geräte sind auf sich gestellt [14].

2.2.2 Dezentrales Netzwerk

Das dezentrale Netzwerk ist eine andere Form der Nachrichtenübertragung. Hier gibt es keinen zentralen Kontrollpunkt. Das Netzwerk besteht aus Bausteinen mehrerer verwaltender Knotenpunkte (Abbildung 2.5 Mitte). Das System ist jederzeit erweiterbar durch weitere Bausteine. Diese Bausteine müssen gewisse Bedingungen erfüllen, um im Netzwerk teilnehmen zu dürfen. Damit ist nicht verlangt, dass die Knoten identisch sind.

Ein Beispiel für solch ein Netzwerk wäre der internationale E-Mail Verkehr. Es sind mehrere E-Mail Provider vorhanden, die unterschiedliche E-Mail Clients nutzen können. Das Transaktionsprotokoll **SMTP** ist aber für jeden E-Mail Provider fest vorgegeben.

Der Vorteil hier ist, dass durch den Ausfall von Bausteinen, das Netzwerk dennoch bestehen bleibt. Die Auslastung ist ebenfalls gering. Die Verarbeitung von Daten wird in den einzelnen Bausteinen realisiert. Im Gegensatz zum zentralisierten Netzwerk wird die Auslastung unregelmäßig aufgeteilt.

Die Wartung der Bausteine ist im Vergleich sehr aufwendig. Jeder einzelne Baustein muss selbstständig gewartet werden. Weiterhin fällt die Performanz enorm, wenn zu viele Bausteine ausfallen [14].

2.2.3 Distribuiertes Netzwerk

Ein distribuiertes (verteiltes) Netzwerk ist ein dezentrales Netzwerk mit dem Unterschied, dass ein Prozess unter den einzelnen Geräten im Netzwerk koordiniert und nebenläufig bearbeitet wird. Das heißt, die Ausführung eines Prozesses wird mit allen Bausteinen im Netz verarbeitet. Information der Ausführungen und Ergebnisse werden mit den anderen Bausteinen im Netzwerk geteilt (siehe Abbildung 2.5 Rechts). Sinn und Zweck eines solchen Netzwerkes ist die sichere und gemeinsame Verwendung von Ressourcen.

Ein Beispiel hierfür ist das Banksystem. Nach der Ein- / oder Auszahlung von Bargeld müssen die anderen Automaten Information vom realen Kontostand haben.

Die Ein- / und Ausgliederung von Bausteinen ist simpel. Es können auch einzelne Bausteine ausfallen, ohne das Netzwerk aufzulösen. Weiterhin wird die Auslastung koordiniert und gleichmäßig an die Bausteine verteilt.

Nachteilig ist die aufwendige Kommunikation zwischen den einzelnen Bausteinen. Im Vergleich zu einem zentralen und dezentralen System ist die Kommunikation zwischen den Bausteinen notwendig. Zu gleich ist es anfällig vor falschen Daten. Wenn sich ein korrumpter Baustein im Netz befindet, ist es möglich Daten zu verfälschen und die restlichen Bausteine zu korrumpern [14] und [19].

2.3 IoT Betriebssysteme

IoT Geräte sind kleine eingebettete Systeme die für ihre entsprechende Tätigkeit sowohl Hardware-technisch als auch Software-technisch spezialisiert sind. Dieser Unterabschnitt benennt einzelne Betriebssysteme für IoT Produkte mit Vor- und Nachteilen und vergleicht diese miteinander. Wie in Abschnitt 2.1 erläutert, ist derzeit nur das Raspberry Pi in der Lage ein vollständiges Betriebssystem aufzuspielen. Die genannten Betriebssysteme richten sich deshalb nur auf das Raspberry Pi.

Mögliche Betriebssysteme für einen Mikrocontroller werden im Unterabschnitt 2.3.4 behandelt.

2.3.1 Windows 10 IoT

Windows 10 IoT ist die Umsetzung von Microsoft für eingebettete Systeme. Das Betriebssystem wurde am August 2015 für die Öffentlichkeit zugänglich gemacht. Es existieren verschiedene Versionen des Betriebssystems.

- **Windows 10 Core** ist das Standardbetriebssystem von Windows 10 IoT für eingebettete Systeme und ist kostenfrei für Entwickler zugänglich.
- **Windows 10 IoT Core Pro** beinhaltet zusätzlich noch Funktionen zum Kontrollieren und Verschieben von Updates.
- **Windows 10 IoT Enterprise** ermöglicht die Verwendung von Windows nativen und auch core Applikationen. Windows native Applikationen dienen zur Kommunikation mit dem Betriebssystemkern. Es gibt keine Startapplikation wie die Core Variante. Das Betriebssystem startet mit einer Shell Konsole.
- **Windows 10 IoT Mobile** ist ein Betriebssystem, welches auf Smartphones angepasst wurde. Es basiert auf das Windows 10 IoT Core Betriebssystem.
- **Windows 10 IoT Mobile Enterprise** basiert auf Windows 10 IoT Mobile. Es ermöglicht zusätzlich die Verwaltung von eingebetteten Systemen wie Barscanner und anderen Bedieneinheiten auf dem Smartphone.

Quelle [4].

Windows 10 IoT bietet mehrere Komfortfunktionen. Beispielsweise ist die Bereitstellung von Treibern für Display und Sound mit integriert. Weitere Treiber werden mit den monatlichen Updates bereitgestellt.

Die Kommunikation zwischen Desktop PC und IoT Gerät findet mit einer integrierten Weboberfläche statt. Auf der Weboberfläche ist es möglich Applikationen zu verwalten, Einstellungen zu bearbeiten, eine Fernwartung durchzuführen und die Systemsoftware zu installieren.

Die Installation des Betriebssystems findet auf dem Desktop statt. Das Betriebssystem wird auf eine Speicherplatte geflasht und in das IoT Gerät eingebunden. Nachdem starten des IoT Gerätes sind keine weiteren Einstellungen notwendig. Der Windows 10 Core Dashboard von Microsoft vereinfacht die Installation und lädt die gewünschte Version herunter. Kostenpflichtige Betriebssysteme erfordern einen manuellen Download.

Das Betriebssystem verfügt über eine REST-Schnittstelle. Es ist möglich das IoT Gerät mit Benutzername und Passwort extern auf einem anderen Gerät zu verwalten. Dazu ist es notwendig eine Anwendung zu entwickeln, der diese Schnittstellen anspricht.

Anwendungen können aus dem Microsoft Store verwendet werden. Es ist auch möglich weitere Anwendung extern zu aktualisieren. Die Installationen der Applikationen verlaufen entweder über die REST-Schnittstelle oder über die bereitgestellte Weboberfläche. Die Programmiersprache der Applikation kann dabei beliebig sein. Windows 10 IoT enthält kein Desktop. Es ist nicht gedacht Anwendungen unabhängig von einem PC zu installieren.

Das Betriebssystem wird mindestens zweimal monatlich mit Verbesserungen und Fehlerbehebungen versorgt. Die Geräte aktualisieren sich automatisch und startet die Startapplikation neu. Der Nutzer ist in der Lage unter den Einstellungen diese Funktion zu deaktivieren. Beim erstmaligen Starten des Betriebssystems erscheint auf dem Display des Gerätes die Hardwareinformationen zum Gerät.

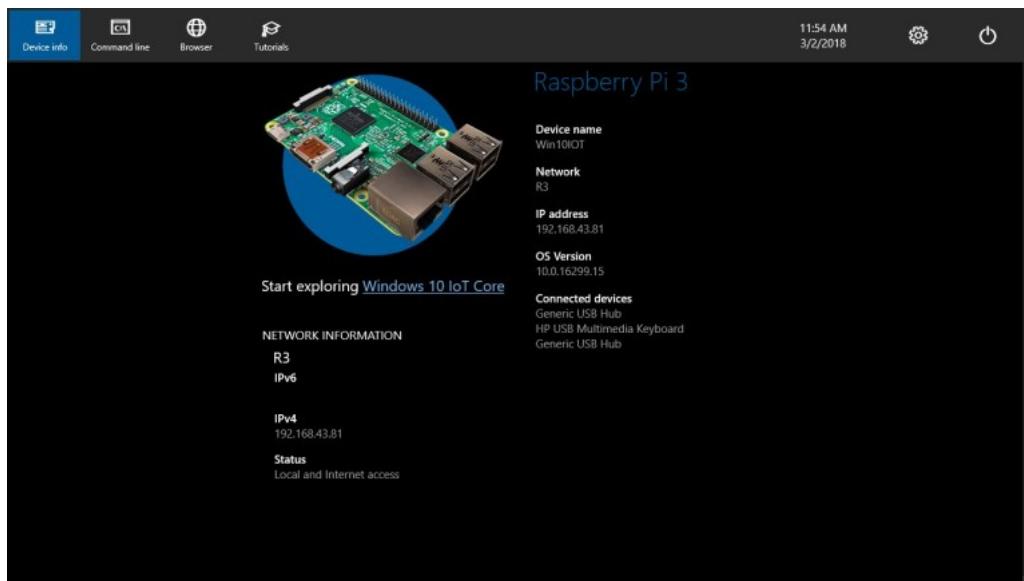


Abbildung 2.6: Startbildschirm von Windows 10 IoT [Quelle: Selbst erstellt]

Auf dem Interface sind vier verschiedene Tabs vorhanden.

- Mit der Device Info Tab in Abbildung 2.6 werden Informationen zu verbundenen Netzwerken und zum Gerät selbst dargestellt.

- Auf der Command Line Tab stellt das Betriebssystem eine Shell zur Verfügung, der der Desktop Version identisch ist.
- Der Browser Tab ermöglicht das Surfen im Internet mit dem Microsoft Edge Browser
- Der Tutorials Tab stellt Anleitungen für Anfänger bereit. Dort können bereits bereitgestellte Beispielprojekte gestartet werden.

Außerhalb der Eingabe mit dem Touchdisplay ist es möglich eine Remote Verbindung mit dem Raspberry Pi zu erstellen und mit der Tastatur und Maus des Desktop PCs das Gerät zu bedienen.

Öffnet man die IP-Adresse des Gerätes im Browser erscheint eine Weboberfläche, um notwendige Einstellungen, sowie Applikationen zu installieren (siehe Abbildung 2.7).

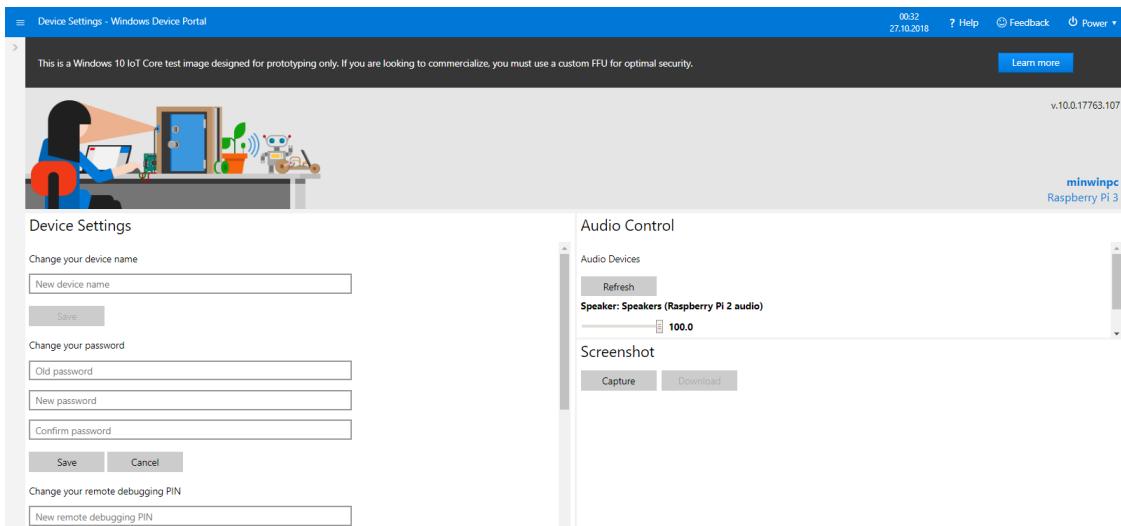


Abbildung 2.7: Weboberfläche der Windows 10 IoT Seite [Quelle: Selbst erstellt]

Auf der Seite ist es möglich Geräteeinstellungen, installierte Applikationen, laufende Prozesse, Netzwerkverbindungen und Updates zu verwalten. Zusätzlich befindet sich auf der Seite ein Tutorial für die Nutzung der Weboberfläche des Raspberry Pi 3 mit dem Betriebssystem Windows 10 IoT.

Jeder der Funktionen auf der Weboberfläche sind ebenfalls durch die integrierte REST-Schnittstelle zu erreichen.

Microsoft empfiehlt die Programmierung von Applikationen mit dem neuesten Visual Studio zu entwickeln (2017). Mögliche Programmiersprachen sind dabei C++, C#, Python und Java. Zu beachten ist, dass die zu installierende Applikation eine .appx oder eine .appxbld Datei sein muss, andere Formate wie beispielsweise .apk werden nicht akzeptiert [4].

2.3.2 Raspbian Linux-Distribution

Raspbian ist das Standardbetriebssystem vom Raspberry Pi. Im Vergleich zu den übrigen Betriebssystemen ist Raspbian das älteste und bekannteste IoT-Betriebssystem auf dem Markt. Es ist eine auf Debian basierende Linux-Distribution angepasst auf das Raspberry Pi.

Das Betriebssystem besitzt eine Desktop-Umgebung basieren auf PIXEL (Pi Improved X-Window Environment Lightweight). PIXEL basiert auf der open source Desktop-Umgebung LXDE. Die Desktop-Umgebung hat einen geringen Speicher- und Energieverbrauch. Im Vergleich zu den anderen Betriebssystemen kann das Betriebssystem aufgrund der Desktop-Umgebung ohne einen externen PC konfiguriert werden [21]. Abbildung 2.8 stellt den Desktop dar.

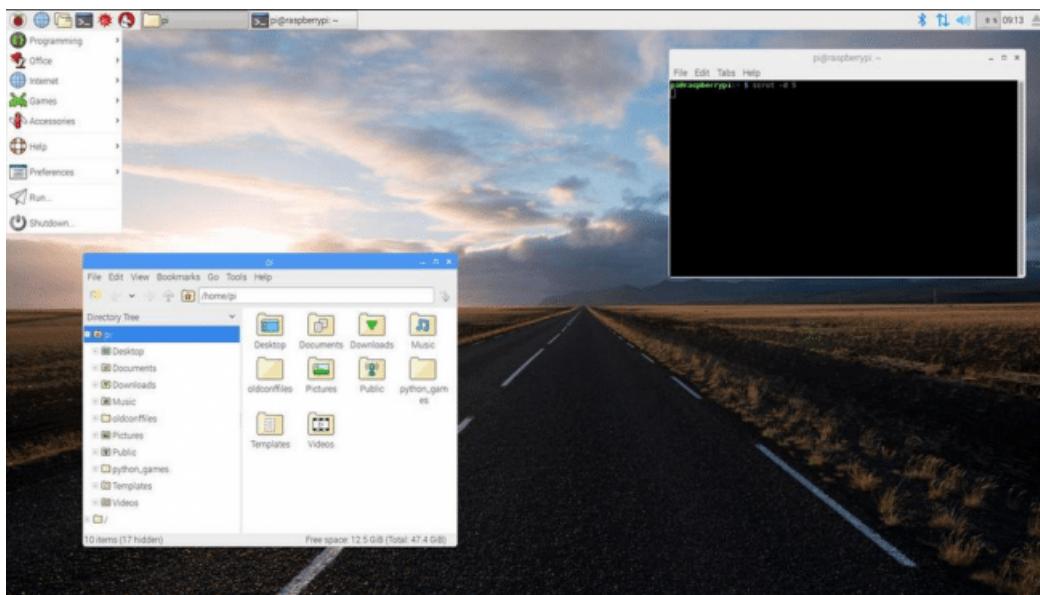


Abbildung 2.8: Desktop vom Raspbian [Quelle: Selbst erstellt]

Die Kommunikation zwischen Desktop PC und IoT Gerät findet mit einer Netzwerkverbindung statt. Bei dem Netzwerk kann es sich um ein Ethernet oder einen WIFI Anschluss handeln. Eine Weboberfläche die durch eine interne IP aufrufbar ist, ist nicht vorhanden. Die Installation des Betriebssystems findet wie bei Windows 10 IoT auf dem Desktop statt. Das Betriebssystem wird auf eine Speicherkarte geflasht und in das IoT Gerät eingebunden. Nachdem Starten des IoT Gerätes sind keine weiteren Einstellungen notwendig. Ein Tool zum Flashen wird von den Entwicklern nicht bereitgestellt.

Raspbian besitzt seit 2012 ein Pi-Store, welches notwendige und optionale Anwendungen zum Herunterladen anbietet und diese installiert. Eine weitere Art der Installation bietet das bereitgestellte Paketverwaltungssystem Application Package Tool (APT) an. Mit dem APT Tool können Anwendungen mit einem Konsolen-Befehl heruntergeladen und instal-

liert werden. Raspbian erzwingt dabei keine spezifische Programmiersprache. Es ist möglich Anwendungen direkt auf dem Gerät selbst zu entwickeln [18].

Das Betriebssystem wird unregelmäßig mit Verbesserungen und Fehlerbehebungen versorgt. Das Update-Intervall bewegt sich je nach Größe des Updates zwischen einem bis vier Monate. Die Aktualisierung der Betriebssystemsoftware geschieht halbautomatisch. Ein Update muss auf dem Gerät bestätigt werden.

Jede Anwendung die auf einem PC mit Linux läuft, kann theoretisch auf den Raspberry Pi laufen. Einzige Einschränkung sind die Hardwarevoraussetzungen der einzelnen Anwendungen. Raspbian ist ein vollständiges Betriebssystem für das Raspberry Pi. Es ist nicht auf das System IoT spezialisiert. Ein Vergleich mit den anderen Betriebssystemen ist deshalb schwer.

Die Programmierung einer IoT Anwendung auf dem Betriebssystem ist möglich, wegen der Universalität geht die Performance dennoch für andere nicht notwendige Prozesse verloren.

2.3.3 Android Things

Android Things (früher Google Brillo) ist ein weiteres auf IoT spezialisiertes Betriebssystem von der Google LLC (Limited Liability Company). Die Veröffentlichung der ersten Release-Version war am Mai 2018. Im Gegensatz zu Windows 10 IoT existiert nur eine Version des Betriebssystems. Diese Version ist kostenlos für jeden zugänglich.

Da das Betriebssystem noch jung ist, hat es nicht viele Komfortfunktionen. Einige Standardtreiber wie GPS und Display sind bereits enthalten. Zusätzliche Treiber wie für die Hardwareerweiterung PiFace 2 sind noch in der Entwicklung.

Eine Kommunikation zwischen Desktop und IoT Gerät aufzubauen ist umständlicher als auf dem Betriebssystem Windows 10 IoT. Für eine Verbindung ist es notwendig eine Android Debug Bridge (ADB) manuell aufzubauen. Die Bereitstellung und Installation kann dann durch Googles bereitgestellte Programmierumgebung Android Studio durchgeführt werden.

Die Installation des Betriebssystems läuft identisch wie bei Windows 10 IoT und Raspbian auf einem Desktop PC ab. Die Speicherplatte wird mit dem Android Things geflasht und in das IoT Gerät eingebaut. Ab hier sind keine weiteren Einstellungen für den Start des Betriebssystems notwendig. Das Android Things Setup von Google vereinfacht die Installation des Betriebssystems, indem es die Netzwerk Einstellungen konfiguriert. Nach der Eingabe von Netzwerknamen und Passwort wird automatisch nach jedem Start des Gerätes eine Internetverbindung mit dem angegebenen Netzwerk eingegangen.

Es gibt derzeit kein offiziellen Store für die Installation von Anwendungen auf dem Gerät. Auch die Verwaltung von Anwendungen ist derzeit nicht möglich. Multiple Anwendungen müssen in APK Format auf der Seite <http://partner.android.com/things/console> für die gemeinsame Installation aufgelistet werden. Danach ist es möglich die Image-Datei von

der Webseite herunterzuladen und die Anwendungen auf die Speicherkarte des IoT Gerätes per ADB zu installieren.

Das Betriebssystem wird monatlich mit Verbesserungen und Fehlerbehebungen versorgt. Die Aktualisierung der Betriebssystemsoftware geschieht nicht automatisch und muss manuell installiert werden.

Beim erstmaligen Starten des Betriebssystems erscheint auf dem Display des Gerätes die Hardwareinformationen zum Gerät. Weitere Optionen sind derzeit nicht vorhanden. Die Oberfläche des Betriebssystems besitzt keinen Desktop. Für die Verwaltung des Gerätes ist ein Desktop PC zwingend notwendig.

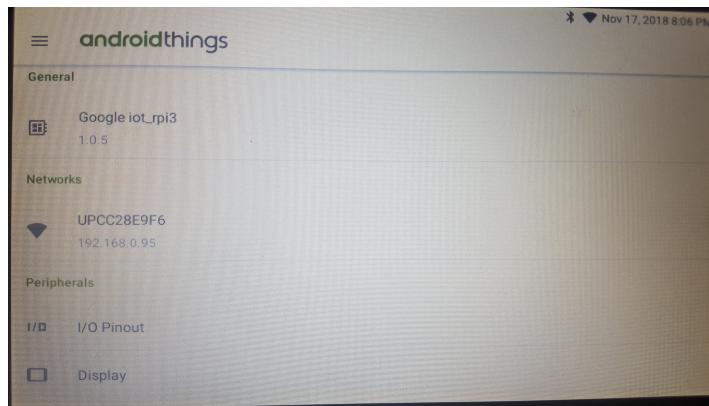


Abbildung 2.9: Startbildschirm von Android Things [Quelle: Selbst erstellt]

Die Abbildung 2.9 zeigte den Startbildschirm von Android Things. Der Startbildschirm zeigt das aktuelle Betriebssystem, die derzeitigen Netzwerkverbindungen und die vorhandenen peripheren Eigenschaften des Betriebssystems.

Das Betriebssystem hat keine Weboberfläche und keine REST-API. Verbindungen zum IoT-Gerät sind ausschließlich mit einer ADB Kommunikation möglich.

Die Entwicklung für Android Things benötigt Android Studio. Nur mit Android Studio ist es derzeit möglich Anwendungen komfortabel ohne ADB Konsole Anwendungen bereitzustellen. Die einzige mögliche Programmiersprache ist hier Java für Android. Zu beachten ist, dass die zu installierende Applikation eine .apk Datei sein muss, andere Formate werden nicht akzeptiert. Weitere Informationen zu dem Betriebssystem sind unter der Entwicklerseite [20] zu finden.

2.3.4 Weitere IoT Betriebssysteme

Die Anzahl der Betriebssysteme für IoT Geräte auf dem Markt sind vielfältig und für verschiedene Nutzungsmöglichkeiten angepasst. Die unten genannten drei Betriebssysteme sind optimiert für Mikrocontroller von 8 bis 32 Bit.

- **RIOT.** RIOT wurde von der Freien Universität Berlin, Institut national de recherche en informatique et en automatique (INRIA) und der Hochschule für angewandte Wirtschaft in Hamburg entwickelt. Der Source Code ist derzeit öffentlich zugänglich und kann von jedem Entwickler verwendet und erweitert werden.
Das Betriebssystem unterstützt Mikrocontroller mit 8, 16 und 32 Bit. Es unterstützt Multithreading für ein schnelles Bearbeiten von Prozessen und ist im Vergleich zum herkömmlichen Linux energieeffizienter. Das Betriebssystem hat eine API für den Zugriff von außen [1].
- **Contiki.** Contiki wurde entwickelt von Adams Dunkeln. Es ist ein freies, internet-fähiges Betriebssystem für 8 Bit Mikrocontroller. Es hat eine minimale grafische Oberfläche siehe Abbildung 2.10. Besonders sind hier die Protothreads. Protothreads benötigen im Gegenteil zu Threads keinen Stapspeicher. Bei einem Kontextwechsel müssen die Protothreads global oder lokal statisch definiert werden. Dies spart Speicher und ist effizienter.
Eine weitere Besonderheit ist die Portabilität des Betriebssystems. Beispielsweise ist es möglich auf älteren Spielekonsolen das Betriebssystem zu installieren [8].

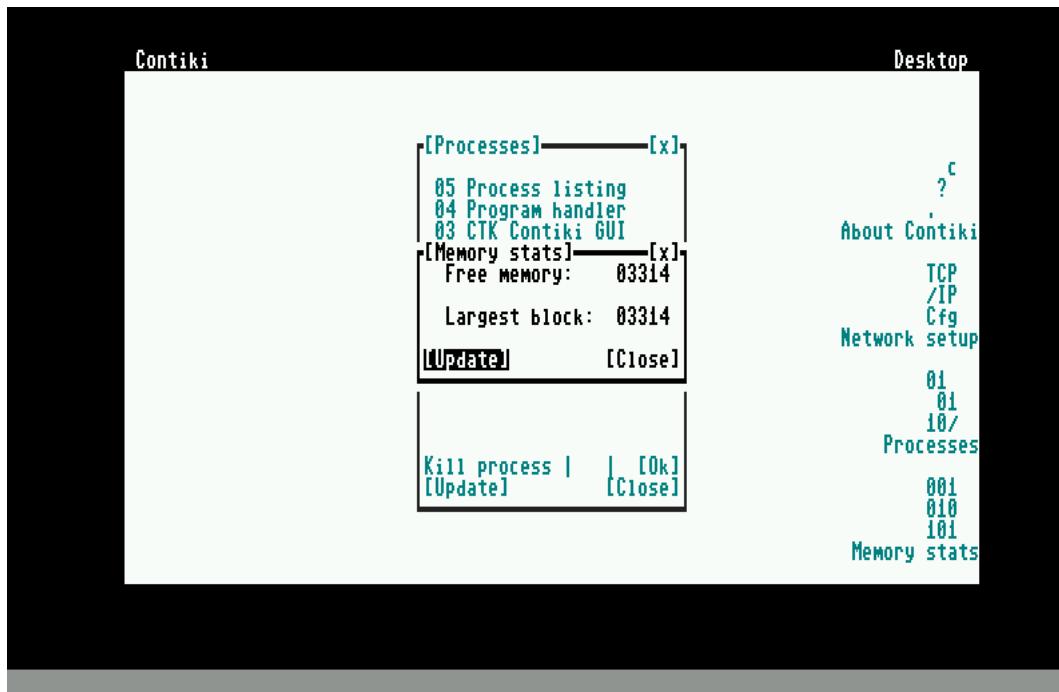


Abbildung 2.10: Startbildschirm von Contiki [8].

- **Zephyr.** Zephyr ist ein Echtzeitbetriebssystem von der Linux Foundation für IoT Geräte. Das Betriebssystem ist optimiert für Geräte mit kleinem Speicher und fester Hardware. Es kommt mit einem minimalen Arbeitsspeicher von 8 KByte und einen maximalen von 512 KByte aus. Das Betriebssystem ist stark anpassbar auf

die Tätigkeit, die es verrichten soll. Eine mögliche Anwendung ist die Steuerung von Beleuchtungs- und Heizungssysteme. Das Betriebssystem unterstützt Mikrocontroller mit 8, 16 und 32 Bit. Es beherrscht Multithreading für ein schnelles Bearbeiten von Prozessen und ist im Vergleich zum herkömmlichen Linux effizienter [10].

2.4 SOAP und REST

REST (**R**Epresentational **S**tate Transfer Architektur) und SOAP (**S**imple **O**bject **A**ccess **P**rotocol) sind beides Web API Services die sich von Ihrer Technik grundlegend unterscheiden. In diesem Abschnitt werden die beiden Techniken einzeln erläutert und zum Schluss miteinander verglichen.

2.4.1 SOAP

SOAP ist die ältere Variante der Web API Services. Es handelt sich um ein Protokoll, welches die Kommunikation zwischen verschiedenen Plattformen und Programmiersprachen ermöglichen soll.

Es wird eine WSDL (Web Services Deskription Language) zur Konfiguration und zum Senden, sowie zum Empfangen von Nachrichten benötigt. Die WSDL Konfigurationsdatei beinhaltet die Instruktionen aller möglichen Funktionen (und Port-freigaben) zum Senden und Empfangen von Daten [26].

Die Konfigurationsdatei enthält die Elemente:

- <types> Datentypen die für den Web-Service erstellt werden
- <message> Die Variable mit Typ, die für die jeweilige Operation benötigt bzw. ausgegeben werden.
- <portType> Definiert die ausführbaren Methoden zum Antworten und Erfragen von Daten.
- <binding> Definiert die Ports für die einzelnen Operationen.

```

<message name="getPriceRequest">
    <part name="item" type="xs:string"/>
</message>

<message name="getPriceResponse">
    <part name="price" type="xs:string"/>
</message>

<portType name="glossaryTerms">
```

```

<operation name="getPrice">
    <input message="getPriceRequest"/>
    <output message="getPriceResponse"/>
</operation>
</portType>

<binding type="glossaryTerms" name="example1">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <operation>
        <soap:operation
            soapAction="http://example.de/getPrice"/>
        <input><soap:body use="literal"/></input>
        <output><soap:body use="literal"/></output>
    </operation>
</binding>
```

Quellcode 2.1: Beispiel einer WSDL Konfiguration auf dem Server

Das Beispiel 2.1 zeigt eine Beispielkonfiguration. Die `<message>` Elemente definieren zwei Methoden `getOperationRequest` und `getOperationResponse` mit jeweils einer String Variable. Das `<portType>` Element definiert `getOperationRequest` als Eingabe- und `getOperationResponse` als Ausgabemethode. Das `<binding>` Element bindet beide Methoden an einen Port (hier: `.../getOperation`).

Zur Kommunikation werden zusätzlich WSDL Dateien benötigt. Die Dateien für das Empfangen und Senden sind ähnlich aufgebaut. Die Datei besitzt ein Envelope-Element, welches die XML Datei als SOAP Nachricht kennzeichnet. Ein Header-Element, das Zusatzdaten (Metadaten) beinhaltet. Beispielsweise einen Authentifizierungstoken zum Signieren der Nachricht. Ein Body-Element, welches die Aufruf- und Antwortinformation enthält und ein Fault-Element, das Fehler- und Statusinformationen angibt.

Das Envelope-Element umschließt das Header-, Body- und das Fault-Element. Es benötigt als Attribute zusätzlich die Angabe einer Kodierungsart, da keine Standardkodierung definiert ist.

Die Kommunikation verläuft über die HTTP-Anfragen (GET, POST, PUT, DELETE, ...). Grund ist die vorhandene Unterstützung von allen verfügbaren Webbrowsersn und Servern [26]. Quellcode 2.2 und 2.3 zeigen einen möglichen Kommunikationsverlauf zwischen Server und Client.

```

POST /InStock HTTP/1.1
Host: www.example.de
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
  soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

  <soap:Body xmlns:m="http://www.example.de/getPrice">
    <m:GetPriceRequest>
      <m:Item>Raspberry Pi 3</m:Item>
    </m:GetPriceRequest>
  </soap:Body>

</soap:Envelope>
```

Quellcode 2.2: Beispiel einer WSDL Anfrage an den Server mit der Konfiguration von Quellcode 2.1

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
  soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

  <soap:Body xmlns:m="http://www.example.de/getPrice">
    <m:GetPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetPriceResponse>
  </soap:Body>

</soap:Envelope>
```

Quellcode 2.3: Antwort der WSDL Anfrage von Quellcode 2.2 an den Client

2.4.2 REST

Während es sich bei SOAP um ein Protokoll handelt, löst REST die Kommunikation zwischen Server und Client mit einem Architekturansatz. Für die Verwendung des REST wird die Einhaltung von sechs Prinzipien empfohlen [28].

- **Einheitliche API.** Die API wird gesondert implementiert. Das steigert die Visibilität der Operationen auf Kosten der Effizienz. Der Aufbau der API ähnelt einer einfachen Aufzählung der Operationen. Jeder Entwickler, der die API einer Anwendung versteht, sollte in der Lage sein alle anderen API zu verstehen.
- **Client-Server-Modell.** Für die Skalierung sollen Serverstrukturen von den Clientstrukturen getrennt werden. Dadurch können Clients auf verschiedenen Systemen implementiert werden.
- **Zustandslosigkeit.** Die Kommunikation zwischen Server und Client ist Zustandslos. Der Server speichert keine Informationen über den Client. Jegliche Art des Kommunikationsaustausches soll vom Client gespeichert werden. Dieses Prinzip erhöht die Visibilität, Zuverlässigkeit und Skalierbarkeit der Schnittstelle auf Kosten der Datenstabilität (Konsistenz).
- **Caching.** Clients sind in der Lage Antworten vom Server zwischenspeichern. Die Informationen müssen dafür auf **cacheable** gesetzt werden. Diese Funktion soll die Effizienz steigern. Das Risiko, das dadurch die Aktualität von Informationen vernachlässigt werden, besteht.
- **Mehrschichtiges hierarchischen System** (Layered System). API, Datenbank und Authentifizierung können auf unterschiedlichen Servern ablaufen. Der Client erkennt die Strukturen im Hintergrund nicht.
- **Code on demand** (optional). Es steht den Entwickler frei ausführbaren Code an den Client als Antwort zu senden. Der Client kann sich dadurch einige unnötige Funktionen sparen. Da der Server mehr Informationen als nötig zurückgibt.

Das Ziel von REST ist es Ressourcen durch Maschine-zu-Maschine Interaktion zu versenden. Bei einer Ressource kann es sich um jegliche Art einer Datei handeln. Text, Bilder, Objekte und weiteres sind möglich.

Die Ressourcen werden durch die festgelegte URI für den Client zugänglich gemacht. Die URI kann beliebig sein und unabhängig vom Methodennamen. Der Aufruf der URI geschieht mithilfe der HTTP Methoden (GET, POST, PUT, DELETE,...). Die Ressourcen können durch den Aufruf gelesen und verändert werden [28].

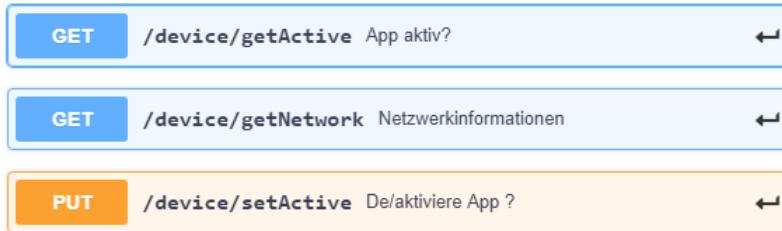


Abbildung 2.11: Beispiel URI für eine REST Abfrage

```
http://127.0.0.1:8000/device/getActive
```

```
HTTP/1.1 200 OK
```

```
...
```

```
{
  "active": true
}
```

Quellcode 2.4: Antwort einer REST Anfrage an den Client

Abbildung 2.11 zeigt mögliche HTTP Anfragen an den Server und Quellcode 2.4 eine mögliche Antwort für die oberste REST Anfrage.

2.4.3 Vergleich zwischen REST and SOAP

Die einzelnen Vor- und Nachteile der einzelnen Web-Services werden in diesem Unterabschnitt kurz aufgelistet und miteinander verglichen.

Generell ist der Vergleich sehr schwer, da REST ein Architekturprinzip ist und SOAP ein Protokollsysteem. Beide Systeme können gemeinsam genutzt werden, um ein bestmögliches Ergebnis zu erhalten. Wichtige Daten können beispielsweise durch SOAP und weniger sicherheitsrelevante durch REST übermittelt werden. Sicherheit und Performanz von beiden Services könnten so optimal genutzt werden.

Tabelle 2.1 vergleicht verschiedene Eigenschaften zwischen den Web-Services miteinander.

Eigenschaft	SOAP	REST
Komplexität	Komplex. Die Implementierung der Anwendung ist in Bezug zu REST sehr komplex. Variablen, Parameter, Methoden und Ports müssen einzeln deklariert werden. Die Verständlichkeit der API wird dadurch schwieriger.	Simple. API sind normalerweise sehr einheitlich und einfach zu schreiben. Jeder Entwickler sollte in der Lage sein eine REST Schnittstelle zu verstehen
Implementierung	Nicht anwendbar. SOAP ist ein Protokoll und kein Architekturkonzept wie REST.	Universal. Durch das Client-Server Modell ist die Implementierung der einzelnen Clients unabhängig vom Server. Das heißt es können verschiedene Programmiersprachen und -architekturen verwendet werden.
Sicherheit	Hohe Sicherheit. Jede Nachricht kann beim Senden und Empfangen der Nachrichten mit einem Authentifizierungstoken versehen werden.	Wenig Sicherheit. Wegen der Zustandslosigkeit wird die Authentifizierung des Clients nicht geprüft. Die Implementierung der Sicherheit muss außerhalb der Schnittstelle stattfinden.
Performanz	Wenig Performanz. Auf Kosten der Performanz wird üblicherweise ein Authentifizierungstoken für die Kommunikation verwendet. Bei Verwendung ist SOAP nicht Zustandslos.	Hohe Performanz und Skalierbarkeit. Durch die Zustandslosigkeit der REST-API werden keine Daten am Server gespeichert. Jede Anfrage wird separat beantwortet.
Nachrichtenformat	Fest. Der Austausch von Nachrichten ist nur in XML.	Universal. Nachrichten können in verschiedenen Formaten gesendet werden. (JSON, YAML, HTML, ...)

Fehlerbehandlung	Vorhanden. Informationen von Prozessfehlern sind in der Nachricht optional eingetragen.	Nicht vorhanden. Die Informationen über Fehler übernimmt der Webbrows-
Anwendungsprotokolle	Universal. Jedes Anwendungsprotokoll wird unterstützt	Nur HTTP. REST ist auf die Verwendung von HTTP-Browsern spezialisiert.
Visibilität	Normal. Der Aufbau des Systems ist abhängig von der Implementierung des Entwicklers. Vorgaben von SOAP gibt es nicht.	Offen. Durch das layered System und der einheitlichen API sind die Vorgänge des Systems leicht zu erkennen.

Tabelle 2.1: Vergleich zwischen REST und SOAP. [28] und [26].

2.5 Sprachen für die Modellierung grafischer Benutzeroberflächen

Die Implementierung von IoT-Applikationen ist für einige IoT Betriebssysteme eingeschränkt. Während IoT-Betriebssysteme wie die Linux-Distribution von Raspberry die Wahl der Implementierungssprache freistellen, legen andere Betriebssysteme diese fest. Android Things verlangt die Nutzung von Java und XML als Implementierungssprache. Windows 10 IoT C# oder C++ mit XAML.

Hier werden die grafischen Implementierungssprachen, die für die IoT-Betriebssysteme in Abschnitt 2.3 verwendet werden, behandelt.

2.5.1 XAML

Extensible Application Markup Language (XAML) ist eine XML basierte Sprache zur Visualisierung von grafischen Oberfläche mit Windows Presentation Foundations (WPF) Anwendungen von Microsoft.

Mit dem Zusammenspiel von CodeBehind-Dateien soll die Zusammenarbeit zwischen der Modellierung und der Implementierung von Funktionen vereinfacht werden. CodeBehind-Dateien stehen für die direkten Verbindungen zwischen Anwendungsprogrammierung und der grafischen Modellierungsdatei. Jede Modellierungsdatei (.xaml) besitzt eine gleichnamige Implementierungsdatei in der zuvor gesetzten Sprache C++ (.xaml.cpp) oder C# (.xaml.cs). Die Objekte, die in der Modellierungsdatei gesetzt sind, können vom Entwick-

ler für die Implementierung der Funktionen abgerufen werden. Für das Abrufen ist eine Bezeichnung des Objekts notwendig. Die Bezeichnung wird mit dem Attribut Name gesetzt [7].

Quellcode 2.5 und 2.6 zeigen das Zusammenspiel zwischen einer Modellierungsdatei und einer zugehörigen Funktionsimplementierung. Abbildung 2.12 zeigt das Ergebnis des Zusammenspiels.

```
<Window x:Class="WpfApp1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        Title="MainWindow" Height="450" Width="800">

    <Grid>
        <!-- Place new controls here -->
        <Label Content="Welcome to Xaml!" 
              VerticalAlignment="Center"
              HorizontalAlignment="Center"/>
    </Grid>

</Window>
```

Quellcode 2.5: Beispiel einer einfachen Xaml Anwendung (Xaml:Source Code)

```
using System.Windows;

namespace WpfApp1
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

Quellcode 2.6: Beispiel einer einfachen Xaml Anwendung (C# Source Code)

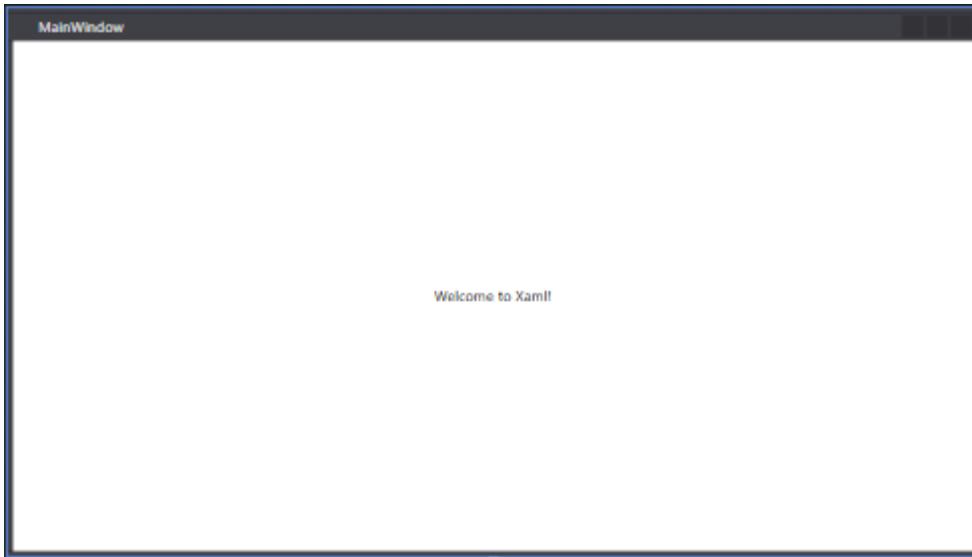


Abbildung 2.12: Visualisierung des Beispiels von 2.5 und 2.6

Der Window-Tag kreiert ein Fenster mit den notwendigen Konfigurationen. Die Attribute xmlns und xmlns:x definieren vorgegebenen Schemen die für die Modellierung verwendet werden.

Das xmlns:x bindet das gesetzte Schema (hier: 'http://schemas.microsoft.com/winfx/2006/xaml') an den Buchstaben x. Der Aufruf eines Schemen Tags/Attributs beginnt mit dem entsprechenden Buchstaben gefolgt von einem Doppelpunkt (Hier: x:Class).

2.5.2 XML für Android Things

Das Betriebssystem Android Things nutzt XML für die Visualisierung der Fensteroberfläche. Die Visualisierung der Oberfläche soll hauptsächlich mit dem Modellierungstool in Android Studio bewerkstelligt werden.

Das Interface einer Android Applikation ist hierarchisch aufgebaut. An oberster Stelle steht ein Viewgroup-Container welches andere Viewgroup-Container und View-Objekte beinhalten kann. View-Objekte sind Objekte die auf dem Fenster vom Nutzer gesehen werden wie ein Button oder ein Label.

Ähnlich wie XAML (Unterabschnitt 2.5.1) enthält jede XML eine entsprechende CodeBehind-Datei. Die CodeBehind-Datei ist eine Java Datei die durch das Attribut **tools:context=„Name“** in der XML verbunden wird [20].

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```

xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".MainActivity">

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:id="@+id/textview_id"
    android:visibility="visible"/>

</android.support.constraint.ConstraintLayout>

```

Quellcode 2.7: Beispiel einer einfachen XML Anwendung für Android (XML:Source Code)

```

package com.example.workstation_tu.myapplication;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

Quellcode 2.8: Beispiel einer einfachen XML Anwendung für Android (Java Source Code)

Die XML Datei ähnelt stark der XAML Syntax in Unterabschnitt 2.5.1. Die XML 2.7 mit der CodeBehind-Datei 2.7 hat die einfache Funktion die Textnachricht **Hello World** einzublenden. Die ersten Zeilen aus der XML verbindet die gewählten Schemata an eine definierte Variable. An dieser Stelle wird auch die CodeBehind-Datei definiert. Danach folgt die Erstellung der Textnachricht mit dem <TextView> Tag.

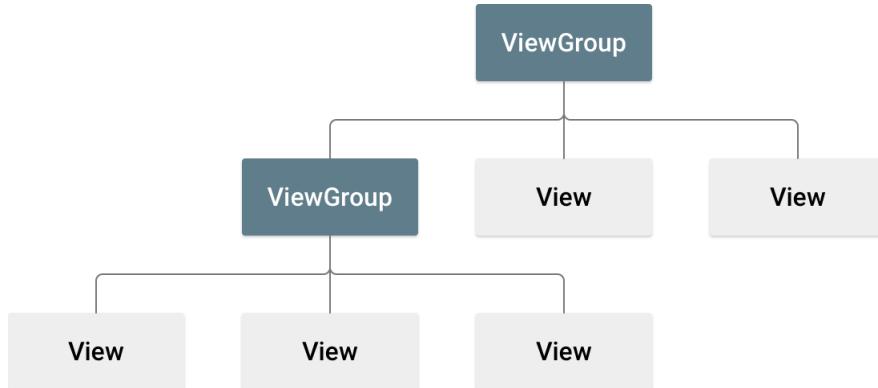


Abbildung 2.13: Visualisierung eines möglichen hierarchischen Aufbaus [20])

Die Java-Datei 2.8 schafft hier nur ein Objekt der Visualisierung der XML und gibt sie aus. Das Ergebnis der beiden Dateien ist in Abbildung 2.14 zu erkennen.

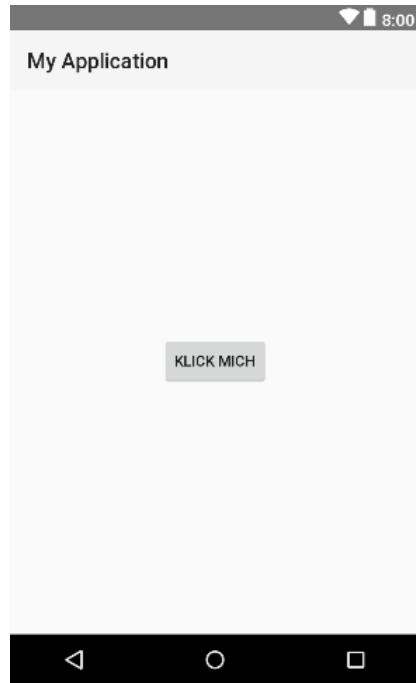


Abbildung 2.14: Visualisierung des Beispiels von 2.7 und 2.8)

2.6 Programmierumgebung

In diesem Abschnitt werden die Umgebungen, die in den Abschnitten 2.3 und 2.5 angesprochen worden sind, näher erläutert. Bibliotheken werden in Ihrer Funktion und Besonderheit näher beschrieben.

2.6.1 C#

C# ist eine objektorientierte und typsichere Sprache im Auftrag von Microsoft (Visual C#). Es ist im Jahre 2001 entstanden. Die Sprache wurde auf die Microsoft .NET Softwareplattform angelehnt, entwickelt und für diese optimiert. Später wurde es durch das Tochterunternehmen Xamarin plattformunabhängig weiterentwickelt [3].

Im nachfolgenden werden die Funktionen der impliziten Typisierung und des asynchronen Aufrufs veranschaulicht und näher gedeutet. Bei der Beschreibung wird die Kenntnis der Programmiersprache Java vorausgesetzt und darauf aufgebaut. Die folgenden Eigenschaften unterscheiden sich von der Programmiersprache Java.

Eine wichtige Eigenschaft von C# ist die Typsicherheit. Typsicherheit heißt, dass Datentypen spätestens bei der Laufzeit ausgewertet werden. Dennoch ist es möglich Variablen implizit zu setzen (siehe 2.9). Dabei wird der Typ nicht vom Entwickler selbst, sondern vom Compiler festgelegt.

```
var i = 10; // Implizite Typisierung
int i = 10; // Explizite Typisierung
```

Quellcode 2.9: Implizite und Explizite Typisierung in C#

Eine andere Änderung ist die Ausführung von Asynchronen Methoden. Durch das Keyword **async** werden Methoden als asynchron markiert. Der Compiler erwartet einen asynchronen Aufruf in der Methode und gibt ein Task zurück. Bei dem Rückgabewert einer asynchronen Methode ist zudem die Typisierung des Tasks notwendig, sowie ein **await** des Methodenaufrufs. Bei einem fehlenden **await** wird die Methode trotz asynchroner Markierung synchron aufgerufen. Die asynchrone Ausführung muss bei der hierarchischen Folge komplett eingehalten werden. Ansonsten besteht die Gefahr einer Prozessblockierung und die asynchrone Ausführung wird ausgehebelt. Der Quellcode 2.10 soll zur Veranschaulichung für die Implementierung einer asynchronen Methode dienen.

```
private static async Task Task_RunAsync()
{
    var restRouteHandler = new RestRouteHandler();
    restRouteHandler.RegisterController<RestController>();

    var configuration = new HttpServerConfiguration()
        .ListenOnPort(8081)
        .RegisterRoute("api", restRouteHandler)
        .EnableCors();
```

```

    var httpServer = new HttpServer(configuration);
    await httpServer.StartServerAsync();
}

```

Quellcode 2.10: Asynchrone Starten eines Servers in C#

Im Bereich IoT ist die asynchrone Ausführung sehr wichtig. Die Geräte kommunizieren ständig miteinander und dürfen bei Fehlern oder Verzögerungen nicht aktiv auf eine Antwort warten. Das Kommunikationsnetzwerk (siehe Abschnitt 2.2) könnte sonst zusammenbrechen.

2.6.2 Microsoft Visual Studio 2017

Bei der C# Programmierung ist es notwendig eine andere Programmierumgebung zu wählen. Da C# wie in Unterabschnitt 2.6.1 zunächst auf die .NET Softwareplattform spezialisierte Programmiersprache ist, wird in diesem Unterabschnitt einige Informationen zu Microsoft Visual Studio 2017 vermittelt. Die Programmierumgebungen für Java werden als Vorwissen vorausgesetzt.

Microsoft Visual Studio 2017 ist eine Programmierumgebung entwickelt von Microsoft für das .Net Framework. Es unterstützt aktuell die Sprachen C#, C++, F#, SQL, TypeScript, Python, HTML, JavaScript und CSS. Mit diesen Programmiersprachen ermöglicht Visual Studio die Entwicklung von dynamischen Webseiten, sowie von mobilen Anwendungen. Jede erstellte Anwendung ist für das unternehmenseigene Betriebssystem Windows optimiert. Die unterstützten Prozessoren sind x86, x64 und ARM. Die Programmierumgebung kann aus syntaktisch korrekten Programmiercode selbstständig die Anwendungen angepasst auf die Prozessoren generieren.

Die Programmierumgebung hat diverse Komfortfunktion bei der Entwicklung von Anwendungen. Die syntaktische Prüfung des Quellcodes durch das eigenständige Hilfsmittel IntelliSense, welches das Ausblenden von Codeblöcken, das Hervorheben von Schlüsselwörter und automatische Ergänzung von Funktionsmethoden ermöglicht, gehören zum Standard. Eine größere Komfortfunktion stellt die eingebaute Echtbilddarstellung oder auch der „What You See Is What You Get“ (kurz WYSIWYG) Editor dar. Der Editor visualisiert die grafische Konstruktion von Xaml Dateien (siehe 2.5.1) während der Bearbeitung [5]. Die Einstellungen des Editors ermöglichen es mögliche Darstellungen auf die Entwicklung anzupassen. Attribute und implementierte Funktionen können ohne Programmierwissen an Ereignissen gebunden werden. Diese Eigenschaft ermöglicht eine Trennung der Programmierung zwischen Model, View und Control (MVC).

Im Laufe der Entwicklung der Programmierumgebung haben sich Gemeinschaften gebildet, die ihre Implementierung anderen Programmierern zur Verfügung stellen, um ihnen einige Bemühungen abzunehmen. Die Verteilung der Software geschieht mithilfe des NuGet Package Managers. Softwarepakete vom Package Manager können auf das derzeitige Pro-

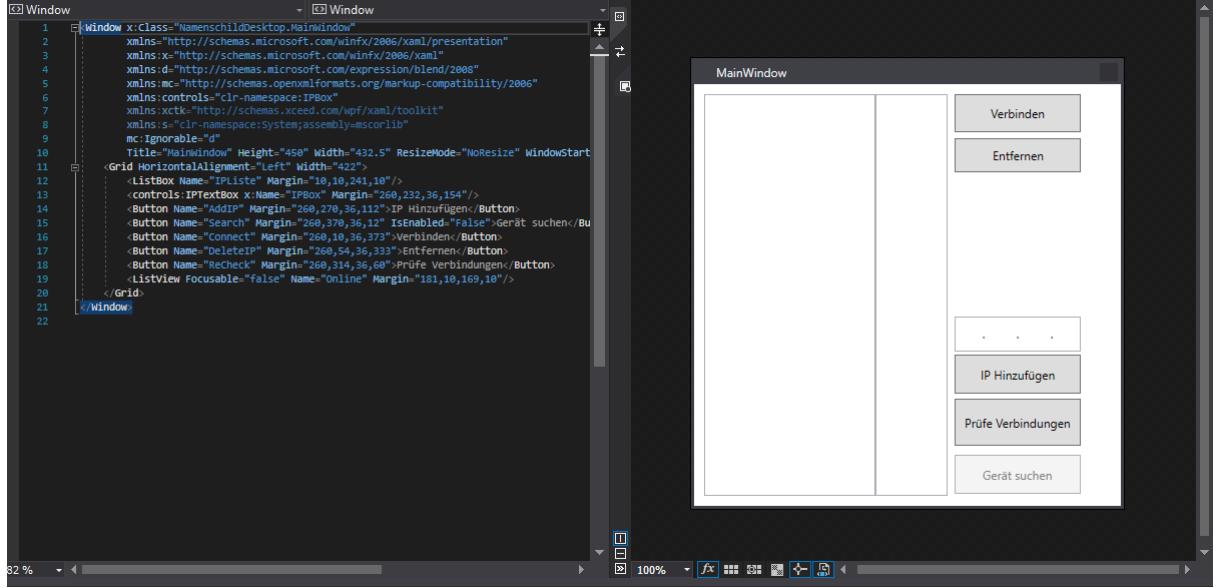


Abbildung 2.15: Echtbilddarstellung von Microsoft Visual 2017. Links Quellcode in Xamarin; Recht: Visualisierung des Quellcodes

jetzt installiert werden. Die Installation der Softwarepakete verläuft automatisch. Nach dem Vorgang können die Funktionen des Softwarepaketes bei der eigenen Entwicklung genutzt werden. Die Sammlung umfasst derzeit etwa über 140.000 verschiedene Softwarepakete. Eine weitere große Extrafunktion ist die HockeyApp. Mit der App ist es möglich die eigene Implementierung der Öffentlichkeit für die Stabilitätsanalyse zur Verfügung zu stellen. Die Software kann heruntergeladen und genutzt werden. Statistiken und Abstürze von Usern werden gesammelt und den Entwickler zur Verfügung gestellt. Weiterhin werden Aktualisierungen automatisch ausgeführt [5].

Die Funktionen stehen erst zur Verfügung, wenn der einzelne Nutzer der Applikation damit einverstanden ist.

2.6.3 Universelle Windows Plattform (UWP)

Eine Universelle Windows Plattform ist eine Laufzeitumgebung für Apps auf Windows Geräten. Die Entwicklung einer Windows Applikation für diese Laufzeitumgebung ist eine UWP-App. Die App ist damit auf jedem Windows Device installierbar und nutzbar. Die Installation der einzelnen Applikationen verläuft durch ein einfaches Package-System, statt einer typischen Installation.

Derzeit kompatible Geräte für UWP-Apps sind die HoloLens, Mobile, PC, Xbox, Surface Hub und diverse andere IoT Geräte. Jeder dieser Geräte muss für die Nutzung eine Variante des Windows Betriebssystems installiert haben. Das Erstellen von UWP-APPS ermöglicht

damit die gleichzeitige Entwicklung für mehrere Geräte.

Die Vorteile sind:

- Eine einheitliche grundlegende API für alle Windows Geräte. Je nach Gerätetyp stehen für jedes einzelne Gerät besondere Funktionen zur Verfügung.
- Sicherheit(Security) durch Autorisierung von einzelnen Funktionen. Die Applikation startet in einer zugriffsbeschränkten Sandbox. Aktivitäten, die nicht freigegeben wurden, können nicht ausgeführt werden. Ein Beispiel ist der Zugriff auf persönliche Daten.
- Eine automatische Anpassung auf die Bildschirmgröße des unterstützten Gerätes.
- Die Unterstützung von mehreren Eingabemöglichkeiten (wie Controller, Touch und Maus) mit Tools.
- Die verfügbare Monetarisierung der App durch den Microsoft Store.
- Bereitgestellte bekannte Windows-Features (Cortana, Live-Kachel, und Weiteres), können für die Entwicklung der eigenen Software verwendet werden [6].

Aufgrund der Beschränktheit auf die Windows Plattform ist einer der größeren Nachteile die Kompatibilität zu anderen Betriebssystemen. Das derzeit beliebteste Betriebssystem ist die Linux Distribution von Raspberry (siehe 2.3.2). Nutzer des Linux Betriebssystems haben keine Möglichkeit die Applikationen auszuführen. Viele Entwickler möchten mit ihren Anwendungen meist die große Menge ansprechen und können dies mit Windows 10 IoT nicht erreichen (siehe 2.3.1).

Ein weiterer Nachteil ist die exklusive Monetarisierung der Applikationen vom Windows Store. Microsoft verlangt 30% des Gewinns bei erfolgreichen Verkäufen von UWP-Apps und andere Veröffentlichungen außerhalb des Stores sind nicht möglich. Eine Ausnahme stellt die IoT Variante dar. Es ist möglich über die vorhandene Web-API des Betriebssystems UWP-Apps zu installieren [6].

2.6.4 Windows Runtime (WinRT)

Bei Windows Runtime handelt es um eine objektorientierte API-Schnittstelle. Sie ist ein fester Bestandteil der universellen Windows Plattform (UWP; siehe Unterabschnitt 2.6.3). Eingeführt wurde sie 2012 mit Windows 8.

Es basiert auf dem Component Object Model (COM). COM ist ein universelles, verteiltes und objektorientiertes System für Softwarekomponenten, die miteinander kommunizieren. Mit COM handelt es sich nicht um eine Programmiersprache, sondern um einen binären

Standard. Der Standard ist binär, da die Regelung erst im übersetzten binären Maschinencode des Programmes greift. Dieser definiert die Voraussetzungen für die Kommunikation zwischen den Objekten. Ein COM-Objekt ist ein Datensatz, der mehrere Sets von Funktionen beinhaltet, die den Datensatz verändern können. Nur durch diese Funktionen ist es möglich den Datensatz zu verändern. Die Funktionen sollen nur indirekt mit Zeigern aufgerufen werden können. Die verwendete Sprache von COM-Objekten kann beliebig sein. Die einzige Voraussetzung ist, dass die objektorientierte Programmiersprache Strukturen von Zeigern beinhaltet und diese Funktionen aufrufen können. Programmiersprachen wie Java und C sind damit nicht in der Lage COM-Objekte zu implementieren.

WinRT kapselt die COM-Objekte unter einer Sprachprojektion und vereinfacht die Nutzung. WinRT passt sich damit der verwendeten Programmiersprache an [2].

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Windows.ApplicationModel.AppService;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.System.Threading;

namespace BackgroundApplication2.Services
{
    public sealed class MessageRelayService
    {
        const string AppServiceName = "UwpMessageRelayService";
        private AppServiceConnection _connection;

        public static MessageRelayService Instance { get; } =
            new MessageRelayService();
        public bool IsConnected => _connection != null;

        private async Task<AppServiceConnection> CachedConnection()
        {
            if (_connection != null) return _connection;
            _connection = await MakeConnection();
            _connection.ServiceClosed
                += ConnectionOnServiceClosed;
            return _connection;
        }

        private void ConnectionOnServiceClosed(AppServiceConnection sender,
                                              object args)
        {
            _connection = null;
        }
    }
}
```

```
    }

    public void Open()
    {
        IAsyncAction asyncAction = ThreadPool.RunAsync(async
            (handler) =>
        {
            await CachedConnection();
        });
    }

    ...
}

}
```

Quellcode 2.11: Ausschnitt der Implementierung eines UWP-Messagerelays in C#

Quellcode 2.11 zeigt die Verwendung von WinRT. Die einzigen Einschränkungen die vorhanden sind, sind bei öffentlichen Methoden zu erkennen, die außerhalb des Objektes aufrufbar sind. Methoden, die öffentlich sind, dürfen bei WinRT keinen variablen generischen Typen haben. List<T>, wobei T ein generischer Typ ist, sind damit nicht verwendbar. Weiterhin können asynchrone Methoden nur mit einer Coroutine implementiert werden. Bei der Coroutine wird am Ende die Rückgabefunktion initiiert. Nach der Initiierung unterbricht sich die Coroutine selbst und gibt bis zur Rückgabe des asynchronen Vorganges die Kontrolle zurück an den Aufrufer (siehe Quellcode 2.11 Methode: public void open). Die Voraussetzungen für die Implementierung eines COM-Objektes sind trotz der Sprachkapselung von WinRT noch gegeben [2].

Kapitel 3

Anforderungen und Konzept

Dieses Kapitel beschäftigt sich mit der Planung des praktischen Teils der Masterarbeit. Es werden zunächst Anforderungen formuliert, die bei der Projektrealisierung erfüllt werden sollen. Anschließend wird ein Konzept für die Realisierung ausgearbeitet.

3.1 Anforderungsanalyse

Für die Realisierung des Ziels sind mehrere Komponenten zu betrachten. Die Realisierung des Projektes benötigt mindestens zwei Anwendungen. Eine Serveranwendung, die die anzuzeigenden Daten verarbeitet und darstellt und ein Clientsystem, das die Daten an die Serveranwendung sendet. Die Kommunikation zwischen den beiden Systemen soll dezentral stattfinden. Jedes dieser Systeme benötigt eine eigene Oberfläche und mehrere Funktionen. Zudem sollen die beiden Anwendungen miteinander kommunizieren können.

Die Clientanwendung soll:

- eine moderne, übersichtliche und einfache Oberfläche besitzen,
- Kommunikationen zwischen mehreren Serveranwendungen ermöglichen,
- Veränderungen der Daten außerhalb des Netzwerkes durch andere Clientanwendungen anzeigen,
- keine unnötige zusätzliche Anforderung besitzen, wie die Installation einer Datenbank,
- eine einfache Bedienoberfläche besitzen, die selbstverständlich ist,
- keine Programmiererfahrung oder ähnliches Wissen für die Nutzung voraussetzen,
- eine gute Validierungen der Eingabe haben und fehlerhafte Eingaben so gut es geht abfangen,

- eine einfache Ausführung der Anwendung ohne Installation ermöglichen,
- (Optional) einen Prüfungsmodus bieten, der aus einer einfachen aber eindeutigen Anzeige besteht,
- (Optional) ein Profilsystem besitzen. Informationen sollen nach Profilbezeichnung lokal gespeichert werden.

Dagegen soll die Serveranwendung:

- eine API besitzen, die die Clientanwendung ansprechen kann,
- keinerlei manuelle Eingaben zulassen,
- Darstellung der einzelnen Übersichten (Profil, Aktuelles, Veranstaltungen und Belegungsplan) haben,
- die Übersichten nach Veränderung selbstständig anpassen,
- nach einer Abfrage des Clients, die aktuellen Daten an den Client senden,
- eine selektive Veränderung der Daten durch die Clienten ermöglichen,
- auf einem IoT-Board verwendetet werden können,
- ein einfaches Verzeichnis für die Speicherung von Daten verwenden.

Die oben genannten Bedingungen sollen bestmöglich realisiert werden. Die verwendeten Werkzeuge dafür sind freigestellt.

3.2 Konzept

Dieser Abschnitt zeigt einen groben Aufbau für die Realisierung des Projektes. Zunächst werden die zu benutzenden Hilfsmittel aus den Grundlagen Kapitel (Kapitel 2) bestimmt. Danach werden die einzelnen Komponenten von der Anforderungsanalyse grob dargestellt.

3.2.1 Auswahl der Hilfsmittel

Wie in der Anforderungsanalyse erwähnt müssen mindestens zwei Anwendungen entwickelt werden. Eine Client- und eine Serveranwendung sind notwendig. Die Hauptanwendung zum Darstellen und Verwalten der Daten findet auf dem IoT Board statt. Deshalb wird die Serveranwendung auf dem IoT Board entwickelt. In Folge dessen wird die Clientanwendung auf dem Desktop PC realisiert.

Die Serveranwendung muss jederzeit Daten speichern, senden und empfangen können. Diese Aktivitäten müssen parallel und asynchron stattfinden. Ein Mikrocontroller (siehe Unterabschnitt 2.1.2 und 2.1.3) ist nicht leistungsstark und kann schlecht mehrere Prozesse

auf einmal koordinieren. Deshalb fallen der Uno Rev 3 und der Esp-Wrover-Kit-VB als Implementierungshardware raus. Der Raspberry Pi 3 ist ein vollständiger **Single Board Computer**, welches ohne Probleme parallele Prozesse und Nebenläufigkeit ermöglicht (siehe Unterabschnitt 2.1.1). Deshalb wird als Hardware das Raspberry Pi 3 genutzt.

Das Raspberry Pi 3 benötigt für seine Funktionalitäten ein Betriebssystem. Das Projekt kann auf allen der Betriebssysteme in Abschnitt 2.3 realisiert werden. Aufgrund der Performance und der Universalität ist Raspbian im Vergleich zur Android Things und Windows 10 IoT leistungsschwächer. Android Things und Windows 10 IoT sind beide angepasst auf den IoT Betrieb. Ein Treiber für die Verwendung des Bildschirms ist bei beiden mit enthalten. Die Kommunikation zwischen Desktop PC und IoT Board ist mit einem Ethernet-Anschluss oder mit einer WLAN-Verbindung möglich. Windows 10 IoT besitzt im Vergleich zu Android Things mehr Funktionen. Eine eingebaute API für die internetbasierte Fernsteuerung des IoT-Boards und die Abfrage von Benutzernamen und Passwort sind einige dieser Funktionen (mehr in Abschnitt 2.3). Weiterhin ist Android Things fehleranfälliger als Windows 10 IoT. Android Things ist im Vergleich zu Windows 10 IoT noch im Anfangsstadium seiner Entwicklung. Aufgrund der aufgeführten Punkte wird das Betriebssystem des Raspberry Pi 3 Windows 10 IoT sein.

Die Sprachen für die Modellierung wird aufgrund der Wahl des Betriebssystems C# (Unterabschnitt 2.6.1) und XAML (Unterabschnitt 2.5.1) sein.

Als Programmierumgebung wird Microsoft Visual 2017 verwendet. Grund dafür ist die eingebaute Unterstützung für die Entwicklung von UWP-Anwendungen. Windows 10 IoT Anwendungen sind alle UWP Anwendungen. UWP sind angepasste Anwendungen auf verschiedensten Windowsoberflächen einschließlich der Desktopumgebung. Mehr Informationen sind im Unterabschnitt 2.6.3 zu finden.

Der Web API Service zwischen Server und Client kann je nach Betriebssystem in SOAP oder in REST realisiert werden (siehe Abschnitt 2.4). Windows 10 IoT unterstützt derzeit SOAP nicht. Deshalb wird die Realisierung der API in REST stattfinden.

3.2.2 Grobe Planung der einzelnen Anwendungen

Dieser Abschnitt beschäftigt sich mit der groben Planung der einzelnen Komponenten der beiden Anwendungen. Es werden die Funktionalitäten beschrieben, die die Anwendungen erfüllen müssen. Weiterhin wird eine grundlegende Architektur aufgesetzt, die bei der Implementierung umgesetzt werden soll.

Funktionsrealisierung der IoT-Anwendung

Die IoT-Anwendung (Serveranwendung) muss verschiedene Funktionen erfüllen.

Speichern und darstellen von Informationen. Eines davon ist das Speichern und Darstellen von Informationen. Die Informationen unterscheiden sich in vier Kategorien (Profil, Aktuelles, Veranstaltungen und Belegungsplan). Die Darstellung der Informationen muss getrennt geschehen. Ein Zeitschalter soll die Informationen auf dem Display ändern.

Wenn die Daten sich während des Betriebs ändern, soll spätestens nach dem nächsten Schalten die aktuellsten Daten angezeigt werden.

Senden von Informationen an die Clientanwendung. Aktuelle Informationen müssen an den Client nach Abruf vollständig gesendet werden. Die Vollständigkeit bezieht sich hierbei auf die einzelnen oben genannten Kategorien. Kategorienübergreifende Informationen sollen nur nach Abruf gesendet werden.

Strukturierte Darstellung von Informationen durch Hilfsmittel. Die Darstellung der Daten muss strukturiert sein. Die Informationen in den einzelnen Kategorien müssen so dargestellt werden, dass ein kurzer Blick genügt, diese zu erkennen. Dies wird durch Schriftgrößen, Schriftfarbe, Zeilenumbrüche und Tabellen geschehen.

Speicherung der Daten. Valide Daten müssen nach Eingang gespeichert werden. Die Speicherung erfolgt in einem Verzeichnissystem. Dabei werden die Informationen basierend auf ihrer Kategorie in verschiedene Verzeichnisse gespeichert. Bei den gespeicherten Daten wird es sich um JSON Dateien handeln.

Ein- und Ausschaltung von Kategorien. Einzelne Kategorien sollen abgeschaltet und eingeschaltet werden können. Bei einer Abschaltung sollen die gespeicherten Daten dieser Kategorien nicht mehr angezeigt werden. Die Abschaltung wird farblich dargestellt. Erst wenn die Kategorie eingeschaltet wird, werden die gespeicherten Daten wieder angezeigt.

Permanente Darstellung von Daten. Einige Daten werden permanent dargestellt. Die Raumnummer, der Inhaber des Raumes, die Universität, die Fakultät und die Fakultätsbezeichnung gehören zu diesen Daten. Während die Raumnummer, der Inhaber und die Fakultätsbezeichnung textlich angezeigt werden, wird die Universität und die Fakultät jeweils mit der passenden Bilddatei dargestellt. Die Symbole werden als PNG-Dateien gespeichert.

Aufruf/Schließen der Prüfungsalarmfunktion (Optional). Eine optionale Funktion ist das Ein- und Ausblenden eines Prüfungsalarms für einzelne Räume. Bei diesen Funktionen wird eine neue GUI aufgelegt, die den Betrachter darauf hinweist, dass derzeit eine Prüfung stattfindet und deshalb um Ruhe gebeten wird. Die Hauptanwendungen bleiben im Hintergrund im Betrieb. Für die zusätzliche Funktion mit

neuer Oberfläche wird eine zusätzliche Anwendung geschrieben. Die Anwendung ist schlicht und beinhaltet nur eine Anzeigefunktion.

Die beiden Hauptbereiche der IoT-Anwendung sind das Speichern und Anzeigen von Informationen, sowie das Versenden und Empfangen von Daten. Für ein wartungsfreundliches Entwickeln wird die IoT Anwendung in zwei Unteranwendungen geteilt. Die erste Anwendung soll die Informationen speichern und anzeigen und die andere Anwendung die Kommunikation mit einer Client-Anwendung ermöglichen. Abbildung 3.1 soll das Ganze veranschaulichen.

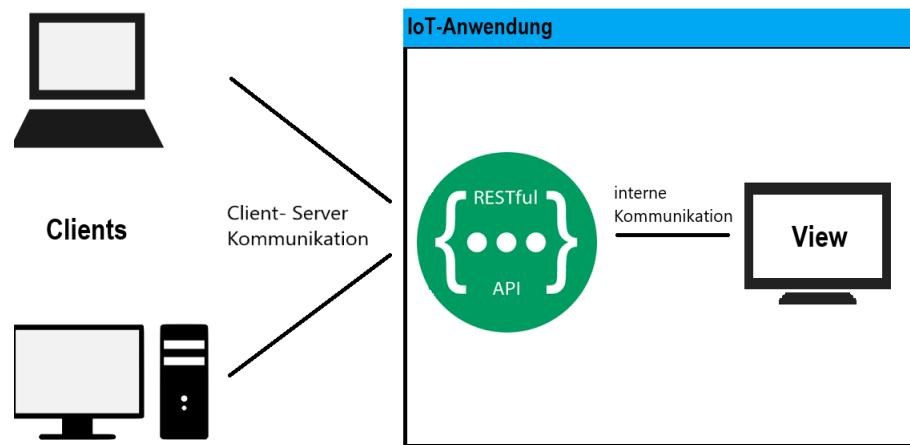


Abbildung 3.1: Die Grundarchitektur der Serveranwendung

Eine feinere Darstellung der Architektur ist in Abschnitt 4.1 zu finden.

Funktionsrealisierung der Desktop-Anwendung

Der Funktionsbereich der Desktop-Anwendung (Clientanwendung) unterscheiden sich von der IoT-Anwendung.

Verwalten von mehreren IPs. Eines davon ist der Aufbau einer Kommunikation zwischen den IoT-Anwendungen. Die Desktop-Anwendung soll dabei eine IP-Liste der einzelnen IoT-Anwendungen verwalten. Der Nutzer soll diese Liste beliebig erweitern können. Die IP wird nach jedem Start der Serveranwendung angepingt. Wenn die IP-Adresse nicht im Netzwerk erreichbar ist, wird das sichtbar dargestellt. Der Nutzer kann dennoch die hinterlegten Dateien des Gerätes ändern aber diese nicht an das IoT-Board senden. Erst wenn der Server erreichbar ist, können die bearbeiteten Daten gesendet werden. Das Senden der Daten geschieht nicht automatisch. Es muss vom Nutzer eingeleitet werden.

Authentifizierung der jeweiligen IoT-Anwendungen. Nach Aufruf einer IP-Adresse wird eine Authentifizierung für das Gerät verlangt. Bei der Authentifizierung handelt es sich um die Administrator-Anmeldedaten für das IoT-Board. Diese werden bei der Installation des Gerätes gesetzt. Für die Einrichtung des Gerätes sind diese Informationen notwendig. Ohne diese Informationen ist es nicht möglich die bereitgestellte API des Betriebssystems anzusprechen.

Änderung von Daten des IoT-Betriebssystems. Nach erstmaliger Authentifizierung werden die Hardwaredaten des IoT Boards abgefragt. Gerätename, Benutzername, Password und IP-Adresse werden angezeigt und können teilweise geändert werden. Bei den Informationen handelt es sich um grundlegende Daten, die unabhängig sind von der eigen-implementierten API.

Installation der IoT-Anwendung auf dem IoT-Board. Zudem soll es ein Button für die Installation der IoT-Anwendung auf dem IoT-Board mit Windows 10 IoT geben. Dabei werden die einzelnen Komponenten der IoT-Anwendung mit ihrer Abhängigkeiten auf dem Board bereitgestellt und installiert. Nach der Installation wird die View-Anwendung als Startanwendung markiert und die Restful-API nach jedem Start des Betriebssystems im Hintergrund ausgeführt. Die Anwendung sendet dabei die Anwendungspakete an die Standard-API. Nach der Installation der Pakete stehen dem Nutzer mehr Funktionen zur Verfügung.

Deinstallation der IoT-Anwendung auf dem IoT-Board. Eine Deinstallation der IoT-Anwendung mit der Clientanwendung soll ebenfalls möglich sein. Die Anwendungen ohne ihre Abhängigkeiten werden dabei vom IoT-System entfernt. Auch hier handelt es sich um eine Standardfunktion der vom Betriebssystem bereitgestellten API.

Änderung von Daten der IoT-Anwendung. Nach der Installation der IoT-Anwendung können einzelne Informationen über die genannten Kategorien aufgerufen und verändert werden. Eine De/Aktivierung einzelner Kategorien ist ebenfalls möglich. Die Informationen sollen auch hier strukturiert dargestellt werden. Der Nutzer soll in der Lage sein, ohne Vorkenntnisse Daten zu erstellen und zu ändern.

Lokale Speicherung von Daten. Jede Veränderung der Informationen zu den einzelnen Kategorien (Profil, Aktuelles, ...) werden zusätzlich nach dem erfolgreichen Versenden lokal gespeichert. Die Speicherung wird in einem Verzeichnissystem geschehen. Das Dateiformat ist in JSON.

Selektives Senden von Daten an die IoT-Anwendung. Die Veränderungen, die in einzelnen Kategorien gemacht worden sind, werden von der Clientanwendung geprüft und an die IoT-Anwendung versendet. Die Clientanwendung prüft, ob die

Daten der einzelnen Kategorien identisch der lokal gespeicherten Daten sind. Jedes Eingabeformular wird nach einer Veränderung untersucht. Wenn die Informationen identisch sind, werden keine Daten der jeweiligen Kategorie an die IoT-Anwendung versendet.

Funktionen für die Wartung der IoT-Anwendung. Zusätzlich wird für die Wartung eine Neustart- und eine Aufnahmefunktion implementiert. Diese Funktionen sollen zur möglichen Kontrolle und Problemlösung bei geänderten Informationen dienen. Die Funktionen sind in der Standard-API hinterlegt.

Verwendung eines Profilsystems (Optional). Optional wird ein Profilsystem eingebaut. Das Speichersystem wird dabei erweitert. Die Daten werden in verschiedene Verzeichnisse gespeichert. Die Verzeichnisse werden nach den gesetzten Profilnamen benannt. Sinn und Zweck dieser Funktion ist es, dass bei einem stetigen Wechsel der Daten, es nicht notwendig ist gleiche Informationen in die Anwendung einzugeben. Es handelt sich hier um eine Komfortfunktion für den Nutzer.

(Optional) Aufruf des Prüfungsalarms. Weiterhin soll die Ausführung des Prüfungsalarms in Unterunterabschnitt 3.2.2 beschrieben. Ein- und abgeschaltet werden können.

Die Clientanwendung wird als alleinstehende Anwendung implementiert. Eine Trennung in mehrere Anwendungen ist hier unnötig. Die eingegebenen Funktionen werden direkt an die IoT-Anwendung versendet und benötigen keine weitere Verarbeitung.

View der IoT-Anwendung (Entwürfe)

Dieser Unterunterabschnitt beschäftigt sich mit der Anzeige der einzelnen Kategorien auf-seiten der IoT-Anwendung. Wie in Abschnitt 3 erwähnt, soll die Oberfläche schlicht gehalten werden, um eine benutzerfreundliche Bedienung zu ermöglichen. Gleichzeitig ist es notwendig einfache und selbstverständliche Funktionen den Nutzer zur Verfügung zu stellen. Bei den folgenden Darstellungen handelt es sich um Entwürfe, die sich während der Implementierungen ändern können. Für die Realisierung des Projektes werden folgende Entwürfe angestrebt.

Grundgerüst. Das Grundgerüst der Applikation wird permanent angezeigt. Es beinhaltet die Raumnummer, den Inhaber des Raumes mit dem Symbol des Bildungsinstituts und der Fakultät, sowie die einzelnen darzustellenden Kategorien (Profil, Aktuelles, Veranstaltungen, Belegungsplan und Wetter). Die Darstellungen der einzelnen Kategorien ändern sich durch einen Zeitschalter und sind damit nicht permanent sondern variabel. Das Gerüst umschließt die Darstellung der Kategorien. Abbildung 3.2 verdeutlicht das Ganze.

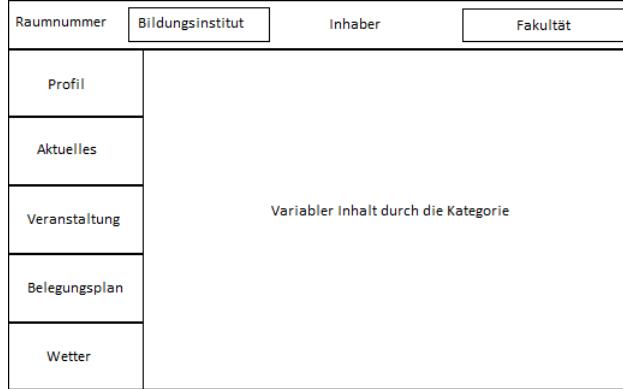


Abbildung 3.2: Entwurf des Grundgerüsts der IoT-Anwendung

Profilinformationen. Die Profile, die abgebildet werden, sind die Inhaber des Raumes. Die Anwendung wird maximal drei Profile für ein Raum anzeigen. Für die Darstellung sind drei verschiedene Views notwendig.

1. **View mit einem Profil.** Die Informationen, die eingeblendet werden sollen, sind Titel, Vorname, Nachname, Straße, Gebäudenummer, Telefon, Fax, E-Mail und ein Profilbild im PNG-Format. Eine mögliche Darstellung zeigt Abbildung 3.3.
2. **View mit zwei Profilen.** Das Fenster wird in zwei Teile geteilt. Ein Teil der Information fällt für die Darstellung weg. Es werden nur Titel, Vorname, Nachname, Telefon, Fax und E-Mail angezeigt. Siehe Abbildung 3.4.
3. **View mit drei Profilen.** Das Fenster wird in drei Teile geteilt. Die Informationen bleiben gleich der 2. View. Es ändern sich nur die Schriftgröße und die Größe des Profilbildes. Siehe Abbildung 3.5.

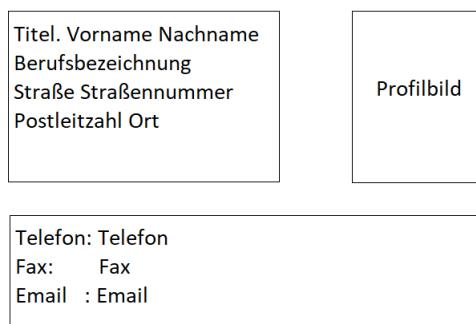


Abbildung 3.3: Entwurf einer einzelnen Profildarstellung

Empfängt der Server mehrere Profile statt einen, wechselt er zu den oben gezeigten Varianten des Layouts.

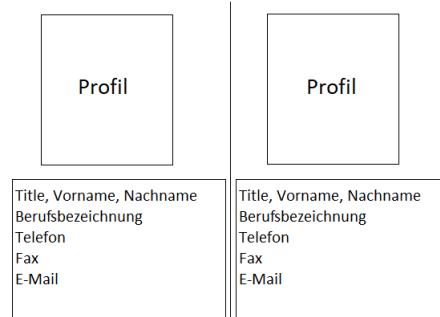


Abbildung 3.4: Entwurf einer dualen Profildarstellung

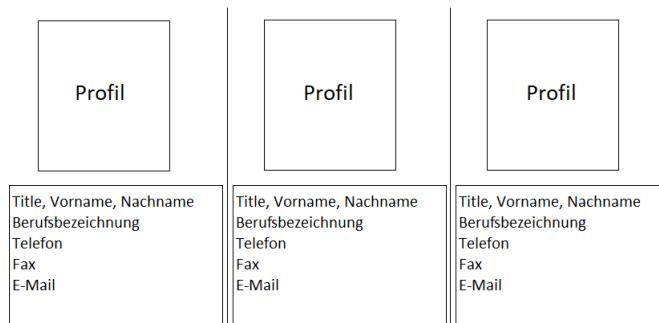


Abbildung 3.5: Entwurf einer triple Profildarstellung

Aktuelles. Die Kategorie Aktuelles beschäftigt sich mit Ereignissen, die aufgetreten sind bzw. werden. Ein Beispiel für ein aufgetretenes Ereignis wäre der krankheitsbedingte Ausfall eines Dozenten. Diese Information wird in dieser Kategorie angezeigt. Die Darstellung der Information ist im reinen Textformat. Es existiert eine Überschrift mit einer größeren und dickeren Schrift und einen anschließenden normal großen Text. Siehe Abbildung 3.14.

Veranstaltungen. Hier werden die stattfindenden Veranstaltungen mit Tag und Uhrzeit für den jeweiligen Raum angezeigt. Es können bis zu vier Veranstaltungen gleichzeitig angezeigt werden. Der Grund für diese Beschränkung ist der vordefinierte Platz auf der View. Abbildung 3.6 zeigt einen Entwurf der Oberfläche für vier Veranstaltungen.

Bei weniger als vier Veranstaltungen ändert sich das Layout nicht. Es werden nur weniger angezeigt.

Belegungsplan. Der Belegungsplan beinhaltet die Belegung eines Raumes für den Zeitraum 08:00 - 20:00 Uhr von Montag bis Freitag. In dieser Zeitspanne werden alle Veranstaltungen tabellarisch aufgelistet. Abbildung 3.7 veranschaulicht das Ganze.

Wetter. Die Kategorie beinhaltet das Wetter für die nächsten vier Tage. In jeder Zeile ist der Tag, die Mindest- und Höchsttemperatur und ein Icon. Die notwendigen Informa-

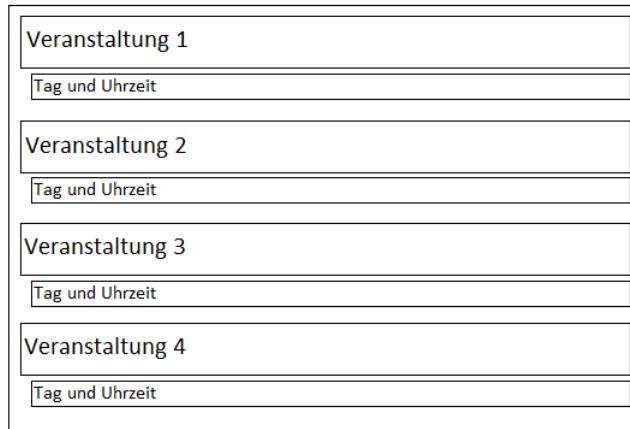


Abbildung 3.6: Entwurf mit vier Veranstaltungen

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
8:00-10:00					
10:00 -12:00		Veranstaltung			
12:00 -14:00		Veranstaltung			
14:00 -16:00				Veranstaltung	
16:00 -18:00					
18:00 -20:00					

Abbildung 3.7: Entwurf eines Belegungsplans mit drei Veranstaltungen

tionen werden von einem Online-Dienst entnommen. Abbildung 3.8 stellt ein Entwurf des Genannten dar.

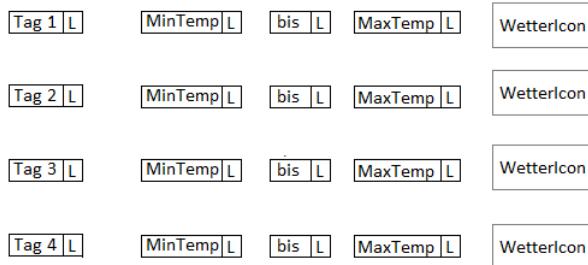


Abbildung 3.8: Entwurf der Kategorie Wetter

View der Desktop-Anwendung (Entwürfe)

Anders als bei der View der IoT-Anwendung werden hier vom Nutzer Eingaben getätigt. Die Aufgabe der View ist es eine klare durchgehende Übersicht für den Nutzer zu schaffen. Änderungen durch Eingaben und vorhandene Funktionalitäten (siehe Unterunterabschnitt 3.2.2) sollen sofort erkannt und verstanden werden. Wie bei der View der IoT-Anwendung handelt es sich hier um Entwürfe, die sich während der Implementierung ändern können. Trotzdem wird die Realisierung der Entwürfe angestrebt.

IP-Liste. Die IP-Liste ist das erste Fenster, welches der Nutzer öffnen soll. Dort befinden sich eine Liste der vom Nutzer eingetragenen IP-Adressen. Jede IP-Adresse ist die Zugangsadresse für die Verwaltung der einzelnen IoT-Anwendungen. Der Nutzer kann die Adresse bearbeiten und einen Verbindungsaufbau mit der IoT-Anwendung des dazugehörigen Geräts ausführen. Die Bearbeitung der Adressen beinhaltet das Löschen, Hinzufügen und das Prüfen von Verbindung nach Erreichbarkeit der IoT-Anwendung.

Das Nutzen dieser Funktionalitäten soll durch einzelne Buttons geschehen. Abbildung 3.9 stellt einen Entwurf des oben genannten dar.

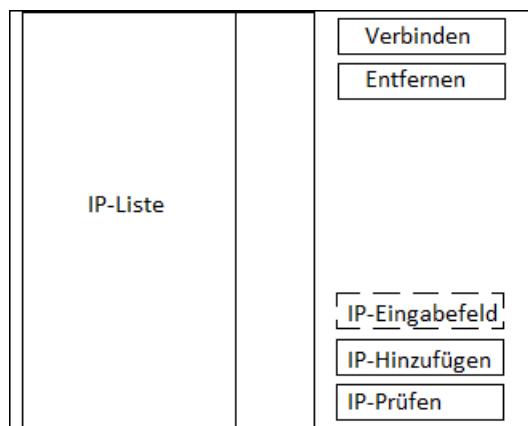


Abbildung 3.9: Entwurf der IP-Liste (Durchgehende Kästen sind Buttons; Kästen mit Lücken sind Textfelder)

Grundgerüst des Clients. Beim Verbinden mit der IoT-Anwendung soll ein separates Fenster geöffnet werden. Das Fenster ist nur für die Kommunikation einer IoT-Anwendung mit der Desktop-Anwendung zuständig. In diesem sollen die Informationen über das IoT-Board angezeigt, einzelne Kategorien eingeschaltet und bearbeitet werden können. Eine Navigation durch Tabs soll einzelne Kategorien durch einen Klick anzeigen. Standardfunktionen wie das Speichern, Senden und das Empfangen sind unabhängig von der Navigation und werden dauerhaft angezeigt. Die optionale Funktion zur Darstellung eines Prüfungsalarms soll ebenfalls dauerhaft im Fenster enthalten sein. Die Realisierung der

Funktionen geschieht durch Buttons. Ein Entwurf der Oberfläche ist in Abbildung 3.10 zu sehen.

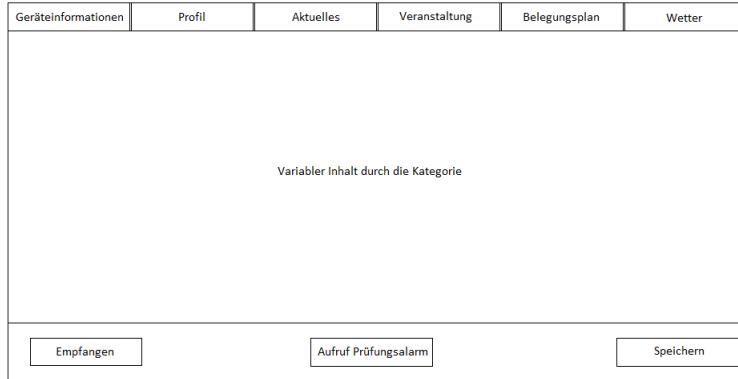


Abbildung 3.10: Entwurf des Verbindungsfensters (Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld))

Geräteinformation. Gerätinformationen wie die Authentifizierungsdaten des Nutzers, der Gerätename, Netzwerkadressen und Verbindungsart, sowie die Version des aktuell installierten Betriebssystems werden hier dargestellt. Weiterhin bietet es Funktionen, die an die Standard-API des Betriebssystems gebunden sind. Die implementierte IoT-Anwendung (siehe Unterabschnitt 3.2.2) mit all ihren Komponenten kann hier installiert und entfernt werden. Ein Screenshot des aktuell dargestellten Bildes des Geräts kann angezeigt werden. Ein Neustart des Gerätes ist ebenfalls möglich. Zudem soll es optional möglich seine Profile zu verwalten und anzulegen. Abbildung 3.11 zeigt eine skizzierte Darstellung der Oberfläche.

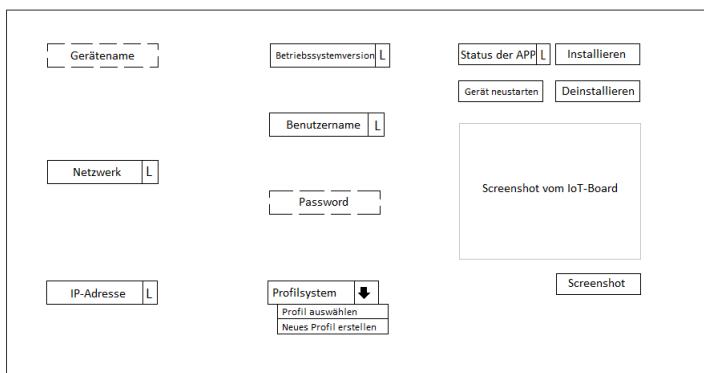


Abbildung 3.11: Entwurf der Gerätinformationsoberfläche. L = Label (Information kann vom Benutzer nicht verändert werden); Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld); Nach unten zeigender Pfeil (Dropdown-Menü)

Kategorie Profile. In dieser Kategorie ist es möglich bis zu drei Profile zu erstellen und diese zu verwalten. Der Inhalt der Profile ist in Unterunterabschnitt 3.2.2 bereits aufgelistet. Die Aufteilung der Profile geschieht durch Tabs. Der Aufbau ist ähnlich dem oben genannten Grundgerüst des Programmes. Abbildung 3.12 zeigt ein Entwurf der Oberfläche für ein Profil.

Tab einschalten	C			
Infoleiste	Profil 1	Profil 2	Profil 3	
Profil einschalten	C			
<input type="text"/> Titel				
Vorname	Straße	FAX		
Name	PLZ	Telefon		
Beruf	Ort	E-Mail		
<input type="button"/> Profil <input type="button"/> Profilbild hochladen				

Abbildung 3.12: Entwurf der Verwaltungsoberfläche für ein Profil von insgesamt drei möglichen. L = Label (Information kann vom Benutzer nicht verändert werden); C = Combobox; Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld)

Zusätzlich wird die Infoleiste in diese Kategorie zugeordnet, mit der Begründung, dass es sich hierbei um ein eingeschränktes Profil handelt. Vorname, Name und Beruf, sowie zwei Logos können hier eingetragen werden. Siehe Abbildung 3.13.

Tab einschalten	C			
Infoleiste	Profil 1	Profil 2	Profil 3	
<input type="text"/> Vorname				
<input type="text"/> Name				
<input type="text"/> Beruf				
<input type="button"/> Linkes Logo			<input type="button"/> Rechtes Logo	
<input type="button"/> Bild hochladen			<input type="button"/> Bild hochladen	

Abbildung 3.13: Entwurf der Verwaltungsoberfläche für die Informationsleiste. Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld)

Kategorie Aktuelles. Diese Kategorie stellt aktuelle Ereignisse im Fließtext dar. Angezeigt werden eine Überschrift mit vergrößerter Schrift und ein Fließtext in Standardschrift-

größte. Abbildung 3.14 zeigt ein Entwurf vom Ganzen.

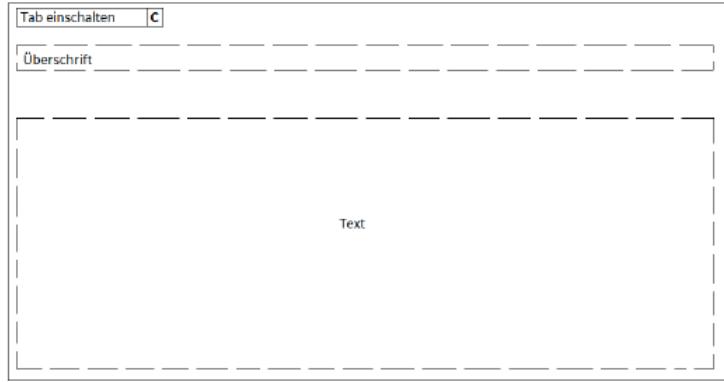


Abbildung 3.14: Entwurf der Verwaltungsoberfläche für die Kategorie Aktuelles. C = Combobox; Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld)

Kategorie Veranstaltung. Hier ist es mögliche mehrere Veranstaltungen zu erstellen. Benötigt werden jeweils der Titel, Tag, Beginn und das Ende der Veranstaltung. Für die Verbesserung der Benutzerfreundlichkeit werden drei Dropdown-Boxes für jede Veranstaltung benötigt. Die erste Box stellt den Tag von Montag bis Freitag dar. Die zweite und die dritte Box Den Beginn und das Ende der Veranstaltung. Der folgende Entwurf (siehe Abbildung 3.15) soll das Erzählte verdeutlichen.

Tab einschalten C			
Anzahl der Veranstaltungen ↓			
1; 2; 3; 4			
Titel	Tag ↓ Mo;Di;Mi;Do;Fr	Von L Uhrzeit ↓ 08-20	Bis L Uhrzeit ↓ 08-20
Titel	Tag ↓ Mo;Di;Mi;Do;Fr	Von L Uhrzeit ↓ 08-20	Bis L Uhrzeit ↓ 08-20
Titel	Tag ↓ Mo;Di;Mi;Do;Fr	Von L Uhrzeit ↓ 08-20	Bis L Uhrzeit ↓ 08-20
Titel	Tag ↓ Mo;Di;Mi;Do;Fr	Von L Uhrzeit ↓ 08-20	Bis L Uhrzeit ↓ 08-20

Abbildung 3.15: Entwurf der Geräteinformationsoberfläche. L = Label (Information kann vom Benutzer nicht verändert werden); C = Combobox; Kasten mit Lücken (Textfeld); Nach unten zeigender Pfeil (Dropdown-Menü)

Kategorie Belegungsplan. Beim Belegungsplan werden Termine in eine Tabelle einge tragen. Für einen Eintrag ist der Titel der Veranstaltung, der Tag, der Beginn und das Ende notwendig. Die Einträge Beginn und Ende werden als Uhrzeit angegeben. Die Tabelle

soll sich nach jeden Eintrag aktualisieren. Die Einträge auf der Tabelle können markiert werden. Nach einer Markierung ist es möglich den Eintrag wieder zu entfernen. Das Entfernen geschieht durch einen Klick auf den entsprechenden Button. Einträge können nicht übereinander gestapelt werden. Der Entwurf 3.16 spiegelt die dargestellten Eigenschaften wider.

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag	
08:00 - 10:00 L	Veranstaltung (markierbar)					
10:00 - 12:00 L						
12:00 - 14:00 L						
14:00 - 16:00 L			Veranstaltung (markierbar)			
16:00 - 18:00 L						
18:00 - 20:00 L						

Veranstaltungsname _____
 Tag
 Mo,Di,Mi,Do,Fr
 Von Bis
 Uhrzeit 08:00-20:00 Uhrzeit 08:00-20:00

Abbildung 3.16: Entwurf der Geräteinformationsoberfläche. L = Label (Information kann vom Benutzer nicht verändert werden); C = Combobox; Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld); Nach unten zeigender Pfeil (Dropdown-Menü)

Kategorie Wetter. Beim Wetter ist ein API-Key zum Verifizieren mit dem Online-Dienst notwendig (OpenWeatherMap). Der Nutzer kann hier ein selbsterstellten API-Key einsetzen. Für die Eingabe ist ein einfaches Textfeld notwendig (siehe Abbildung 3.17).

Tab einschalten <input type="button" value="C"/>
API-Key _____

Abbildung 3.17: Entwurf der Wetteroberfläche für den Desktop. Kasten mit Lücken (Textfeld).

Kapitel 4

Anwendungsentwurf und -implementierung

Dieses Kapitel beinhaltet die Fortsetzung der Architektur und die Implementierung der Funktionen die grob in Abschnitt 3.2.2 erläutert worden sind. Zunächst wird die finale Architektur der Anwendung dargestellt, anschließend werden bei der Implementierung die Funktionen, die in Abschnitt 3.2.2 aufgelistet sind, schrittweise mithilfe von Quellcode, Abbildungen und Bildschirmaufzeichnungen aufgelistet.

4.1 Architektur der Anwendung

Zu betrachten sind zwei Unterteilungen der Anwendung. Die Erste ist die Serveranwendung, welches Daten anzeigt und speichert. Die zweite ist die Clientanwendung, die die Daten durch die Eingabe des Nutzers an die Serveranwendung sendet.

Die Serveranwendung unterteilt sich in die Standard-API und in vier separate Unteranwendungen, die abhängig zueinander sind. Bei den abhängigen Anwendungen handelt es sich um eine eigene implementierte API, die die Standard API erweitert, eine View, die die Daten auf dem Bildschirm des Raspberry Pi 3 anzeigt, einem UWP-Messagerelay-Service, die eine Kommunikation zwischen den Anwendungen ermöglicht und der Prüfungsapp, die eine separate Oberfläche anzeigt.

4.1.1 Kommunikationswege der Anwendung

Im Folgenden werden die Kommunikationswege zwischen den Anwendungen erläutert.

Standard-API. Die Standard-API von Windows ermöglicht es, Anwendungen bereitzustellen und zu installieren. Für die Bereitstellung der Serveranwendung kontaktiert der Client den Server durch die Standard-API und versendet die zu installierende Pakete samt ihrer Abhängigkeiten an den Server. Der Server installiert nach

einer erfolgreichen Authentifizierung des Clients die vier Anwendungen auf dem Server. Die API steht bereits nach der Installation des Betriebssystems den Nutzer zur Verfügung.

Eigene implementierte API. Die eigene API erweitert die Standard-API, um Funktionen für den Austausch von Daten. Bei den Erweiterungen handelt es sich, um die Verwaltung der Informationen zu den Kategorien (Hardwareinformationen, Tabs, Infoleiste, Profile, Aktuelles, Veranstaltungen, und Belegungsplan). Für die Kategorie Hardwareinformationen besteht eine Kommunikation zwischen Standard-API und eigene implementierte API. Genauer werden die Funktionen für die Netzwerkdaten und die Bildschirmaufnahme aufgerufen.

UWP-Messagerelay-Service (UMS). Der UMS dient der Interprozesskommunikation zwischen der eigenen implementierten API und der View. Es handelt sich um eine Message Queue, wo Nachrichten vom Sender in eine Nachrichtenschlange gesetzt werden, die dann vom Empfänger abgeholt werden können. Der Sender ist hier die eigene implementierte API und der Empfänger die View.

View der IoT-Anwendung. Die View empfängt Daten von der eigenen implementierten API durch das UMS und stellt diese dar. Außerhalb existiert keine Kommunikation zwischen anderen Unteranwendungen.

Prüfungsalarm Applikation. Die Applikation wird von der eigen implementierten API aufgerufen und beendet. Ansonsten besteht keine Kommunikation zu den anderen Unteranwendungen.

Clientanwendung. Die Clientanwendung besteht aus einer einzigen Anwendung. Es kommuniziert anfangs nur mit der Standard-API. Nach der Installation der eigenen implementierten API erweiterten sich die Kommunikation auf diesen. Es empfängt und sendet Daten an die Standard-API und die eigene implementierte API.

Die Abbildung 4.1 stellt die oben genannte Abhängigkeiten zwischen den Anwendungen bildlich dar.

4.1.2 Aufbau der Anwendungen

Jede (Unter-)Anwendung ist in drei Komponente unterteilt. Die Komponenten heißen Model, View und Control (MVC). Jeder dieser Komponenten besitzt unterschiedliche Bereiche der Anwendung.

Die **Model** Komponente enthält die Informationen über die Kategorien, die von der Clientanwendung empfangen und von der der View der Serveranwendung dargestellt werden.

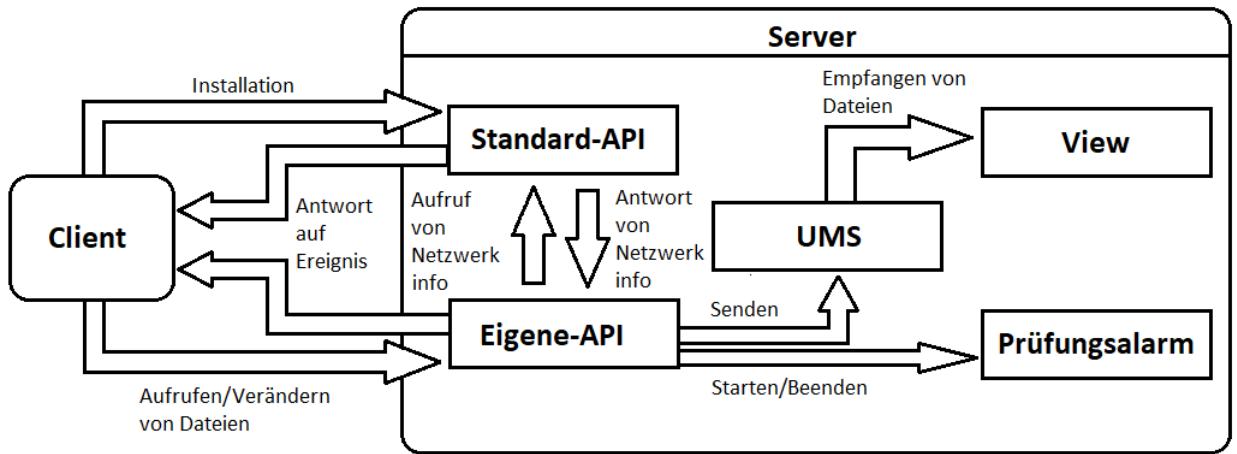


Abbildung 4.1: Kommunikationswege zwischen den Anwendungen

Die Komponente enthält keine Funktionen und keine Darstellungen. Es sind reine Objektdateien. Für jede Kategorie (Tabs, Infoleiste, Profil, Aktuelles, Veranstaltung, Hardwareinformationen, Belegungsplan und Wetter) existiert jeweils ein Model-Objekt.

Die **View**-Komponente bezieht sich auf die Darstellung von Funktionsergebnissen (aus den Control-Objekten) und von Dateien (aus den Model-Objekten). Weiterhin leitet es Nutzereingaben weiter zu den Control-Objekten. Es ist die Oberfläche die der Nutzer sieht und bedienen kann. Jedes separate Fenster (ausschließlich des Dialogfensters) ist ein eigenes View-Objekt.

Die Serveranwendung hat insgesamt zwei View-Klassen. Die erste ist die View für das Anzeigen der Model-Objekte, die zweite ist die Darstellung für den Prüfungsalarm.

Die Clientanwendung besitzt drei Views. Die IP-Liste, das Installationsfenster und das Eigenschaftsfenster.

Die **Control**-Komponente verarbeitet die Eingaben des Nutzers aus der View und verwaltet die Model-Objekte. Es kann einzelne Einträge der Model-Objekte bearbeiten und die Darstellung der View anpassen. Es wurde für jede Kategorie und für das UMS eine Control-Klasse implementiert. Die API enthält zusätzlich eine Control-Klasse für die REST-Schnittstelle. Abbildung 4.2 zeigt das oben genannte bildlich dar.

Es existiert weiterhin eine Gruppe, die in die MVC-Komponenten nicht zuzuordnen ist. Es handelt sich um Service-Klassen die komplexere Funktionen erfüllen und nicht direkt mit den MVC-Komponenten interagieren. Eines dieser Klassen übernimmt das Formatieren von Bild-Dateien zu Byte-Arrays. Für das Versenden und Bearbeiten von Bild-Dateien müssen diese in Byte-Arrays konvertiert und danach wiederhergestellt werden. Es handelt es sich um eine zusätzliche Funktionalität die unabhängig zu den Komponenten ist. Diese Erweiterungen werden im Projekt der selbsternannten Service-Komponente zugeordnet.

Die Anwendungs-Architektur bietet eine übersichtliche Aufteilung des Projektes. Kompo-

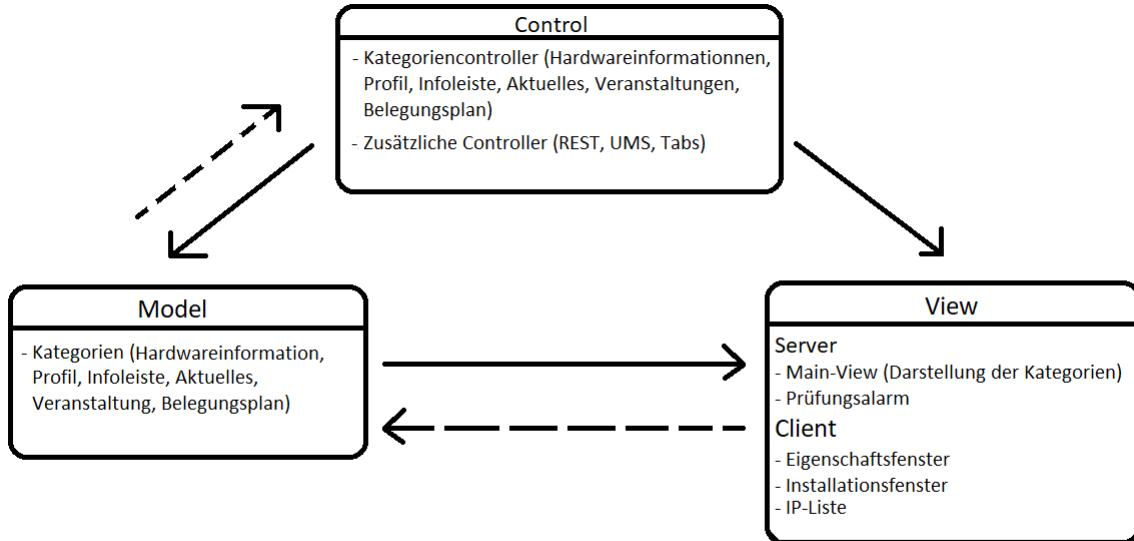


Abbildung 4.2: Architektur der Anwendungen (durchgehende Pfeile = Bearbeitet/Verändert Komponente; nicht durchgehend = beobachtet die Komponente und reagiert auf Veränderungen)

nenten können für zukünftige Projekte wiederverwendet und erweitert werden. Zudem finden sich externe Entwickler besser im Quelltext zurecht. Das ist vorteilhaft für eine zukünftige Erweiterung des Projektes.

4.2 Frameworks und Erweiterungen

Außerhalb der eigenen entwickelten Serviceklassen. Wurden für die Implementierung zahlreiche Frameworks und Erweiterungen genutzt. Bei diesem Hilfsmittel handelt es sich um frei nutzbaren Code, die die Implementierung vereinfachen und den Quelltext verständlicher machen.

Im Folgenden werden die wichtigsten Frameworks, welche in Kapitel 2 noch nicht benannt worden sind, mit Ihrer Funktion aufgelistet.

Json.NET. Json.NET von Newtonsoft konvertiert Modelobjekte in JSON Code und zurück. Das Framework ist in der Lage Arrays mit (selbsterstellten) Objekten in JSON Code zu formatieren. Der Entwickler muss sich nicht mit der Codierung beschäftigen und erhält den Code für die Kommunikation. Der Code kann im String-Format an andere Anwendungen gesendet und zurückformatiert werden. Die Formatierung in JSON und die Wiederherstellung benötigen jeweils ein Funktionsaufruf mit Objekt-Typ. Der Quellcode 4.1 zeigt ein Nutzungsbeispiel [22].

```

//-----Konvertierung-----

Product product = new Product();
product.Name = "Microsoft";
product.Expiry = new DateTime(2010, 12, 28);
product.Sizes = new string[] { "Big" };

string json = JsonConvert.SerializeObject(product);
// {
// "Name": "Microsoft",
// "Expiry": "2010-12-28T00:00:00",
// "Sizes": [
// "Big"
// ]
// }

//-----Wiederherstellung-----


Product p = JsonConvert.DeserializeObject<Product>(json);
string name = p.Name;
// Microsoft

```

Quellcode 4.1: Beispiel für die Konvertierung und Wiederherstellung eines JSON-Codes

Restup. Restup von Tom Jark erstellt einen REST-Webserver für UWP-Apps. Derzeit existiert keine einfache Möglichkeit von Microsoft einen REST-Webserver auf den IoT-Geräten zu erstellen. Microsoft bietet den Nutzer an, die IoT-Funktionalitäten durch ihr Cloud-Service (Azure) zu nutzen. Außerhalb davon ist der Entwickler gezwungen eine eigenen REST-Server zu entwickeln.

Restup ist ein Framework, welches eine REST-Schnittstelle erstellt und die Kommunikation zwischen Windows 10 IoT und Desktop-PCs vereinfacht. Das Framework erstellt einen REST-Server und bietet den Nutzer mithilfe von Annotationen an, HTTP-Methoden auszuwerten. Es werden die HTTP-Methoden GET, POST, PUT und DELETE unterstützt. Weiterhin formatiert es empfangenen JSON-Code direkt in das entsprechende Objekt für die weitere Nutzung. Als Anforderung muss der Typ des zu formatierenden Objektes benannt werden.

Der Quellcode 4.2 zeigt die Erstellung eines REST-Servers mit der Einbindung eines REST-Controllers. Dieser wurde aus dem Projekt entnommen.

```

public sealed class Server
{
    public static IAsyncAction Task_Run()
    {
        return Task_RunAsync().AsAsyncAction();
    }

    private static async Task Task_RunAsync()
    {
        // Registrierung des RestControllers
        var restRouteHandler = new RestRouteHandler();
        restRouteHandler.RegisterController
        <RestController>();

        // Starten des Webservers unter dem Port 8081
        var configuration = new HttpServerConfiguration()
            .ListenOnPort(8081)
            .RegisterRoute("api", restRouteHandler)
            .EnableCors();
        var httpServer = new HttpServer(configuration);
        await httpServer.StartServerAsync();
    }
}

// Ausfuehren der Funktion nach Aufruf der Methode durch den
// HTTP Befehl (hier:GET)
[RestController(InstanceCreationType.PerCall)]
public sealed class RestController
{
    //Device
    [UriFormat("/device/getActive")]
    public IGetResponse GetActive()
    {
        return new GetResponse(
            GetResponse.ResponseStatus.OK, HardwareController
            .GetDeviceInfo());
    }
    ...
}

```

Quellcode 4.2: Erstellung eines Servers mit der Verbindung eines REST-Controllers.

Die anderen HTTP-Methode, die nicht im Quellcode 4.2 erwähnt werden, haben eine ähnliche Struktur wie die angegebene GET Methode [16].

Ookii Dialogs. Ookii Dialogs von Sven Groot erweitern die vorhandenen Dialoge des .Net Frameworks. Mit Ihnen ist es möglich komplexere Dialoge vereinfacht darzustellen. Es beinhaltet einen Task-Dialog, der eine Nachricht mit einem Custom-Button beinhaltet, einen Progress-Dialog, der einen Fortschrittsbalken mit Text anzeigt, einen Credential-Dialog der Anmeldedaten aufnehmen und speichern kann und vieles Weitere [15]. Die oben beschriebenen Dialoge wurden auf der Client-Seite angewendet. Der Quellcode 4.3 stellt eine beispielhafte Programmierung für den Credit-Dialog dar. Abbildung 4.3 ist das Ergebnis des Quellcodes.

```
CredentialDialog crDiag = new CredentialDialog
{
    Content = "Diese Information stammen vom Windows 10 IoT Gerät und
    werden für die Kommunikation mit der API benötigt.",
    MainInstruction = "Bitte geben Sie Benutzername und Passwort für "+
        IPListe.SelectedItem.ToString() + " ein.",
    Target = "Anmeldefenster",

    ShowSaveCheckBox = true
};
```

Quellcode 4.3: Erstellung eines Servers mit der Verbindung eines REST-Controllers.

4.3 Serverseitige Implementierung

Dieser Abschnitt beschäftigt mit der Implementierungen der jeweiligen Anwendungen auf-seiten des Servers. Der Vorgang der Implementierung wird schrittweise angegangen und teilweise mit Quellcode aus dem Projekt abgeglichen.

4.3.1 Layout der Anwendung (Foreground-APP)

Der erste Schritt der Implementierung ist es eine Applikation zu schaffen, die das Darstellen von Informationen bewerkstelligt. Für diese Anforderung wird die Oberfläche entsprechend der Entwürfe aus Unterunterabschnitt 3.2.2 realisiert.

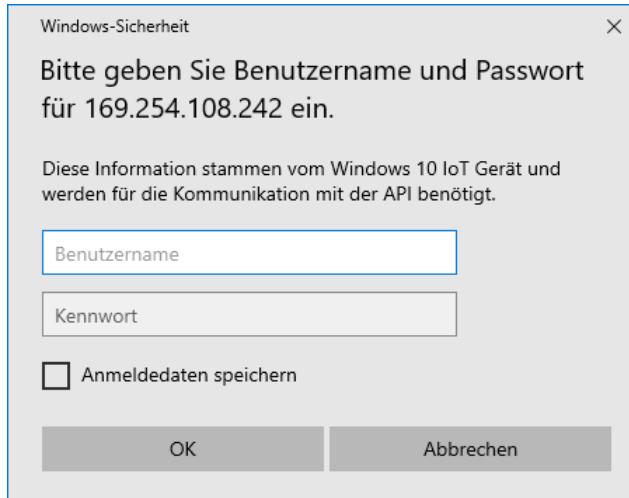


Abbildung 4.3: Ein Credential-Dialog kreiert durch die Ookii Dialog Erweiterung.

Grundgerüst

Das Grundgerüst ist eine Darstellung, die sich nicht ändert, somit kann sie fest kodiert implementiert werden. Die Implementierung des Grundgerüsts findet in der Beschreibungssprache Xaml statt. Einzelne Bereiche des Fenster werden dabei in mehrere Tabellen geteilt. Es existiert eine große Tabelle, die die ganze Oberfläche einnimmt und weitere kleinere Tabellen die die einzelnen Bereiche der Oberfläche in Ihre Funktionen aufteilt. Die Tabellen sind teilweise ineinander geschachtelt.

Die große Tabelle wird nicht aufgeteilt. Sie markiert den Anfang des Inhalts. In dieser Tabelle befindet sich die gesamte Oberfläche.

Danach folgt darin ein Stackpanel. Der Stackpanel markiert die Infoleiste ,die die oberste Fläche der Anzeige belegt. Nach dem Stackpanel folgt eine Tabelle, die die Restfläche in zwei Abschnitte aufteilt. Die linke Fläche ist die Navigationsleiste und die rechte der variable Inhalt durch die einzelnen Kategorien.

In dem Stackpanel befindet sich eine Tabelle mit einer Spalte und sechs Zeilen. Die Informationen, die in diesen Zeilen dargestellt werden, sind die Raumnummer, das Symbol der Bildungseinrichtung, der Titel des Raumes, das Symbol der Fakultät und die Bezeichnung der Fakultät. Die genannten Daten außer dem Titel des Raumes werden jeweils in einer Zeile dargestellt. Aufgrund der Länge des Titels werden für die Darstellung zwei Zeilen vereint. Die Breite der Spalte erstreckt sich über die gesamte Oberfläche. Es schneidet auch den variablen rechten Teil der Fläche. Damit es nicht zu Überlappungen der Informationen kommt, ist die jeweils erste Spalte auf beiden Seiten für die Informationsleiste reserviert. Der Quellcode 4.4 stellt die Realisierung der Infoleiste dar.

```
<StackPanel HorizontalAlignment="Left" VerticalAlignment="Top">
```

```
<Grid Grid.Row="0" Background="DarkBlue" Width="800" Height="70">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="3*"/>
        <ColumnDefinition Width="3*"/>
        <ColumnDefinition Width="3*"/>
        <ColumnDefinition Width="3*"/>
        <ColumnDefinition Width="3*"/>
        <ColumnDefinition Width="2*"/>
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
        <RowDefinition Height="3*"/>
    </Grid.RowDefinitions>

    <TextBlock Name="Room" Text="R 2.018" FontSize="20"
        Grid.Column="0" VerticalAlignment="Center"
        HorizontalAlignment="Center"/>
    <Image Name="LeftImage"
        Source="ms-appx://HelloWorld/Assets/TuLogo.png"
        Height="30" Grid.Column="1" VerticalAlignment="Center"
        HorizontalAlignment="Center"/>
    <TextBlock Name="Title" Text="Dr. Lars Hildebrand"
        FontSize="20"
        Grid.Column="2" Grid.ColumnSpan="2"
        VerticalAlignment="Center" HorizontalAlignment="Center"/>
    <Image Name="RightImage"
        Source="ms-appx://HelloWorld/Assets/tu_fk04.png"
        Height="30" Grid.Column="4" VerticalAlignment="Center"
        HorizontalAlignment="Center"/>
    <TextBlock Name="Chair" Text="LS14" FontSize="15"
        Grid.Column="5"
        VerticalAlignment="Center" HorizontalAlignment="Center"/>

</Grid>

</StackPanel>
```

Quellcode 4.4: Code für die Darstellung der Infoleiste.

Die Navigationsleiste ist eine Tabelle mit einer Zeile und mehreren Spalten (Anzahl der Kategorien). Die Länge der Tabelle erstreckt sich über die komplette linke Fläche des Stackpanels. Der Quellcode ist ähnlich der Infoleiste mit einer Ausnahme. Für die Hintergrundfärbung der Navigationsbutton wird zusätzlich zum Textblock ein Border benötigt. Dadurch kann der Button beim Öffnen des jeweiligen variablen Inhalts farblich markiert werden.

```
...
<Border Name="ProfilBorder" Grid.Row="0" BorderBrush="#FF179AC8"
BorderThickness="0,0,2,2" />
<TextBlock Text="Profil" Padding="31" Grid.Row="0"
VerticalAlignment="Center" HorizontalAlignment="Center"/>
...
```

Quellcode 4.5: Darstellung des Textblocks mit einem Border.

Die Abbildung 4.4 zeigt die Ausgabe der Implementierung für die Infoleiste und der Navigationsleiste.



Abbildung 4.4: Darstellung des Grundgerüsts ohne variablen Inhalt.

Für das Ein-/ und Ausschalten der einzelnen Kategorien existiert ein Array mit booleschen Werten. Falls der Wert für eine Kategorie falsch ist, wird die Kategorie gräulich gefärbt und bei der Zeitschaltung ignoriert. Die lokal gespeicherten Informationen, die für die Kategorie

vorhanden sind, bleiben dennoch erhalten. Nur der Aufbau des visuellen variablen Inhalts fällt weg.

Variabler Inhalt

Die Darstellung des Inhalts ist abhängig von der markierten Kategorie. Da die Anzeige nicht permanent ist, muss der Quellcode generiert werden. Die Erstellung des Quellcodes findet in C# statt. Die Darstellungen der Kategorie erfordern unterschiedliche Layouts. Aufgrund der starken Ähnlichkeiten der verschiedenen Layouts werden nur zwei Implementierung von Kategorien näher erläutert.

Normalfall. Für die Visualisierung einer Kategorie müssen verschiedenen Objekte erstellt werden. Benötigt werden Image Objekte für die Darstellung von Bildern und mehrere Textblöcke für das Anzeigen der Informationen. Die Komponenten werden in eine selbst erstellte Tabelle eingefügt. Zum Schluss wird die gefüllte Tabelle in das Grundgerüst eingebettet. Die Anordnung der Textblöcke ist je nach Kategorie unterschiedlich. Die Information die dargestellten werden, werden ausschließlich aus dem lokalen Speicher der Anwendung verwendet. Bei Veränderungen werden die lokalen Speicherdateien überschrieben, sodass beim nächsten Aufbauen der Komponenten die neuen Daten verwendet werden.

Belegungsplan. Ein Spezialfall stellt die Realisierung des Belegungsplans dar. Der Belegungsplan hat zusätzliche Funktionen, wie das Markieren der aktuellen Uhrzeit und die farbliche Hervorhebung von belegten Zeiträumen auf den Plan. Für den Belegungsplan werden zwei zweidimensionale Felder genutzt. Das erste Feld beinhaltet die Tage und die Uhrzeiten auf dem Belegungsplan. Das zweite Feld befindet sich innerhalb des Ersten und beinhaltet die Veranstaltungen. Jedes dieser Felder hat einen Textblock und ein Border. Der Textblock enthält den Namen der Veranstaltung. Die Border hebt die Veranstaltung aus dem Belegungsplan hervor. Die Hervorhebung geschieht durch die Färbung des Feldes. Wenn die Veranstaltung auf der gleichen Spalte über mehrere Zeilen geht, wird die Gesamtfläche zu einer Fläche vereint. Dies komprimiert die Informationen auf den Belegungsplan und schafft Übersicht. Für das Markieren der aktuell laufenden Veranstaltung auf dem Belegungsplan wird der gespeicherte Wochentag mit Uhrzeit aus dem Betriebssystem abgefragt und auf den Zeitplan angewendet. Bei einer zeitlichen Überschneidung wird die entsprechende Fläche des Belegungsplans dunkelblau markiert. Abbildung 4.5 zeigt das Ergebnis der Implementierung.

Für die Zeitschaltung des variablen Inhalts existiert ein Ladebalken der in 13 Sekunden die nächste aktive Kategorie öffnet. In der Implementierung wird ein Event-Handler genutzt, der in Millisekunden den Ladebalken füllt. Wenn der Ladebalken voll ist, wird die nächste aktive Kategorie geöffnet und der Inhalt des Balkens wieder geleert. Sollte es keine weitere aktive Kategorie mehr geben, begibt sich der Zyklus zurück in die Ausgangsposition und

Profil		Montag	Dienstag	Mittwoch	Donnerstag	Freitag
Aktuelles	08:00 - 10:00					
Veranstaltung	10:00 - 12:00	Veranstaltung	Veranstaltung	Veranstaltung	Veranstaltung	Veranstaltung
Belegungsplan	12:00 - 14:00					
Wetter	14:00 - 16:00	Veranstaltung	Veranstaltung	Veranstaltung	Veranstaltung	Veranstaltung
	16:00 - 18:00					
	18:00 - 20:00					

Abbildung 4.5: Endfassung des Belegungsplanes.

beginnt von vorn. Der Quellcode 4.6 zeigt die Implementierung und Einbindung des Event-Handlers.

```

...
DispatcherTimer Timer = new DispatcherTimer();

public MainPage()
{
    ...
    //Bind Method to Eventhandler
    Timer.Tick += Timer_Tick;

    //Calculate interval
    double IntervalinSec = 10;
    int FinalInterval = (int)(IntervalinSec * 1000) /
        (int)progressBar1.Maximum;

    //Set interval
    Timer.Interval = new TimeSpan(0, 0, 0, 0, FinalInterval);
    Timer.Start();
    ...
}
```

```

}

...
//Timer
private void Timer_Tick(object sender, object e)
{
    //24 hour time
    CultureInfo.CurrentCulture = new CultureInfo("de-DE");

    //If progressbar not full, fill it
    if (progressBar1.Value < progressBar1.Maximum)
    {
        progressBar1.Value += 1;
    }

    //If progressbar full, go to next navigationtab
    else if (NavPosition < 4)
    {
        progressBar1.Value = 0;
        NavPosition += 1;
        OpenNavPage(NavPosition);
    }

    //If end of navigation, go to start
    else
    {
        progressBar1.Value = 0;
        NavPosition = 0;
        OpenNavPage(NavPosition);
    }
}

```

Quellcode 4.6: Kommentierte Implementierung eines Handlers (Hier Timehandler).

Wetter. Die Kategorie Wetter wurde zusätzlich implementiert und soll vor allem zur Veranschaulichung für die Entwicklungen weiterer Kategorien dienen. Die gesamte Implementierung wurde in der Foreground-App bewerkstelligt

Für die Realisierung der Kategorie ist es notwendig eine Kommunikation mit einem Wetterdienst zu bewerkstelligen. Dieser muss durch eine Programmierschnittstelle ansprechbar sein, um die Daten auf dem IoT-Gerät verwerten zu können. Der Online Dienst OpenWeatherMap.org ist ein Wetterdienst der solch eine Programmierschnittstelle anbietet.

Zuerst wird das variable Layout erstellt. Die Implementierung des Layouts geschieht dynamisch in C#. Es wird ein 5x5 Grid erstellt, der das Wetter für vier Tage voraussagt. Jede Zeile enthält das Icon, den Tag und die Mindest- und Höchsttemperatur. Die restlichen Felder dienen zur Verbesserung der Optik. Die notwendigen Informationen werden vom Online-Dienst entnommen.

Für die Kommunikation bietet OpenWeatherMap eine Programmierschnittstelle an, die alle Informationen anhand der derzeitigen Lokalisation und einem API-Key ausgibt.

Für die Lokalisationen müssen Berechtigung vom Nutzer erbittet werden. Dies geschieht durch das Hinzufügen des Quellcodes 4.7 in das Package.appxmanifest der einzelnen Anwendungen.

```
<Extension Category="windows.backgroundTasks"
EntryPoint="BackgroundApplication1.StartupTask">
    <BackgroundTasks>
        <Task Type="general" />
        <Task Type="systemEvent" />
        <Task Type="location" />
        <Task Type="deviceUse" />
    </BackgroundTasks>
</Extension>
```

Quellcode 4.7: Notwendige Berechtigungen für die OpenWeatherMap Kategorie.

Nach der erfolgreichen Vergabe der Berechtigung ist es möglich, den internen Geolocator (Quellcode 4.8) anzusprechen, der die derzeitige des IoT-Geräts an die Anwendungen weiterleitet.

```
public async static Task<Geoposition> GetPosition()
{
    // Ist die Berechtigung erteilt?
    var accessStatus = await Geolocator.RequestAccessAsync();
    if (accessStatus != GeolocationAccessStatus.Allowed)
        throw new Exception();
    // Genauigkeit der Lokalisation
    var geolocator = new Geolocator { DesiredAccuracyInMeters = 0 };
    // Ermittlung der Position
    var position = await geolocator.GetGeopositionAsync();
    return position;
}
```

Quellcode 4.8: Zugriff auf den Geolocator.

Der API-Key wird durch eine Registration auf dem Online-Dienst erworben. Dieser kann durch die Schnittstelle nicht generiert werden, sondern wird manuell in die Implementierung eingetragen. Für den Empfang der Informationen von der OpenWeatherMap Seite muss eine entsprechende Model-Klasse erstellt werden, die die ankommenden Informationen verarbeitet. Aufgrund der Anzahl der Attribute und Parameter ist eine manuelle Implementierung der Klassen sehr aufwendig. Mit dem Online-Tool Json2csharp von Jonathan Keith ist es möglich eine C# Model-Klasse durch eine JSON-Datei automatisiert zu erstellen [17].

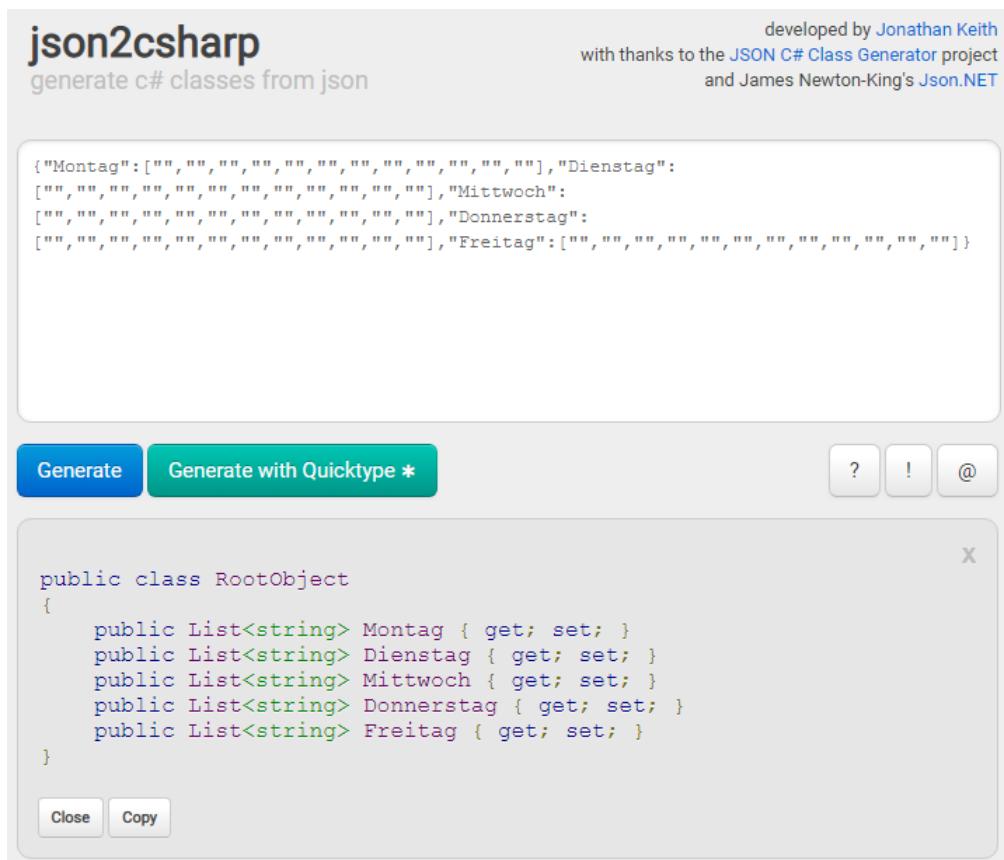


Abbildung 4.6: Benutzung des Online-Tools Json2csharp [17].

Nachdem die Informationen nun empfangen werden können, müssen diese an die Komponenten vom Layout gebunden werden. Dies geschieht durch die Namensgebung der einzelnen Komponenten. Abbildung 4.7 zeigt das Ergebnis der Implementierung.

4.3.2 REST-Schnittstelle (Background-APP)

Die REST-Schnittstelle ist eine Hintergrundapplikation. Der Nutzer merkt nichts von der Anwesenheit der Applikation. Es ermöglicht eine Kommunikation zwischen der Foreground-

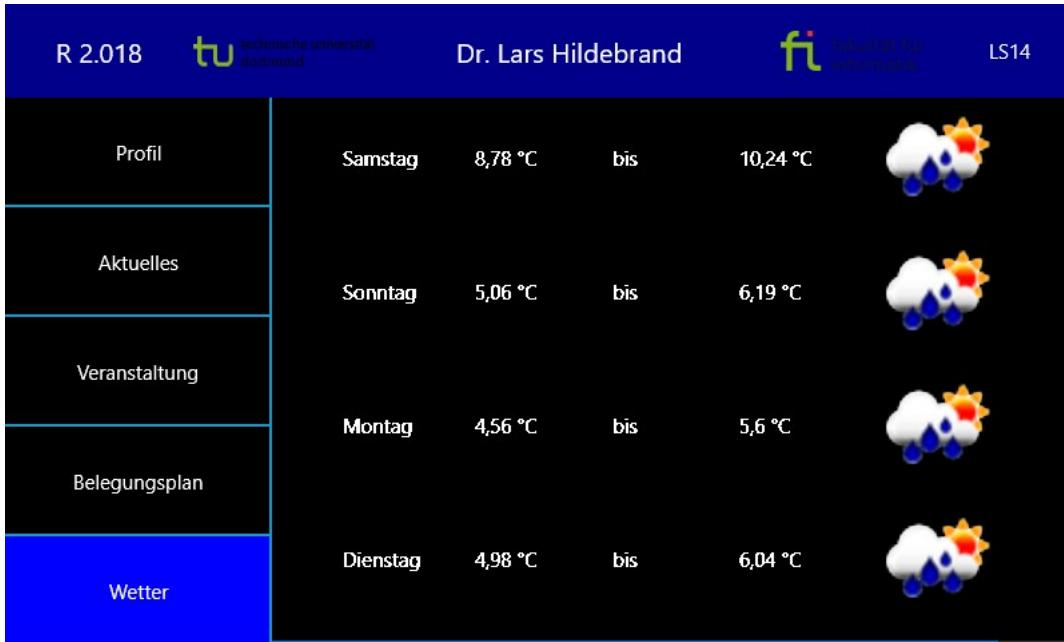


Abbildung 4.7: Endfassung der Kategorie Wetter.

APP (siehe Unterabschnitt 4.3.1) und der Client-APP (Desktop) (siehe Abschnitt 4.4). Bei der Implementierung wurden die sechs Prinzipien, die in Abschnitt 2.4.2 erläutert worden sind, berücksichtigt. In diesem Unterabschnitt wird die genaue Funktionsweise der Schnittstelle erläutert.

Die REST-Schnittstelle wird mit dem Framework Restup erstellt (siehe Abschnitt 4.2). Es wird ein zusätzlicher Server geschaltet der auf den Port 8001 aufgerufen werden kann. Je nach HTTP-Methode werden die vorhandenen Daten aufgerufen oder ersetzt. Folgende Dokumentation der REST-Schnittstelle wurde mithilfe des Open-Source-Software-Frameworks SwaggerHub erstellt [27].

Nutzerschnittstelle zur eigenen API

Gerät

GET	/device/getActive Ist die App aktiv?	Gibt ein Zustand Objekt aus.
GET	/device/getNetwork Gib die Netzwerkinformationen wieder.	Gibt ein HardwareInfo Objekt aus.
GET	/device/getLogin Gib die Nutzerdaten wieder.	Gibt ein LoginDaten Objekt aus.
PUT	/device/setActive De/aktiviere App ?	Benötigt ein Zustand Objekt.
PUT	/device/setLogin Setze die Nutzerdaten	Benötigt ein LoginDaten Objekt.

Tabs

GET	/tabs/getActivated Welche Tabs sind aktiv?	Gibt ein Tab Objekt aus.
PUT	/tabs/setActivated Editiert die aktiven Tabulatoren.	Benötigt ein Tab Objekt.

Infoliste

GET	/infoleiste/getLeiste Gibt die derzeitigen Dateien zur Infoliste aus.	Gibt ein Infoliste Objekt aus.
PUT	/infoleiste/setLeiste Editiert die Dateien der Informationsleiste	Benötigt ein Infoliste Objekt.

Profil

GET	/allgemein/getProfil Gibt die derzeitigen Profile wieder.	Gibt ein Profil Objekt aus.
PUT	/allgemein/setProfil Editiert die derzeitigen Profil.	Benötigt ein Profil Objekt.

Aktuelles

GET

/aktuelles/getAktuellesText Gibt den Text in der Kategorie Aktuelles aus.

Gibt ein **String** aus.

PUT

/aktuelles/setAktuellesText Ediert den Text in der Kategorie Aktuelles.

Benötigt einen **String**.

Veranstaltung

GET

/veranstaltung/getAktuell Gibt die aktuellen Veranstaltungen aus.

Gibt ein **String** aus.

PUT

/veranstaltung/setAktuell Ediert die aktuellen Veranstaltungen.

Benötigt einen **String**.

Belegungsplan

GET

/belegungsplan/getBelegungsplan Gibt den Belegungsplan aus.

Gibt ein **Belegungsplan** Objekt aus.

PUT

/belegungsplan/setBelegungsplan Ediert den Belegungsplan.

Benötigt ein **Belegungsplan** Objekt.

Wetter

GET

/wetter/getOpenWeather Ruft den API-Key für OpenWeatherMap ab.

Gibt ein **OpenWeather** Objekt aus.

PUT

/wetter/setOpenWeather Setzt den API-Key für OpenWeatherMap.

Benötigt ein **OpenWeather** Objekt.

Prüfung

POST

/pruefung/setOpenWeather De-/Aktiviert die Prüfungs-APP.

Benötigt ein **Zustand** Objekt.

```
Zustand {
    active      boolean
}
```

```

HardwareInfo ▼ {
    name          string
    hardwareId   string
    network       string
    ip            string
    version       string
    ipv6          boolean
    ipv4          boolean

}

Tabs ▼ {
    activeTabs     ▼ [
        maxItems: 5
        minItems: 5
        boolean]
}

}

```

```

Infoliste ▼ {
    title          string
    room           string
    chair          string
    leftLogo       ▼ [string($byte)]
    rightLogo      ▼ [string($byte)]

}

}

```

```

Profil ▼ {
    name          string
    surname       string
    role          string
    street        string
    place         string
    country       string
                nullable: true
    placenumber   integer($int32)
    title         string
    phone         string
                nullable: true
    fax           string
                nullable: true
    email         string
                nullable: true
    image         ▼ [string($byte)]

}

}

```

Die Dokumentation zeigt die verfügbaren Methoden mit der abhängigen Teil-URL. Bei Aufruf der URL mit der entsprechenden HTTP-Methode wird die entsprechende Funktion ausgeführt. Die notwendigen Model-Objekte bei der Ausführung sind im Punkt Schema näher beschrieben. Empfangen und gesendet werden die Dateien im JSON-Format. Die

```
Belegungsplan ▼ {
    belegungsplan
        ▼ [
            maxItems: 12
            minItems: 12
            ▼ [
                maxItems: 5
                minItems: 5
                string]]
}
}
```

```
OpenWeather ▼ {
    API
        string
}
}
```

```
LoginDaten ▼ {
    Username
        string
    Password
        string
}
}
```

De-/Kodierung geschieht durch das Framework Json.net (siehe Unterabschnitt 4.2).

Die URL stellt sich aus der IP des Gerätes mit dem Port 8081, der entsprechenden HTTP-Methode und der in der Dokumentation genannten Teil-URL zusammen. Nachdem Aufruf einer URL mit der HTTP-Methode PUT wird die Nachricht zunächst lokal in der Applikation gespeichert und danach durch das UWP-MessageRelay an die Foreground-APP weitergeleitet. Bei dem Aufruf einer GET-Methode werden die Daten aus dem lokalen Speicher der API weitergeleitet. Ein zusätzlicher Kommunikationsweg von Foreground-APP zu Background-APP wird dadurch unnötig.

4.3.3 Nachrichtenkanal (UWP-MessageRelay)

Der UWP-MessageRelay wird für den Informationsaustausch zwischen der Foreground-APP und der Background-APP benötigt. Für einen Nachrichtenaustausch wird jeweils ein Nachrichtenkanal zwischen dem UWP-MessageRelay und den Anwendungen erstellt. Beim Eingang einer Nachricht versendet es diese durch die Nachrichtenkanäle an die angebundenen UWP-Applikationen weiter (Broadcast). Die Grundstruktur der Applikation stammt vom Projekt **UwpMessageRelay** von Lee Richardson. Veröffentlicht wurde es mit der MIT-Lizenz, welches eine kostenlose Verwendung und Veränderung ohne Auflagen ermöglicht [24].

Die Applikation ist eine Headless-Applikation. Das heißt, sie besitzt keine Benutzerschnittstelle. Die APP nimmt keine Eingaben vom Nutzer an und gibt keine Ergebnisse an den

Nutzer aus. Es handelt sich hier um eine reine App-to-App Kommunikation. Eine Headless-Applikation ist eine spezielle Art der Hintergrundapplikation.

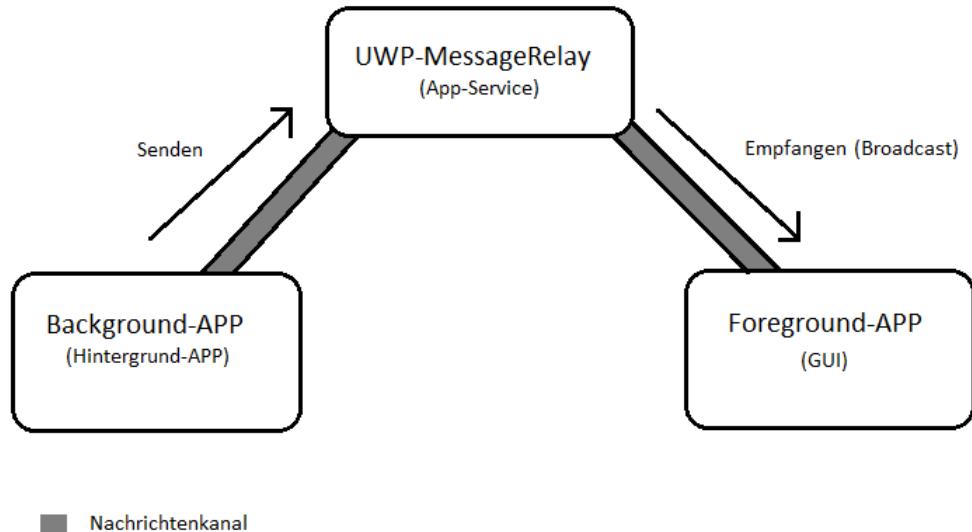


Abbildung 4.8: Grafische Darstellung der Funktionsweise vom UWP-MessageRelay.

Der Belegungsplan besteht aus einem zweidimensionalen 12 (Zeiten) x 5 (Tage) String-Array.

Microsoft kategorisiert Headless-Hintergrundapplikationen mit den TAG **AppService**. Für die Änderung der Kategorie ist es notwendig die Package.appxmanifest, wie in Quellcode 4.9 dargestellt, zu verändern.

```
...
<Extensions>
    <uap:Extension Category="windows.appService"
        EntryPoint="UwpMessageRelay.StartupTask">
        <uap:AppService Name="UwpMessageRelayService" />
    </uap:Extension>
</Extensions>
...
```

Quellcode 4.9: Änderung des Package.appxmanifests zur Kategorie AppService.

Zusätzlich zu der Kategorie der Applikation wird die Start-Klasse (hier:UwpMessageRelay.StartupTask) und der Name der Applikation (hier:UwpMessageRelayService) festgelegt. Die Namensgebung der Anwendung ist notwendig für den Aufruf durch andere Applikationen.

Die Funktion der Start-Klasse wird jedes Mal aufgerufen, wenn der Client (hier:Background-APP) einen Nachrichtenkanal zum Server (hier:Foregorund-APP) erstellen möchte. Bei der Erstellung werden statische Variablen an die Applikation mitversendet. Diese statischen Variablen ermöglichen es eindeutige applikationsgebundene Nachrichtenkanäle zwischen den Applikationen zu erstellen. Diese Variablen werden in eine statische Liste mit einer einzigartigen ID gespeichert [24].

Die Startup-Klasse besitzt eine Run-Methode, die nach jedem Aufruf ausgeführt wird. In dieser Run-Methode wird der Kanal zwischen Server und Client geschaffen und mit Handlern versehen. Quelltext 4.10 zeigt die Realisierung der Methode.

```
public sealed class StartupTask : IBackgroundTask
{
    ...
    public async void Run(IBackgroundTaskInstance taskInstance)
    {
        ...
        // Verhindert das Terminieren der Anwendung nach der
        Ausfuehrung.
        _backgroundTaskDeferral = taskInstance.GetDeferral();
        // Speichert fuer jede Verbindung eine einzigartige ID
        _thisConnectionGuid = Guid.NewGuid();

        // Verschafft Zugang zum Eingang und Ausgang des
        Nachrichtenkanals
        var triggerDetails = taskInstance.TriggerDetails as
            AppServiceTriggerDetails;
        var connection = triggerDetails?.AppServiceConnection;
        ...
        // Belegt die Handler fuer die Verbindung
        connection.RequestReceived += ConnectionRequestReceived;
        ...
    }
    ...
}
```

Quellcode 4.10: Run-Methode der Startup-Klasse.

Wenn der Kanal eine Nachricht empfängt, sendet es an alle in der statischen Liste enthaltenen Kanäle die Nachricht weiter.

Für die Verbindung zwischen den Anwendungen ist es notwendig eine weitere Klasse je-

weils für den Server und Client zu erstellen, die das UWP-MessageRelay anspricht und Nachrichten sendet bzw. empfängt.

Client - Senden von Dateien

Die Client-Variante hat die Aufgabe Daten zu versenden. Es muss eine Verbindung mit dem UWP-Message-Relay eingehen und kann danach den Nachrichtenkanal mit Informationen versorgen.

Zunächst wird eine Instanz des UWP-Message-Relays auf der Clientseite erstellt. Falls bereits eine Instanz vorhanden ist, wird diese stattdessen aufgerufen. Danach folgt die Suche des UWP-Message-Relays im internen App-Service Katalog der Projektmappe. Gesucht wird durch den Namen des App-Services (hier: **UwpMessageRelayService**). Dieser Schritt ist notwendig, damit das UWP-Message-Relay die notwendige Berechtigung bekommt, Dateien an andere Applikationen weiterzuleiten. Nach einem erfolgreichen Fund der Applikation wird ein Verbindungskanal geöffnet und gespeichert. Wenn die Verbindung steht, können Nachrichten als Paare von String und Objekt gesendet werden. Quellcode 4.11 zeigt einen Teil des kommentierten Codes für die Realisierung der Theorie.

```
...
const string AppServiceName = "UwpMessageRelayService";
private AppServiceConnection _connection;
...

public async Task<AppServiceConnection> MakeConnection()
{
    //Suche nach der App-Service
    var listing = await
        AppServiceCatalog.FindAppServiceProvidersAsync
            (AppServiceName);

    ...

    // Erstelle eine Verbindung mit der App-Service-Applikation
    var packageName = listing[0].PackageFamilyName;
    var connection = new AppServiceConnection
    {
        AppServiceName = AppServiceName,
        PackageFamilyName = packageName
    };

    var status = await connection.OpenAsync();
```

```

    ...

        return connection;
    }

    ...

    public async Task SendMessageAsync(KeyValuePair<string, object>
keyValuePair)
{
    var connection = await CachedConnection();
    // Sende die Nachricht durch den Nachrichtenkanal
    var result = await connection.SendMessageAsync(new ValueSet {
        keyValuePair });

    if (result.Status == AppServiceResponseStatus.Success)
    {
        return;
    }
    throw new Exception("Error sending " + result.Status);
}

...
}

```

Quellcode 4.11: Run-Methode der Startup-Klasse.

Der erste Wert des Paars beinhaltet die Kategorie (Profil, Aktuelles, ...) und der zweite Wert die JSON Konstruktion des entsprechenden Objekts (String).

Server - Empfangen von Dateien

Die Server-Variante soll die Daten vom Client empfangen und bearbeiten. Nach der Instantiierung bzw. das Aufrufen der vorhandenen Instanz, muss ebenfalls eine Verbindung mit dem Nachrichtenkanal eingegangen werden. Beim Erhalt einer Nachricht wird ein Handler mit einer Funktion ausgelöst, die die Nachricht nach der Kategorie (Profil, Aktuelles, ...) sortiert und das JSON-Konstrukt zur Validierung erst decodiert und sie dann intern speichert. Quellcode 4.12 veranschaulicht das Ganze.

```

private async void ConnectionOnMessageReceived(ValueSet valueSet)
{
    //Asynchroner normal-priorisierter Kontextwechsel zum Erhalt der
    //gesendeten Nachricht
    await Dispatcher.RunAsync

```

```
(Windows.UI.Core.CoreDispatcherPriority.Normal, async () =>
{
    var message = valueSet.First().Value;
    var key = valueSet.First().Key;

    //Sortierung der Nachricht nach Kategorie
    switch (key)
    {
        case "Aktuelles":
            //Validierung durch Dekodierung
            Aktuelles aktuelles =
                AktuellesController.CreateModelFromJson
                ((string)message);
            //Speicherung des Model-Objekts
            SaveController.WriteToJsonFile
                ("Aktuelles.txt", aktuelles);
            return;
        case "Profil":
            //Validierung durch Dekodierung
            Profil[] profil =
                ProfilController.CreateModelFromJson
                ((string)message);
            //Speicherung des Model-Objekts
            SaveController.WriteProfilArrayToJsonFile
                ("Profil.txt", profil);
            return;
        ...
        default:
            return;
    }
});
```

Quellcode 4.12: Handler für das Empfangen von Nachrichten.

Die intern gespeicherten Daten werden, wie in Unterabschnitt 4.3.1 erläutert und auf der View dargestellt.

4.3.4 Prüfungsapplikation (ForeGround-Prufung)

Die Prüfungsapplikation ist eine einfache View, die einen roten Hintergrund mit weißer Schrift darstellt. Es dient als Overlay für Nachrichten, die eine besondere Priorität haben. Weiterhin ist es eine eigene Anwendung, die den Betriebsablauf der anderen Applikation nicht stört. Diese Anwendung kann zukünftig mit anderen Layouts erweitert werden. Die Abbildung 4.9 zeigt das derzeit einzige implementierte Layout der Anwendung.



Abbildung 4.9: Layout der ForeGround-Prufung Applikation.

4.4 Clientseitige Implementierung

Hier wird die clientseitige Implementierung erläutert. Es wird auf das Layout, auf die Implementierung und auf die Validierung eingegangen. Die Erläuterung wird teilweise durch Quellcode und Schaubilder veranschaulicht. Realisiert wurde das Layout durch die grafische Sprache Xaml (siehe Abschnitt 2.5.1). Beim Design des Layouts wurde die Echtbilddarstellung von Microsoft Visual 2017 benutzt (siehe Unterabschnitt 2.6.2).

4.4.1 Layout der Anwendung

Auf der Seite des Clients existieren mehrere Views mit unterschiedlichen Oberflächen.

- Die IP-Liste, die für die Verbindung mit dem IoT-Gerät zuständig ist.
- Das Eigenschaftsfenster, welches die Eigenschaften der Kategorien für das entsprechende IoT-Gerät abfragt und sie verändern kann.
- Und das Progress-Fenster, das die IoT-Anwendungen auf dem IoT-Gerät installiert.

Zudem kommen noch mehrere Dialogfenster, die durch bestimmte Nutzereingaben geöffnet werden. Das Layout wurde in Bezug auf die Entwürfe aus Unterunterabschnitt 3.2.2 wie folgt realisiert.

IP-Liste

Die IP-Liste besteht aus zwei Listen mit vier Schaltflächen und einem Eingabefeld. Es ist das Startfenster der Client-Anwendung.

Das Feld dient zur Eingabe von IPv4-Adressen. Es unterteilt sich in vier separate Felder, welche mit einem Punkt (.) voneinander getrennt werden.

Die vier Schaltflächen sind notwendig für das Hinzufügen, Entfernen, Einrichten und Prüfen von Verbindungen. Nachdem Klicken auf die Schaltfläche Entfernen und Hinzufügen soll das Layout durch Veränderungen der oben genannten Listen reagieren. Beispielsweise wird nach dem Druck auf die Schaltfläche Hinzufügen die gültige IP-Adresse aus dem Eingabefeld in die erste Liste hinzugefügt, während in der zweiten Liste dessen Verfügbarkeit durch die Signalbegriffe (online, offline und checking) angezeigt wird. Die Signalbegriffe werden zusätzlich farblich markiert, um eine bessere Optik zu gewährleisten. Die Realisierung der festen Oberfläche wird in Abbildung 4.10 angezeigt.

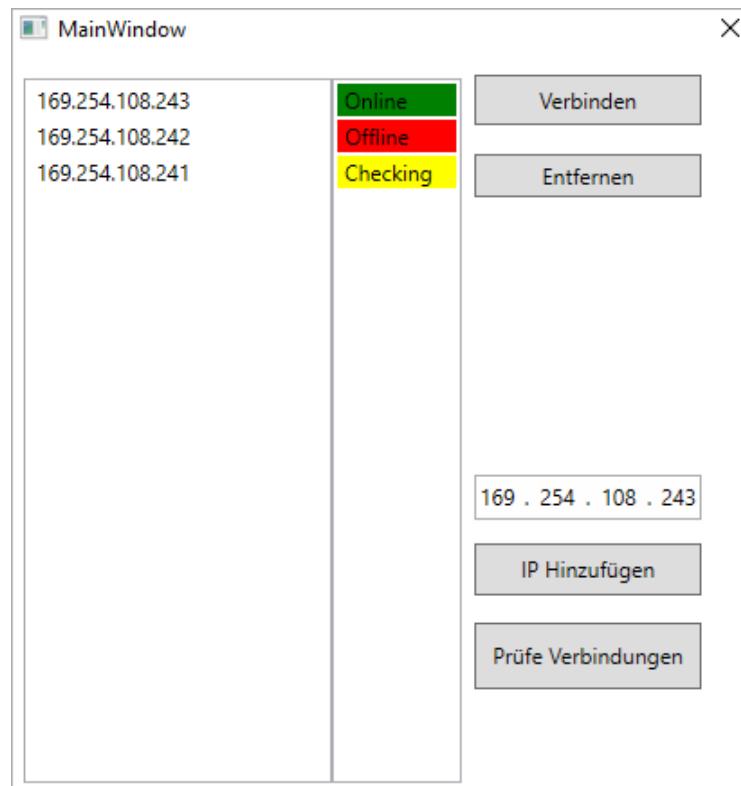


Abbildung 4.10: Layout des IP-Liste Fensters.

Beim Klick auf Verbinden wird ein zusätzliches Dialogfenster geöffnet, die die Anmelde-daten durch ein Textfeld und ein Passwortfeld für das Gerät abfragt. Zusätzlich existiert eine Checkbox für das Abspeichern der Anmelde-daten. Das Dialogfenster wurde bereits mit Quellcode im Abschnitt 4.2 (Ookii Credential-Dialog) erläutert.

Eigenschaftsfenster

Das Eigenschaftsfenster ist ähnlich aufgebaut wie das Layout der IoT-Anwendung (siehe Unterabschnitt 4.3.1). Es besitzt ein Grundgerüst mit einem variablen Teil, der die Eigenarten der einzelnen Kategorien darstellt.

Das Grundgerüst der Anwendung besteht aus einer Tab-Navigation und drei Schaltflächen zum Aktualisieren und Speichern von Daten und zum Öffnen des Prüfungsmodus. Die Schaltflächen befinden sich im unteren Teil des Fensters. Den Rest belegt die Tab-Navigation. Der Inhalt der einzelnen Kategorien wird hier abgebildet. Zusätzlich beinhaltet der Titel des Fensters die IP-Adresse mit der dazugehörigen Verfügbarkeit (Online, ...).

Aufgrund der Tab-Navigation müssen keine Komponenten gelöscht und neu konstruiert werden, wie bei der serverseitigen Implementierung (siehe Unterabschnitt 4.3.1). Die Oberflächen für das Grundgerüst und von vielen Kategorien können mit der Echtbilddarstellung erstellt werden. Zu den Kategorien gehören Gerät, Profil, Aktuelles, Veranstaltungen und Wetter. Die eingegebenen Informationen können durch die Namensgebung der einzelnen Komponenten in Xaml abgerufen und verarbeitet werden.

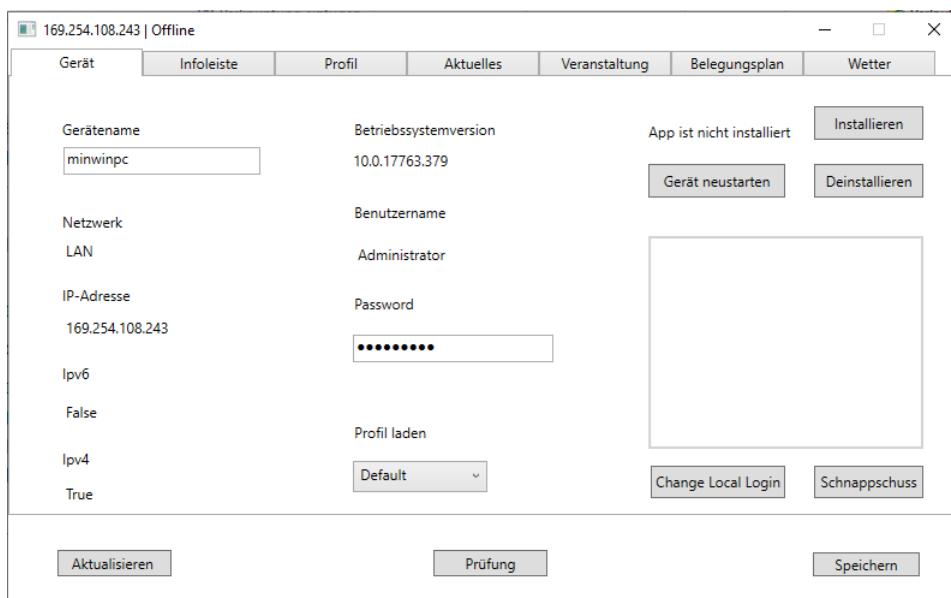


Abbildung 4.11: Layout des Eigenschaftsfensters mit dem Tabinhalt Gerät.

Die Kategorien ähneln stark den Entwürfen aus Unterabschnitt 3.2.2 und bestehen aus einfachen Komponenten. Die Abbildung 4.11 zeigt die Realisierung einer Kategorie mit dem Grundgerüst.

Der Belegungsplan besitzt eine zusätzliche Funktion, die nicht ausschließlich in Xaml dargestellt werden kann. Sie hat eine Tabelle, die nach jedem Eintrag aktualisiert wird. Die eingetragene Veranstaltung wird farblich in den entsprechenden Zeitraum in der Tabelle eingetragen. Nach einem Klick auf diesen Eintrag wird ein Handler ausgelöst, der die

Hintergrundfärbung des Eintrags ändert und es markiert. Nach einem Klick auf die Schaltfläche Entfernen wird der Eintrag gelöscht. Abbildung 4.12 veranschaulicht die Änderung der Hintergrundfarbe nach dem Klick auf die Veranstaltung.

	Montag
8:00 - 10:00	
10:00 - 12:00	Mafi I
12:00 - 14:00	
14:00 - 16:00	
16:00 - 18:00	
18:00 - 20:00	

	Montag
8:00 - 10:00	
10:00 - 12:00	Mafi I
12:00 - 14:00	
14:00 - 16:00	
16:00 - 18:00	
18:00 - 20:00	

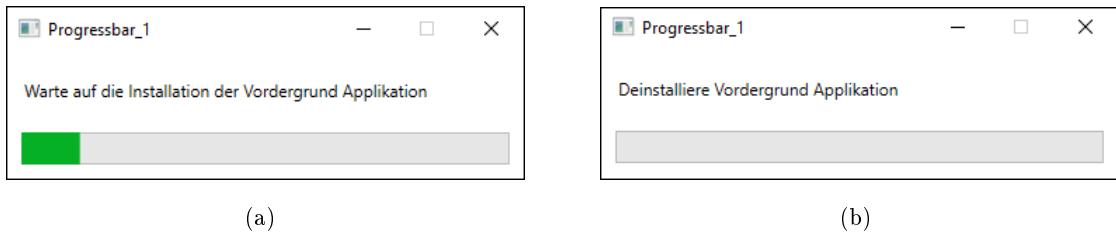
(a)

(b)

Abbildung 4.12: a) Nicht markierte Veranstaltung im Belegungsplan; b) Markierte Veranstaltung im Belegungsplan

Progress-Fenster

Das Progress-Fenster dient zur De-/Installation der IoT-Anwendung. Es wird ein einfacher Fortschrittsbalken angezeigt, der sich nach jeder Operation zum Teil füllt. Der Status der De-/Installation wird über dem Fortschrittsbalken bis zum Abschluss angezeigt. Abbildung 4.13 zeigt das entsprechende Layout des Fensters.



(a)

(b)

Abbildung 4.13: a) Progress-Fenster für die Installation; b) Progress-Fenster für die Deinstallation

4.4.2 Implementierung der Anwendung

Dieser Unterabschnitt beschäftigt sich mit der implementierten Logik der Anwendung. Es werden besondere Funktionen erläutert und originalem Quellcode belegt. Aufgrund des Umfanges wird nicht die komplette Implementierung behandelt, sondern nur Besonderheit wie die Kommunikation zu dem IoT-Gerät.

Unterteilt wird die Logik in Bezug zu den vorhandenen Fenstern, wie in Unterabschnitt 4.4.1.

IP-Liste

Die IP-Liste ist das Startfenster der Anwendung. Der Nutzer kann IP-Adressen löschen, hinzufügen und prüfen. Zusätzlich kann er durch die IP-Adresse eine Verbindung mit dem IoT-Gerät eingehen.

Es existiert ein Listenobjekt, welches alle hinzugefügten IP-Adressen beinhaltet. Nach jedem Hinzufügen/Löschen einer IP-Adresse wird die Liste unabhängig vom Nutzer im Hintergrund lokal gespeichert, sodass beim Neustarten der Anwendung die eingetragenen IP-Adressen nicht verloren gehen.

Nach dem Hinzufügen einer IP-Adresse, wird dieser angepingt. Damit soll festgestellt werden, ob das Gerät im Netzwerk existiert. Der Ping-Vorgang wird maximal dreimal in Folge in zehn Sekunden Abständen vollzogen. Der Quellcode 4.13 zeigt die Methode zum Anpingen von IP-Adressen.

```
public async static Task<bool> ValidateIP(string ip)
{
    try
    {
        // Pinge die IP an.
        Ping pingSender = new Ping();
        IPAddress address = IPAddress.Parse(ip);

        // Schaffe einen willkürlichen Buffer von 32 Bytes zum Senden.
        string data = "aaaaaaaaaaaaaaaaaaaaaaaaaaaa";
        byte[] buffer = Encoding.ASCII.GetBytes(data);

        // Warte zehn Sekunden auf eine Antwort.
        int timeout = 5000;
        PingReply reply = await pingSender.SendPingAsync
            (address, timeout, buffer);

        if (reply.Status == IPStatus.Success)
            return true;

        return false;
    }
    ...
}
```

```
}
```

Quellcode 4.13: Der Ping-Aufruf zum Testen der IP-Adresse.

Beim Anpingen wird nicht die Art des Gerätes festgestellt. Wenn die IP gültig im Netzwerk registriert ist, wird die IP bestätigt. Bei keiner Antwort wird der Vorgang abgebrochen und dem Nutzer mitgeteilt, dass das Gerät derzeit nicht zur Verfügung steht. Die Art der Mitteilung wird in Unterunterabschnitt 4.4.1 erklärt.

Eigenschaftsfenster

Das Eigenschaftsfenster ist das Hauptfenster der Anwendung. In diesem Fenster befinden sich die Funktionen zur Kommunikation und Verarbeitung der Kategorien. Die Kommunikation wird ausschließlich von den Controllern übernommen. Jede Kategorie besitzt seinen eigenen Controller. Die Controller der Kategorien übergeben die vom Nutzer eingegebenen Informationen weiter an das entsprechende IoT-Gerät. Die Verarbeitung der Dateien geschieht aufgrund der Initialisierung der Handler bei der CodeBehind-Datei des Fensters (siehe Unterabschnitt 2.5.1). Die Codebehind-Datei ist unterteilt in die Kategorien Gerät, Profil, Aktuelles, Veranstaltungen, Belegungsplan und Wetter. Die Implementierung für die Kategorien ist wie Folgt aufgebaut.

Gerät. Die Kategorie Geräte greift auf die Standard-API zu und entnimmt Informationen über das Netzwerk, die Betriebssystemversion und den Gerätenamen. Zusätzlich kann das Passwort und die Bezeichnung des Gerätes verändert werden. Die Buttons De-/Installation öffnet das Progress-Fenster und leiten die Bereitstellung der APP-Pakete samt ihrer Abhängigkeiten ein. Mehr dazu in Unterunterabschnitt 4.4.2. Für die Kommunikation zur Standard-API wird die Rest-Dokumentation von Microsoft verwendet. Für die Entnahme der Informationen wird eine verschlüsselte Authentifizierung des Clients verlangt. Quellcode 4.14 zeigt die Implementierung zur Authentifizierung.

```
public static void ChangeHttp(string user, string password)
{
    //Erstelle einen HTTP-Client.
    http = new HttpClient();

    //Schreibe die Authentifizierungsdaten in Base64 kodiert
    //in den Header
    var byteArray = Encoding.ASCII.GetBytes
        (user+":"+password);
    var header = new AuthenticationHeaderValue
        ("Basic", Convert.ToBase64String(byteArray));
    http.DefaultRequestHeaders.Authorization = header;
}
```

```

    }

    public static async Task<bool> Authentification(string ip)
    {
        ...
        //Versuche eine Kommunikation mit dem IoT-Geraet
        //durchzufuehren
        var response = await
            http.GetAsync(String.Format
                ("http://{0}:8080/api/iot/device/information", ip));
        return response.IsSuccessStatusCode;
        ...
    }
}

```

Quellcode 4.14: Authentifizierung zur Standard-API

Belegungsplan. Jeder einzelner Tag des Belegungsplans wird in ein eigenes Feld gespeichert. Jedes Feld hat als Länge die Anzahl der Stunden, die pro Tag zur Verfügung stehen (Von 8 bis 20. Insgesamt zwölf Stunden). Beim Hinzufügen einer Veranstaltung wird der Name von diesem als String in das entsprechende Feld gespeichert. Der Nutzer sieht das Ergebnis des Vorganges auf dem Layout (siehe Unterunterabschnitt 4.4.1).

Die Handler, die nach jedem Hinzufügen einer Veranstaltung erstellt werden, werden durch den Maus-Fokus ausgelöst. Wenn die Veranstaltung markiert ist, wird die Veranstaltung in ein Hilfsfeld gespeichert und für das Löschen freigegeben. Wird die Markierung aufgelöst, wird der Eintrag vom Hilfsfeld rückgängig gemacht. Quellcode 4.15 zeigt die Realisierung des Ganzen.

```

//Bei Erhalt des Maus-Fokus
private void BelegungsPlanCell_OnFocus
(object sender, MouseButtonEventArgs e, TextBlock textBlock)
{
    //Faerbe den Hintergrund Gelb
    Border border = sender as Border;
    border.Background = new SolidColorBrush(Colors.Yellow);
    //Fuege die Veranstaltung in den Hilfsarray
    helpBorders.Add(border);
    helpTextBlocks.Add(textBlock);
    //Tausche die Eventhandler
    border.MouseLeftButtonDown -= new MouseButtonEventHandler
        ((sender2, e2) => BelegungsPlanCell_OnFocus

```

```

        (sender2, e2, textBlock));
    border.MouseLeftButtonDown += new MouseButtonEventHandler
        ((sender2, e2) => BelegungsPlanCell_FocusLost
        (sender2, e2, textBlock));
}

//Bei Verlust des Maus-Fokus
private void BelegungsPlanCell_FocusLost
(object sender, MouseButtonEventArgs e, TextBlock textBlock)
{
    //Faerbe den Hintergrund Hellblau
    Border border = sender as Border;
    border.Background = new SolidColorBrush(Colors.LightBlue);

    //Entferne den Eintrag vom Hilfsarray
    helpBorders.Remove(border);
    helpTextBlocks.Remove(textBlock);

    //Tausche die Eventhandler
    border.MouseLeftButtonDown -= new MouseButtonEventHandler
        ((sender2, e2) => BelegungsPlanCell_FocusLost
        (sender2, e2, textBlock));
    border.MouseLeftButtonDown += new MouseButtonEventHandler
        ((sender2, e2) => BelegungsPlanCell_OnFocus
        (sender2, e2, textBlock));
}

```

Quellcode 4.15: Eventhandler fuer den Maus-Fokus

Die Felder werden beim Speichern zum entsprechenden IoT-Board weitergeleitet. Das Absenden erfolgt durch die eigen-implementierte API.

Restliche Kategorien. Die Daten der einzelnen Kategorien, die vom Nutzer eingegeben worden sind, werden aus den einzelnen Komponenten entnommen, in Objekt umgewandelt und weitergeleitet. Die Implementierungen sind strikt und besitzen keinerlei Besonderheiten.

Progressfenster

Das Progressfenster dient zur De-/Installation der IoT-Anwendung auf dem IoT-Gerät. Für die Kommunikation mit dem IoT-Gerät nutzt es einen zusätzlichen Controller, der für das Neustarten, Ausschalten und für die Installation von Anwendungen vorhanden ist (Hardwarecontroller).

Bei der Installation werden die Abhängigkeiten der Anwendungen und die Anwendung selbst durch die Standard-API an das IoT-Gerät gesendet. Der Transfer der Pakete geschieht einzeln mit zeitlichen Abständen. Nach jedem Sendevorgang fragt die Anwendung in zehn Sekundenabständen den Installationsstatus des IoT-Geräts ab. Wenn ein Success-Code eintrifft, wird mit der Installation der nächsten Anwendungen fortgefahren. Quellcode 4.16 zeigt die Installation einer Anwendung durch die Standard-API.

```

public async static Task<bool> UploadApp
(string ip, string path, string fileName)
{
    //Vorbereitung und Kodierung der notwenigen Parameter (fileName)
    var param = new Dictionary<string, string> {
        { "package", fileName}
    };
    var urlparam = new FormUrlEncodedContent(param);
    var urlVar = await urlparam.ReadAsStringAsync();

    //Erstellen des RestClients mit Authentifizierung
    //((Benutzername und Passwort)
    string[] load = Save.LoadLogin(ip);
    var client = new RestClient(String.Format
    ("http://{0}:8080/api/app/packagemanager/package?{1}", ip, urlVar))
    {
        Authenticator = new HttpBasicAuthenticator(load[1], load[2])
    };
    var request = new RestRequest(Method.POST);

    //Kodierung der Anwendung in ein FileParameter
    FileStream stream = File.Open(path, FileMode.Open);
    request.Files.Add(new FileParameter
    {
        Name = "fileAttach",
        Writer = (s) =>
        {
            stream.CopyTo(s);
            stream.Dispose();
        },
        FileName = fileName,
    });
}
```

```
        ContentLength = stream.Length  
    );  
  
    //Senden und warten auf Abschluss des Ganzen  
    IRestResponse response = client.Execute(request);  
    var content = response.Content;  
    return response.IsSuccessfull;  
}
```

Quellcode 4.16: Installation einer Anwendung auf dem IoT-Gerät

4.4.3 Speicher- und Profilsystem

Das Speicher- und Profilsystem wurde durch ein Verzeichnissystem realisiert. Übergeordnet existiert für jede IP-Adresse ein eigener Ordner. Darin befinden sich die erstellten Profile und ein Default-Verzeichnis. Jeder dieser Profile ist ein eigenes Verzeichnis und besitzt für jede Kategorie eine eigene Textdatei, die in JSON codiert ist.

Wenn ein Profil angelegt wird, wird ein zusätzliches Verzeichnis in der jeweiligen IP-Adresse erstellt. Wenn kein Profil angelegt wird, werden die Daten in dem Default Ordner gespeichert. Ein Beispiel für die Speicherung der Kategorie Aktuelles wäre: .../Namenschild-Desktop/bin/169.254.108.243/Default/Aktuelles.txt.

Kapitel 5

Benutzerhandbuch

In diesem Kapitel werden die Funktionen für die Nutzung der Anwendung erklärt.

5.1 Installation

In diesem Abschnitt wird die Installation der Anwendung erläutert.

5.1.1 Installationsvoraussetzungen

Für das korrekte Ausführen der Anwendung wird folgende Software benötigt.

- (Desktop) Das Betriebssystem Windows Vista oder Aktueller
- (Desktop) .Net Framework 4.6.1 oder Aktueller
- (IoT Board) Das Betriebssystem Windows 10 IoT.
- (Beide) Ein gemeinsames Netzwerk.

5.1.2 Ausführen der Anwendung

Entpacke Sie die ZIP-Datei in ein eigenen Ordner und führen Sie die Anwendung NamenschildDesktop.exe aus.

5.2 Verbindungs vorgang

Dieser Abschnitt beschäftigt sich mit dem Startfenster der Anwendung. In diesem Fenster werden die IP-Adressen der Geräte verwaltet.

5.2.1 Verbindung hinzufügen

Für eine Verbindung mit dem IoT-Gerät, vergewissern Sie sich, dass das IoT-Gerät das Betriebssystem Windows 10 IoT installiert hat und beide Geräte im gleichen Netzwerk angeschlossen sind.

Fügen Sie am Hauptfenster, die IP des IoT-Geräts in das IP-Textfeld ein und klicken Sie auf IP Hinzufügen. Die Anwendung fügt nun die IP in die Liste ein und überprüft, ob dieser erreichbar ist. Der Status der Überprüfung wird neben dem Eintrag farblich angezeigt. Mögliche Status-Nachrichten sind: Online, Offline und Checking. Die Zeit für die Überprüfung kann je nach Netzwerk variieren.

5.2.2 Verbindung löschen

Für das Löschen einer IP markieren Sie die IP aus der IP Liste und klicken Sie auf Entfernen.

5.2.3 Verbindungen überprüfen

Für die Aktualisierung der Statusmeldungen für die eingetragenen IP-Adressen, klicken Sie auf Verbindungen prüfen. Die Anwendung geht alle IP-Adressen in der Liste schrittweise durch. Während Vorgangs können IP-Adressen entfernt und hinzugefügt werden.

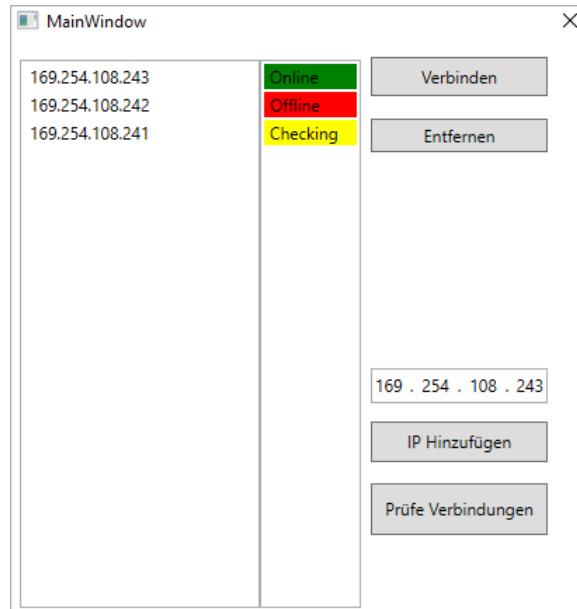


Abbildung 5.1: Startfenster der Anwendung. Hier verwalten Sie die IP-Adressen der IoT-Geräte.

5.2.4 Eigenschaftsfenster öffnen

Markieren Sie eine IP-Adresse in der Liste und klicken Sie auf Verbinden. Es öffnet sich ein Dialogfenster. Geben Sie die Anmelddaten des IoT-Geräts für die entsprechende IP-Adresse in die Textfelder ein und klicken Sie auf OK.

Optional können Sie die Daten durch einen Klick auf die Checkbox **Anmelddaten speichern** lokal speichern. Das Dialogfenster für diese IP wird danach nicht mehr erscheinen.

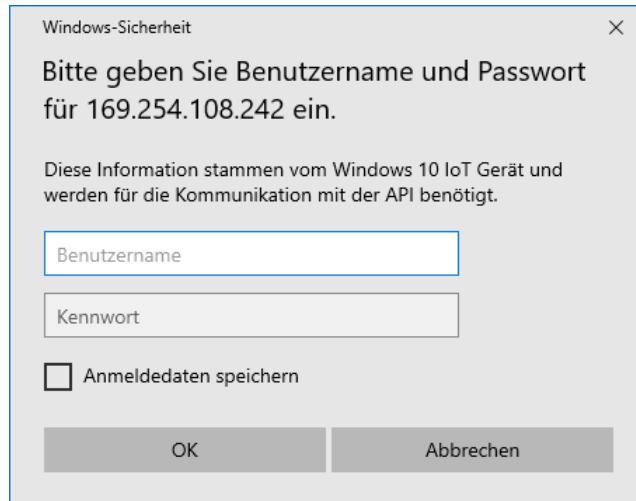


Abbildung 5.2: Geben Sie hier Ihre Anmelddaten für das Gerät ein.

Das Öffnen des Eigenschaftsfensters ist auch ohne Anmelddaten möglich. Es können jedoch keine Daten gesendet und empfangen werden. Die Einträge des Nutzers werden nur lokal gespeichert.

5.3 Eigenschaftsfenster

In diesem Fenster kann die Anzeige für die einzelnen Kategorien bearbeitet werden. Die Kategorien unterteilen sich in: Gerät, Profil, Aktuelles, Veranstaltungen, Belegungsplan und Wetter. Jede Kategorie ist durch die Tab-Navigation aufrufbar.

5.3.1 Gerät

Diese Kategorie enthält die Allgemeinen Einstellungen des IoT-Geräts. Hier kann die IoT-Anwendung installiert, Netzwerkdaten des IoT-Geräts aufgerufen, ein Screenshot des Geräts erstellt, der Gerätename und das Zugangspasswort geändert und Speicherprofile erstellt und geladen werden.

Speicherprofil erstellen und laden

Für das Erstellen eines Profils wählen Sie beim Dropdown-Menü Profil laden **Neues Profil** aus. Es erscheint ein Dialogfenster. In diesem Dialogfenster geben Sie den Namen des neuen Profils ein. Das Profil wird erstellt und standardmäßig ausgewählt.

Für das Laden eines Profils wählen Sie beim Dropdown-Menü Profil laden, das entsprechende Profil aus. Die Informationen der Kategorie werden nach der Auswahl von der Anwendung geladen.

IoT-Anwendung de-/installieren

Wichtig: Dieser Vorgang ist notwendig, um die restlichen Funktionen des Abschnittes ordnungsgemäß zu verwenden.

Vor der Installation vergewissern Sie sich, dass das IoT-Gerät die Voraussetzungen aus Unterabschnitt 5.1.1 erfüllt.

Klicken Sie auf Installieren. Es öffnet Sie ein Fenster mit einem Fortschrittsbalken, der den Vorgang der Installation mit den notwendigen Einstellungen einleitet. Dieser Vorgang kann einige Minuten dauern. Nach einer erfolgreichen Installation öffnet sich ein Dialog-Fenster mit dem Titel **Installation beendet**.

Nach der Installation stehen Ihnen die kommenden Funktionen zur Verfügung.

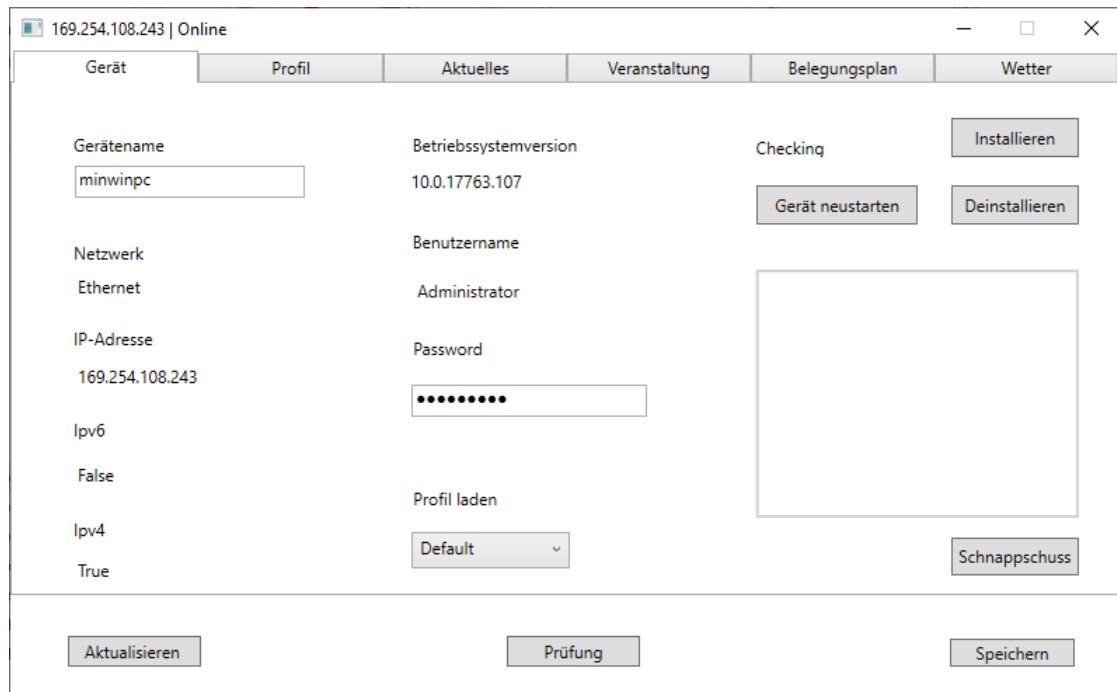


Abbildung 5.3: In diesem Fenster können Sie die Einstellungen für eine konkrete IoT-Anwendung ausführen.

5.3.2 Ein- / und Ausschalten von Kategorien

Jede Kategorie außer Gerät oben links eine Checkbox mit dem Label **Tab einschalten**. Standardmäßig sind alle Tabs deaktiviert. Für das Aktivieren bzw. Deaktivieren einer Kategorie klicken Sie auf die Checkbox. Wenn die Checkbox deaktiviert ist, werden auch die Eingabefelder der Kategorie deaktiviert. Sie können keine Eingaben tätigen.

Bei Einschalten der Kategorie können wieder Eingaben getätigt werden.

Wichtig: Für das Senden der Tab-Einstellungen muss mindestens ein Tab aktiv sein.

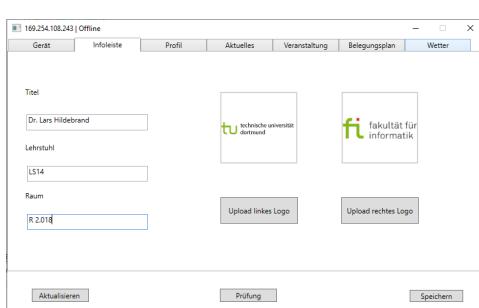
5.3.3 Infoleiste verändern

Die Infoleiste bearbeitet die oberste Leiste des Displays. Wählen Sie in der äußeren Tab-Navigation die Kategorie Infoleiste aus. Nun können Sie die Informationen der Infoleiste bearbeiten.

Zusätzlich können Sie zwei Symbole hinzufügen. Das Symbol kann rechts oder links positioniert werden.

Wenn Sie auf **Upload linkes/rechtes Logo** öffnet sich ein Verzeichnis. Wählen Sie das entsprechende Symbol aus. Das Symbol wird nun auf dem Eigenschaftsfenster angezeigt.

Wichtig: Das Programm unterstützt derzeit nur Bildformate mit der Endung **.png**.



(a)



(b)

Abbildung 5.4: Dieses Fenster ermöglicht die Verwaltung der obersten Navigationsleiste der IoT-Anwendung

5.3.4 Profile erstellen/verändern

Für das Erstellen eines Profils wählen Sie in der Tab-Navigation **Profil** aus. Hier befindet sich eine weitere Tab-Navigation mit drei Profilen. Die Anwendung unterstützt derzeit maximal drei Profile.

Für das Erstellen eines Profils, klicken Sie zunächst auf ein Profil Tab in der unteren Tab-Navigation und dann auf die Checkbox Profilaktivieren. Nun tragen Sie die Daten des

Profils ein. Es muss nicht jedes Eingabefeld ausgefüllt werden.

zusätzlich ein Profilbild hinzugefügt werden. Klicken Sie auf **Upload Profil Logo** und laden Sie eine Bilddatei hoch. **Wichtig: Das Programm unterstützt derzeit nur Bildformate mit der Endung .png.**

Für das Aktivieren von weiteren Profilen wiederholen Sie das Ganze mit den anderen Profilen in der Tab-Navigation.

(a)
(b)

Abbildung 5.5: Hier können Sie die Besitzer des Raumes festlegen. Es können maximal drei Besitzer zugewiesen werden.

5.3.5 Textnachricht bereitstellen

Für das Bereitstellen einer Textnachricht klicken sie im Tab-Navigator auf **Aktuelles** und geben Sie einen Titel und den entsprechenden Text in die Textfelder ein.

(a)
(b)

Abbildung 5.6: Tragen Sie eine Nachricht ein, die auf dem Bildschirm angezeigt wird.

5.3.6 Veranstaltung hinzufügen

Für das Hinzufügen von Veranstaltungen klicken Sie bei der Tab-Navigation auf **Veranstaltung** und geben Sie auf dem Dropdown-Menü die Anzahl der Veranstaltungen ein. Es können Maximal vier Veranstaltungen aufgelistet werden.

Jede Veranstaltung hat einen Titel, einen Tag und eine Anfangs-/Enduhrzeit.

Mögliche Eingaben für den Tag sind die Wochentage Montag bis Freitag. Wählen Sie aus dem Dropdown-Menü den entsprechenden Tag aus.

Für die Anfangs-/Enduhrzeit stehen jeweils ein Dropmenü zur Verfügung. Gültige Zeiteingaben liegen zwischen 08:00 und 20:00 Uhr.

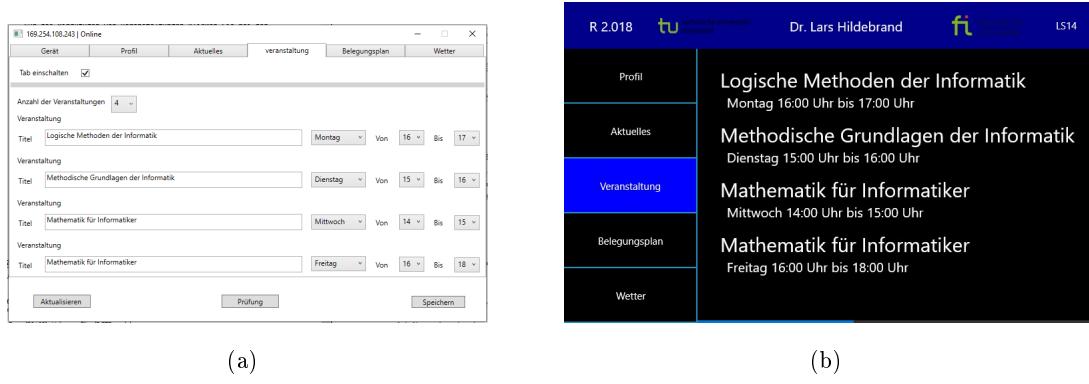


Abbildung 5.7: Es können maximal bis zu vier Veranstaltungen eingetragen werden.

5.3.7 Belegungsplan

Der Belegungsplan ist unabhängig von der Kategorie Veranstaltung.

Zum Hinzufügen einer Veranstaltung in der Kategorie Belegungsplan, klicken Sie in der Tab-Navigation auf **Belegungsplan**. Es öffnet sich eine leere Tabelle mit Eingabefeldern rechts.

Die Eingabefelder ähneln denen aus der Kategorie Veranstaltungen (siehe Unterabschnitt 5.3.6)

Veranstaltung hinzufügen

Geben Sie die entsprechenden Informationen der Veranstaltung rechts ein und klicken Sie auf Hinzufügen. Die Veranstaltung wird nun in die Tabelle eingetragen.

Veranstaltung löschen

Zum Entfernen der Veranstaltung markieren Sie die Veranstaltung auf dem Belegungsplan mit einem Klick und klicken Sie danach auf den Button Entfernen.

5.3.8 Wetter anzeigen lassen

Die Einstellungen dieser Kategorie sind für eine Verbindung mit dem Wetterdienst (OpenWeatherMap.org) notwendig.

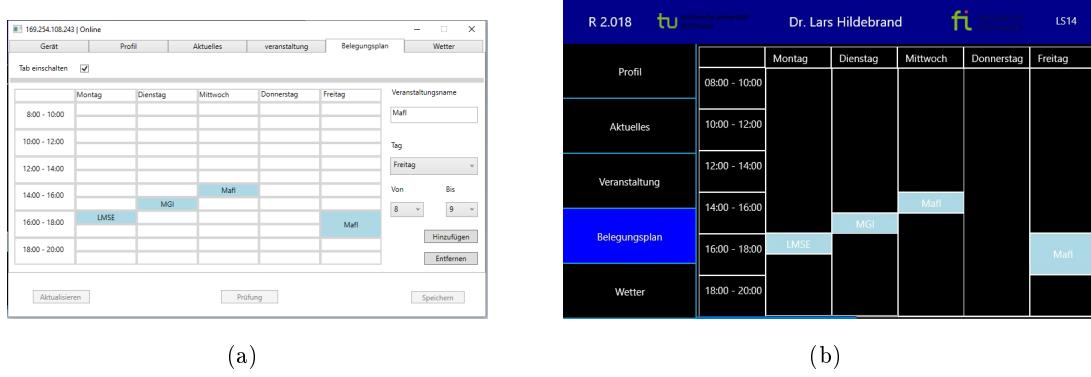


Abbildung 5.8: Jede Änderung bewirkt eine Veränderung auf dem Belegungsplan.

Klicken Sie auf die Tab-Navigation Wetter und geben Sie einen API-Key für den Wetterdienst ein.

Einen API-Key erhalten Sie bei einer erfolgreichen Registration auf OpenWeatherMap.org.



Abbildung 5.9: Geben Sie den API-Key für das Anzeigen vom Wetter ein. Sie erhalten den Key bei einer erfolgreichen Registration auf der OpenWeatherMap.org Seite.

5.4 Prüfungsalarm aufrufen

Zum Starten und Beenden des Prüfungsmodus klicken Sie auf den Button **Prüfung** im Eigenschaftsfenster.

5.5 Abrufen der Kategoriedaten von der IoT-Anwendung

Klicken Sie auf den Button **Aktualisieren** im Eigenschaftsfenster.

Warnung: Gespeicherte Daten werden nach diesem Vorgang überschrieben!



Abbildung 5.10: Aufruf des Prüfungsmodus

5.6 Daten senden

Klicken Sie beim Eigenschaftsfenster auf den Button **Speichern** um Ihre Veränderung auf die IoT-Anwendung anzuwenden.

Kapitel 6

Zusammenfassung und Ausblick

Dieses Kapitel enthält eine grobe Zusammenfassung der Arbeit. Weiterhin beinhaltet es Ideen für eine zukünftige Weiterentwicklung des Projekts.

6.1 Zusammenfassung

Das Ziel dieser Ausarbeitung war es eine IoT-Anwendung (Türschilder) in einem dezentralen Netzwerk zu schaffen. Vor der eigentlichen Projektrealisierung war es notwendig, Grundlagen (siehe Kapitel 2) zu bilden, die eine Realisierung ermöglichen. Dieses Kapitel beschäftigte sich mit der unkommentierten Sammlung von Informationen zu geeigneter Hardware und Software. Es wurden verschiedene Arten von IoT-Geräten und IoT-Betriebssysteme erläutert. Enthalten waren zwei **Mikrocontroller** und ein **Single Board Computer**. Die getesteten IoT-Betriebssysteme befassten sich nur mit dem **Single Board Computer**, da bereits vorherige Tests ergeben hatten, dass Mikrocontroller für die Umsetzung nicht geeignet waren (siehe 3.2.1). Getestet wurden die IoT-Betriebssysteme Windows 10 IoT, Android Things und Raspbian.

Abseits davon enthielten die restlichen Abschnitte Erläuterungen zu technische Hilfsmitteln für die Umsetzung des Projektes und Begriffserläuterungen für den Leser. Die Hilfsmittel wurden dabei auch mit ihren Alternativen erläutert.

Mit Kapitel 3 folgte die Anforderungsanalyse, der zu entwickelnden Anwendung. Es wurden primäre und optionale Funktionen für die einzelnen Anwendungen formuliert. In Abhängigkeit wurde darauffolgend ein Konzept für die Realisierung erstellt. Das Konzept beinhaltete die begründete Auswahl von Hilfsmittel aus den Grundlagen Kapitel, eine grobe Strategie der Funktionsrealisierung und Design Entwürfe für das Endprodukt. Die ausgewählte Hilfsmittel waren:

Hardware. Raspberry Pi 3. Wegen Performance und Nebenläufigkeit, die das Ausführen von mehreren Anwendungen ermöglicht.

Netzwerke. Dezentrales Netzwerk. Aufgrund der Unabhängigkeit durch eine Zentralinstanz und der eindeutigen Bestimmung von Client und Server.

IoT Betriebssystem. Windows 10 IoT. Wegen der Kompaktheit und der zusätzlichen Funktionen, wie das mitgelieferte Web-API.

SOAP und REST. REST. Aufgrund der mangelnden Unterstützung von SOAP vom Betriebssystem und der Simplizität.

Programmiersprachen. XAML und C#. Wegen der bevorzugten Unterstützung der Sprachen durch das Betriebssystem.

In Kapitel 4 folgte die technische Realisierung des Projektes. Zunächst wurde eine Softwarearchitektur zusammengestellt. Die Architektur beinhaltete den Aufbau der verschiedenen Anwendungen von Server und Client. Die Serveranwendung wurde unterteilt in die REST-Schnittstelle (Background-APP), die Hauptanwendung (Foreground-APP), den Nachrichtenkanal (UWP-MessageRelay) und die Prüfungsapplikation (ForeGround-Prufung). Die Aufteilung geschah hauptsächlich für eine vereinfachte Wartung des Projekts. Danach folgten zusätzliche technische Erweiterungen, die für die Implementierung genutzt wurden. Im Anschluss wurde die eigentliche Realisierung des Projekts erläutert. Beschrieben wurde der Aufbau und die Logik der einzelnen Funktionen mithilfe von kommentierten Quellcode. Der Aufbau der Anwendungen geschah durch die MVC-Architektur. Die Erläuterungen zu den einzelnen Funktionen wurde aufgeteilt in die (Unter-)Anwendungen. Die Aufteilung der Anwendungen folgte durch die obengenannte zusammengestellte Softwarearchitektur. Die Hauptanwendung (ForeGround-APP) ist die View zur Darstellung von Nutzereingaben, die REST-Schnittstelle (Background-APP) dient zum Empfangen von Daten, das Nachrichtenkanal (UWP-MessageRelay) hat die Aufgabe eine interne Kommunikation der Serverprozesse zu ermöglichen, die Prüfungsanwendung (ForeGround-Prufung) stellt ein zusätzliches Fenster für die Darstellung einer Prüfungsbenachrichtigung dar und die Clientanwendung ermöglicht Nutzer die Eingabe von Daten, die auf dem IoT-Gerät dargestellt werden.

Zum Schluss wurde in Kapitel 5 ein bebildertes Handbuch für die Nutzung der Anwendung erstellt. Es enthält jede mögliche Funktion die genutzt werden kann.

6.2 Ausblick

Entwickelt wurde eine vollständige Anwendung für die Darstellung von Türschildern auf Basis von IoT in einem dezentralen Netzwerk. Die Funktionen die implementiert worden sind, erfüllen eine gute Abdeckung der Darstellung von Informationen zu einzelnen Bereichen für die Vergabe von Veranstaltungsräumen. Abgedeckt werden die Kategorien Profil, Aktuelles, Veranstaltungen, Belegungsplan und Wetter. Diese Kategorien sind fest

implementiert und können nicht erweitert oder grundlegend verändert werden. Eine zukünftige Weiterführung des Projektes wäre es ein Kategorien-Editor zu entwickeln, der zusätzliche Kategorien mithilfe einer Echtbilddarstellung erstellen kann und die vorhandenen Anwendungen erweitert. Diese Kategorien sollen dem Nutzer zur Verfügung stehen und ihn ermöglichen vorhandene fest-implementierte mit den benutzerdefinierten Kategorien auszutauschen. Die Funktionen können dabei über die Hauptanwendung für intelligente Türschilder hinausgehen. Beispielsweise eine Kategorie zum Abspielen von Musik und Radio. Der Nutzer gibt eine URI für die entsprechende Datei in die Clientanwendung an und spielt die Audiodatei durch einen Druck auf den entsprechenden Button auf dem IoT-Gerät ab. Erstellt wird die Anwendung durch den zuvor genannten Editor. Zudem könnten die erstellten Anwendungen mit anderen Nutzern geteilt werden. Es könnte sich eine Community aufbauen, die das Projekt stetig erweitert.

Abbildungsverzeichnis

2.1	Links Raspberry Pi 1, Mitte Raspberry Pi 2, Rechts: Raspberry Pi 3. Entnommen und leicht verändert aus Quelle [25]	4
2.2	a) Ein Raspberry Pi 3 mit Bildschirm von vorne; b) Ein Raspberry Pi 3 mit Bildschirm von hinten. [Quelle: Selbst erstellt]	4
2.3	Der „Uno Rev 3“ von Arduino [12]	5
2.4	a) Ein ESP-Wrover-Kit von hinten; b) Ein ESP-Wrover-Kit von vorne [9]	6
2.5	Links: Zentrales Netzwerk; Mitte: Dezentrales Netzwerk; Rechts: Verteiltes Netzwerk [13]	7
2.6	Startbildschirm von Windows 10 IoT [Quelle: Selbst erstellt]	10
2.7	Weboberfläche der Windows 10 IoT Seite [Quelle: Selbst erstellt]	11
2.8	Desktop vom Raspbian [Quelle: Selbst erstellt]	12
2.9	Startbildschirm von Android Things [Quelle: Selbst erstellt]	14
2.10	Startbildschirm von Contiki [8].	15
2.11	Beispiel URI für eine REST Abfrage	20
2.12	Visualisierung des Beispiels von 2.5 und 2.6	24
2.13	Visualisierung eines möglichen hierarchischen Aufbaus [20])	26
2.14	Visualisierung des Beispiels von 2.7 und 2.8)	26
2.15	Echtbilddarstellung von Microsoft Visual 2017. Links Quellcode in Xamarin; Recht: Visualisierung des Quellcodes	29
3.1	Die Grundarchitektur der Serveranwendung	37
3.2	Entwurf des Grundgerüsts der IoT-Anwendung	40
3.3	Entwurf einer einzelnen Profildarstellung	40
3.4	Entwurf einer dualen Profildarstellung	41
3.5	Entwurf einer triple Profildarstellung	41
3.6	Entwurf mit vier Veranstaltungen	42
3.7	Entwurf eines Belegungsplans mit drei Veranstaltungen	42
3.8	Entwurf der Kategorie Wetter	42
3.9	Entwurf der IP-Liste (Durchgehende Kästen sind Buttons; Kästen mit Lücken sind Textfelder)	43

3.10 Entwurf des Verbindungsfensters (Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld))	44
3.11 Entwurf der Geräteinformationsoberfläche. L = Label (Information kann vom Benutzer nicht verändert werden); Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld); Nach unten zeigender Pfeil (Dropdown-Menü)	44
3.12 Entwurf der Verwaltungsoberfläche für ein Profil von insgesamt drei möglichen. L = Label (Information kann vom Benutzer nicht verändert werden); C = Combobox; Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld)	45
3.13 Entwurf der Verwaltungsoberfläche für die Informationsleiste. Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld)	45
3.14 Entwurf der Verwaltungsoberfläche für die Kategorie Aktuelles. C = Combobox; Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld)	46
3.15 Entwurf der Geräteinformationsoberfläche. L = Label (Information kann vom Benutzer nicht verändert werden); C = Combobox; Kasten mit Lücken (Textfeld); Nach unten zeigender Pfeil (Dropdown-Menü)	46
3.16 Entwurf der Geräteinformationsoberfläche. L = Label (Information kann vom Benutzer nicht verändert werden); C = Combobox; Durchgehender Kasten (Button); Kasten mit Lücken (Textfeld); Nach unten zeigender Pfeil (Dropdown-Menü)	47
3.17 Entwurf der Wetteroberfläche für den Desktop. Kasten mit Lücken (Textfeld).	47
 4.1 Kommunikationswege zwischen den Anwendungen	51
4.2 Architektur der Anwendungen (durchgehende Pfeile = Bearbeitet/Verändert Komponente; nicht durchgehend = beobachtet die Komponente und reagiert auf Veränderungen)	52
4.3 Ein Credential-Dialog kreiert durch die Ookii Dialog Erweiterung.	56
4.4 Darstellung des Grundgerüsts ohne variablen Inhalt.	58
4.5 Endfassung des Belegungsplanes.	60
4.6 Benutzung des Online-Tools Json2csharp [17].	63
4.7 Endfassung der Kategorie Wetter.	64
4.8 Grafische Darstellung der Funktionsweise vom UWP-MessageRelay.	69
4.9 Layout der ForeGround-Prufung Applikation.	74
4.10 Layout des IP-Liste Fensters.	75
4.11 Layout des Eigenschaftsfensters mit dem Tabinhalt Gerät.	76
4.12 a) Nicht markierte Veranstaltung im Belegungsplan; b) Markierte Veranstaltung im Belegungsplan	77
4.13 a) Progress-Fenster für die Installation; b) Progress-Fenster für die Deinstallation	77

5.1	Startfenster der Anwendung. Hier verwalten Sie die IP-Adressen der IoT-Geräte.	86
5.2	Geben Sie hier Ihre Anmeldedaten für das Gerät ein.	87
5.3	In diesem Fenster können Sie die Einstellungen für eine konkrete IoT-Anwendung ausführen.	88
5.4	Dieses Fenster ermöglicht die Verwaltung der obersten Navigationsleiste der IoT-Anwendung	89
5.5	Hier können Sie die Besitzer des Raumes festlegen. Es können maximal drei Besitzer zugewiesen werden.	90
5.6	Tragen Sie eine Nachricht ein, die auf dem Bildschirm angezeigt wird.	90
5.7	Es können maximal bis zu vier Veranstaltungen eingetragen werden.	91
5.8	Jede Änderung bewirkt eine Veränderung auf dem Belegungsplan.	92
5.9	Geben Sie den API-Key für das Anzeigen vom Wetter ein. Sie erhalten den Key bei einer erfolgreichen Registration auf der OpenWeatherMap.org Seite.	92
5.10	Aufruf des Prüfungsmodus	93

Algorithmenverzeichnis

2.1	Beispiel einer WSDL Konfiguration auf dem Server	16
2.2	Beispiel einer WSDL Anfrage an den Server mit der Konfiguration von Quellcode 2.1	17
2.3	Antwort der WSDL Anfrage von Quellcode 2.2 an den Client	18
2.4	Antwort einer REST Anfrage an den Client	20
2.5	Beispiel einer einfachen Xaml Anwendung (Xaml:Source Code)	23
2.6	Beispiel einer einfachen Xaml Anwendung (C# Source Code)	23
2.7	Beispiel einer einfachen XML Anwendung für Android (XML:Source Code)	24
2.8	Beispiel einer einfachen XML Anwendung für Android (Java Source Code) .	25
2.9	Implizite und Explizite Typisierung in C#	27
2.10	Asynchrone Starten eines Servers in C#	27
2.11	Ausschnitt der Implementierung eines UWP-Messagerelays in C#	31
4.1	Beispiel für die Konvertierung und Wiederherstellung eines JSON-Codes .	52
4.2	Erstellung eines Servers mit der Verbindung eines REST-Controllers.	53
4.3	Erstellung eines Servers mit der Verbindung eines REST-Controllers.	55
4.4	Code für die Darstellung der Infoleiste.	56
4.5	Darstellung des Textblocks mit einem Border.	58
4.6	Kommentierte Implementierung eines Handlers (Hier Timehandler)	60
4.7	Notwendige Berechtigungen für die OpenWeatherMap Kategorie.	62
4.8	Zugriff auf den Geolocator.	62
4.9	Änderung des Package.appxmanifests zur Kategorie AppService.	69
4.10	Run-Methode der Startup-Klasse.	70
4.11	Run-Methode der Startup-Klasse.	71
4.12	Handler für das Empfangen von Nachrichten.	72
4.13	Der Ping-Aufruf zum Testen der IP-Adresse.	78
4.14	Authentifizierung zur Standard-API	79
4.15	Eventhandler fuer den Maus-Fokus	80
4.16	Installation einer Anwendung auf dem IoT-Gerät	82

Literaturverzeichnis

- [1] BERLIN, FU: *Riot*, 2017. <https://riot-os.org/>; abgerufen am 02. April 2019.
- [2] CORPORATION, MICROSOFT: *Erstellen von Komponenten für Windows-Runtime in C# und Visual Basic*. <https://docs.microsoft.com/de-de/windows/uwp/winrt-components/creating-windows-runtime-components-in-csharp-and-visual-basic>; abgerufen am 30. März 2019.
- [3] CORPORATION, MICROSOFT: *Leitfaden für C#*. <https://docs.microsoft.com/de-de/dotnet/csharp/>; abgerufen am 30. März 2019.
- [4] CORPORATION, MICROSOFT: *An overview of Windows 10 IoT*. <https://docs.microsoft.com/de-de/windows/iot-core/windows-iot>; abgerufen am 30. März 2019.
- [5] CORPORATION, MICROSOFT: *Visual Studio-Dokumentation*. <https://docs.microsoft.com/de-de/visualstudio/?view=vs-2017>; abgerufen am 30. März 2019.
- [6] CORPORATION, MICROSOFT: *Was ist eine Universelle Windows-Plattform (UWP)-App?* <https://docs.microsoft.com/de-de/windows/uwp/get-started/universal-application-platform-guide>; abgerufen am 30. März 2019.
- [7] CORPORATION, MICROSOFT: *XAML in WPF*. <https://docs.microsoft.com/de-de/dotnet/framework/wpf/advanced/xaml-in-wpf>; abgerufen am 30. März 2019.
- [8] DUNKELN, ADAMS: *Contiki: The Open Source OS for the Internet of Things*, 2015. <http://contiki-os.org/>; abgerufen am 02. April 2019.
- [9] ESPRESSIF: *ESP-WROVER-KIT V4.1 Getting Started Guide*. PDF. https://www.mouser.de/datasheet/2/891/Espressif_Systems_01162019_ESP-WROVER-KIT-VB-1522995.pdf.
- [10] FOUNDATION, LINUX: *Zephyr Project*, 2019. <https://zephyrproject.org/>; abgerufen am 02. April 2019.

- [11] GITTER, PROF. DR. ALFRED H.: *Noch eine Raspberry Kurzanleitung*. PDF. Steig aktualisiert und zur Verfügung gestellt unter http://web.eah-jena.de/~gitter/nerka/pi_nerka.pdf; abgerufen am 29. März 2019.
- [12] GMBH REICHELT ELEKTRONIK und Co. KG: *ARDUINO UNO DIP 01*. https://cdn-reichelt.de/bilder/web/artikel_ws/A300/ARDUINO_UNO_DIP_01.jpg.
- [13] GRAF, HAGEN: *Zentrale, dezentrale und verteilte Systeme*. <https://blog.novatrend.ch/wp-content/uploads/2017/12/dcap.png>; Abgerufen am 30. März 2019.
- [14] GRAF, HAGEN: *Zentrale, dezentrale und verteilte Systeme*. <https://blog.novatrend.ch/2017/12/25/zentrale-dezentrale-und-verteilte-systeme/>; abgerufen am 30. März 2019.
- [15] GROOT, SVEN: *Ookii*, 2019. <http://www.ookii.org/software/dialogs/>; abgerufen am 30. März 2019.
- [16] JARK, TOM: *Restup*, 2016. <https://www.newtonsoft.com/json>; abgerufen am 30. März 2019.
- [17] KEITH, JONATHAN: *json2csharp*, 2019. <http://json2csharp.com/>; abgerufen am 30. März 2019.
- [18] KÜHNEL, C.: *Embedded Linux mit dem Raspberry Pi: für Ein- und Umsteiger*. Skript Verlag Kühnel, Altendorf, 2013.
- [19] KUROSE, J.F. und K. W. ROSS: *Computer Networking*. Pearson, Malaysia, 7 Auflage, 2017.
- [20] LLC, GOOGLE: *Android Things*. <https://developer.android.com/things/>; abgerufen am 02. April 2019.
- [21] McMANUS, SEAN, MIKE COOK und GERHARD FRANKEN: *Raspberry Pi für Dummies*. Für Dummies. Wiley Publications, Weinheim, 2018.
- [22] NEWTONSOFT: *Json.NET*, 2019. <https://www.newtonsoft.com/json>; abgerufen am 30. März 2019.
- [23] PAJANKAR, ASHWIN: *INTERNET OF THINGS WITH ARDUINO AND BOLD IOT*. BPB Publications, New Delhi, 2018.
- [24] RICHARDSON, LEE: *Android Things*. <https://visualstudiomagazine.com/articles/2017/02/01/solving-uwp.aspx>; abgerufen am 02. April 2019.

- [25] SB-COMPONENTS-LTD: *raspberry-pi-1-2-3-comparison-led-placement.* <https://shop.sb-components.co.uk/>.
- [26] SNELL, J., D. TIDWELL und P. KULCHENKO: *Webservice-Programmierung mit SOAP.* Köln.
- [27] SOFTWARE, SMARTBEAR: *SwaggerHub*, 2019. <https://swagger.io/tools/swaggerhub/>; abgerufen am 02. April 2019.
- [28] TILKOV, S., M. EIGENBRODT, S. SCHREIER und O. WOLF: *REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web.* dpunkt.verlag, Heidelberg, 2015.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 25. Mai 2019

Tolgay Usul

