



Höhere Technische Bundeslehranstalt
und Bundesfachschule
im Hermann Fuchs Bundesschulzentrum

Universal SmartHome

Diplomarbeit

Schulautonomer Schwerpunkt Mobile Computing

ausgeführt im Schuljahr 2019/2020 von:

Johannes Baischer (JB), 5AHELS

Raphael Brachmaier (RB), 5AHELS

Betreuer:

Dipl.-Ing. Gerhard Waser

14. April 2020

Thema:

Universal SmartHome

Subthemen und Bearbeiter:

Anwendungslogik und Serverkommunikation

Johannes Baischer, 5AHELS

Betreuer: Dipl.-Ing. Gerhard Waser

User Interface mit HTML5, CSS und Bootstrap

Raphael Brachmaier, 5AHELS

Betreuer: Dipl.-Ing. Gerhard Waser

DIPLOMARBEIT DOKUMENTATION

Verfasser/innen	Johannes Baischer, Raphael Brachmaier
Jahrgang Schuljahr	5AHELS 2019/2020
Thema der Diplomarbeit	Universal SmartHome
Aufgabenstellung	In den letzten Jahren wurde im Rahmen von Diplomarbeiten ein universelles SmartHome-System entwickelt. In einem Gebäude sind Steuermodule mit verschiedenen I/O's installiert, welche über WLAN mit einem Home-Server verbunden sind. Dieser bietet über Internet eine verschlüsselte HTTP-REST-Schnittstelle zur Konfiguration und Steuerung. Zusätzlich existieren IP-Kameras für Live-Videostreams. Es soll eine neue Webapplikation für alle gängigen Endgeräte entstehen, um diese WLAN-fähigen Schaltrelais zu steuern. Dadurch können angeschlossene Geräte per Smartphone, Tablet oder PC bedient und überwacht werden. Live-Videostreams sollen nach dem Augmented Reality (Erweiterte Realität) Prinzip Geräte und Daten dazu in Echtzeit anzeigen und die selbe Bedienung/-Steuerung ermöglichen. Weiters soll die Verwendung von Web-Sockets für die Übertragung von Livedaten integriert werden.
Realisierung	Für die Entwicklung wird Blazor verwendet, ein neues Microsoft Framework, welches auf C# sowie Razor basiert und MVVM (Model View ViewModel) als Anwendungsarchitektur verwendet. Die Kommunikation zwischen Website und Modul erfolgt über eine REST-API vom hausinternen SmartHome Server. Die Übertragung von Livedaten sowie Livestreams wird mit SignalR-Hubs realisiert. Nach der Installation des Servers und der Module ist für den Benutzer kein technisches Knowhow erforderlich. Für die Benutzeroberfläche wird Bootstrap 4 verwendet, um die Website ansprechend und dynamisch zu gestalten.
Ergebnisse	Das Ergebnis entspricht den Erwartungen. Rein durch Software und Verwendung von Hardware-Modulen ist das Steuern der analogen und digitalen Ein- und Ausgänge (Glühbirne, LED) ermöglicht. Durch die Verwendung von Livestreams mit Augmented Reality ist die Steuerung der Geräte auch rein graphisch möglich. Die Administrator-Verwaltung ist dank einfach und effizient gestalteter Benutzeroberfläche schon mit grundlegendem technischen Wissen möglich.

Typische Grafik,
Foto etc.
(mit Erläuterung)

Startseite des Universal SmartHome:

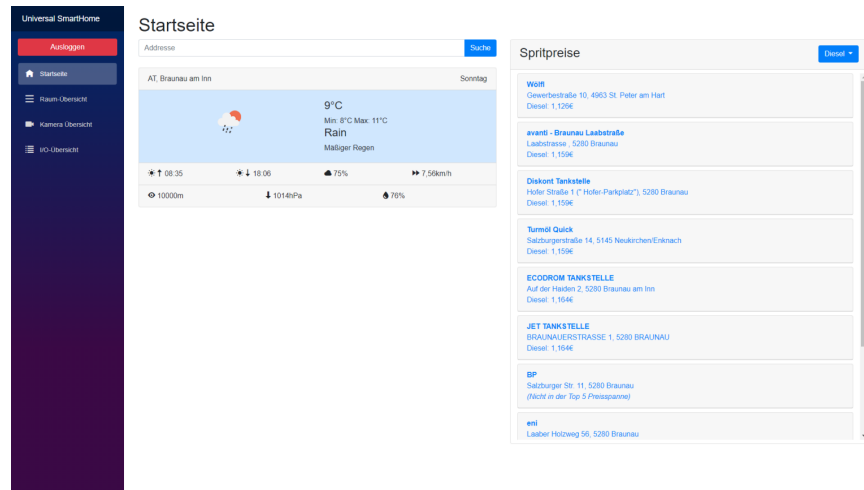


Abbildung 0.1: Bild der Startseite

Teilnahme an
Wettbewerben,
Auszeichnungen

Teilgenommen an:

- Computer Creative
- EDISON der Preis
- FH Kärnten Maturaprojekt-Wettbewerb
- Immotopia
- Jugend Innovativ

Möglichkeiten der
Einsichtnahme
in die Arbeit

Archiv der HTL Braunau, bzw.
<https://diplomarbeiten.berufsbildendeschulen.at/>

Approbation (Datum / Unterschrift)

Prüfer/Prüferin:

Direktor/Direktorin
Abteilungsvorstand/Abteilungsvorständin

DIPLOMA THESIS Documentation

Author(s)	Johannes Baischer, Raphael Brachmaier
Form Academic year	5AHELS 2019/2020
Topic	Universal SmartHome
Assignment of tasks	In recent years, a universal smarthome system has been developed within the scope of diploma theses. In a building control modules with different I/O's are installed, which are connected to a home server via WLAN. This home server offers an encrypted HTTP-REST interface for configuration and control via Internet. Additionally, IP cameras for live video streams are installed. A new web application for all common devices is to be created to control these WLAN-capable switching relays. This allows connected household appliances to be operated and monitored via smartphone, tablet or PC. Live video streams will display appliances and related data in real time based on the Augmented Reality concept and allow the same operation/control. Furthermore, the use of WebSockets for the transfer of live data is to be integrated.
Realisation	For the development Blazor, a new Microsoft framework based on C# as well as Razor while utilizing MVVM (Model View ViewModel) as application architecture is used. Communication between the website and modules is handled via a REST API from the in-house smarthome server. The transmission of live data and live streams is realized with SignalR-Hubs. After installation of the server and the modules, no technical know-how is required for the end-user. For the user interface Bootstrap 4 is used to make the website attractive and dynamic.
Results	The result is in line with expectations. Purely through software and the use of hardware modules the control of analog and digital inputs and outputs (bulb, LED) is possible. By using live streams with Augmented Reality the control of the devices is also possible in a purely graphical way. The administrator maintenance is already possible with basic technical knowledge thanks to a simple and efficiently designed user interface.

Illustrative graph, photo
(incl. explanation)

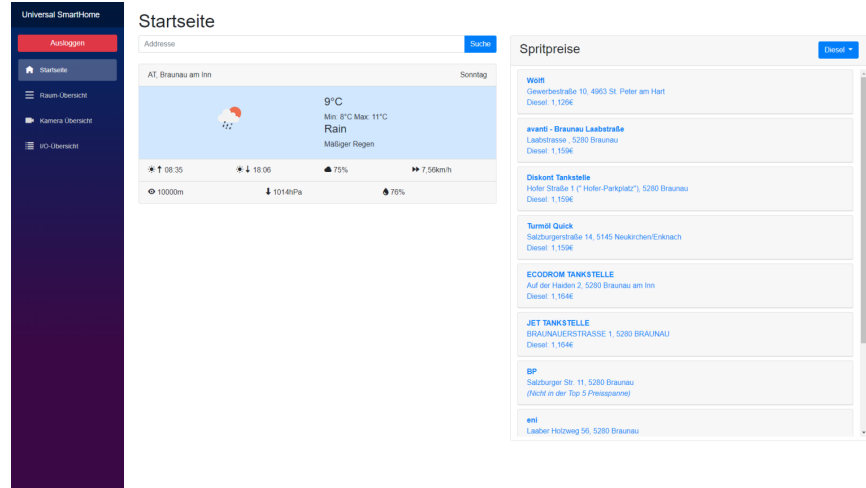


Abbildung 0.2: Picture of the Homepage

Participation in
competitions,
Awards

Participated:

- Computer Creative
- EDISON der Preis
- FH Kärnten Maturaprojekt-Wettbewerb
- Immotopia
- Jugend Innovativ

Accessibility of
diploma thesis

HTL Braunau archive, or
<https://diplomarbeiten.berufsbildendeschulen.at/>

Approval (date / signature)

Examiner

Head of College / Department

Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als angegebene Quellen und Hilfsmittel nicht direkt benutzt und die benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Braunau/Inn, 14.04.2020

Johannes Baischer

Ort, Datum

Vorname Nachname

Unterschrift

Braunau/Inn, 14.04.2020

Raphael Brachmaier

Ort, Datum

Vorname Nachname

Unterschrift

Inhaltsverzeichnis

Vorwort	xi
Zusammenfassung	xii
Abstract	xiii
1 Aufgabenstellung/Pflichtenheft	1
1.1 Thema SmartHome	1
1.2 Aufgabenstellung	1
1.3 Pflichtenheft	2
1.3.1 Benutzer	2
1.3.2 Administrator	2
1.3.3 Optionale Features	2
2 Tools	3
2.1 Blazor ^{JB}	3
2.1.1 Blazor Syntax	3
2.1.2 Einführung in Blazor Projekte	4
2.1.3 Blazor und Webassembly	6
2.2 ASP.NET Core ^{JB}	6
2.3 SignalR ^{JB}	6
2.4 Newtonsoft JSON ^{JB}	6
2.5 Bootstrap ^{RB}	7
2.5.1 Layout	7
2.5.2 Gegenüberstellung	9
2.5.3 Komponenten	9
2.6 MultiLanguage ^{RB}	11
2.7 Toasts ^{RB}	13
3 Benutzeroberfläche^{RB}	14
3.1 Konzept	14
3.1.1 Erstes Design	14
3.1.2 Finales Design	15
3.2 Design und Ablauf	16
3.2.1 Blazor Vorlage	16
3.2.2 Stil festlegen (Bootstrap Einarbeitung)	16
3.2.3 Login Seite	17
3.2.4 Unterschied Admin/User	17
3.2.5 Startseite	18
3.2.6 Übersichts-Seiten	18
3.2.7 Raum-Einzelansicht	19

3.2.8	Kamera/Augmented Reality	19
3.2.9	I/O-Liste einer Gattung	20
3.2.10	Modul-Einzelansicht	21
3.2.11	SmartHome Regeln einrichten	21
3.2.12	Rendering	21
3.3	Fehlerbehandlung graphisch	22
4	Implementierung^{JB}	23
4.1	Software-Architektur	23
4.1.1	Erstes Design	23
4.1.2	Finales Design	23
4.2	Design und Ablauf	24
4.2.1	Blazor Vorlage	24
4.2.2	Code- & Filestruktur	24
4.2.3	Universal SmartHome API	25
4.2.4	JSON Codierung	27
4.2.5	Databinding	28
4.2.6	Der Server	28
4.2.7	Inheritance & Dependency Injection	29
4.2.8	Asynchrone Programmiermodelle	30
4.2.9	Aufbereiten der API Daten	30
4.2.10	Blazor Lifecycle Methoden	31
4.2.11	URL Parameter: /{var}	32
4.2.12	Das Problem mit deutschem Visual Studio	32
4.2.13	SmartHome Regeln	33
4.2.14	Livestreams & Augmented Reality	33
4.2.15	JavaScript - C# Brücke	36
4.2.16	Dynamisches Programmieren: <i>ExpandoObject</i>	37
4.2.17	Live Daten mit SignalR	38
4.2.18	Freigeben von Ressourcen (IDisposable)	42
4.2.19	Internet Information Services (IIS)	43
4.2.20	Wetter-, Sprit-, und GeoCoding-API	44
4.3	Fehlerbehandlung im Code	45
5	Test und Inbetriebnahme^{JB}	47
5.1	Der Server	47
5.2	Die Module	47
5.2.1	Pinbelegung der Module	47
5.3	Das Haus	48
5.3.1	Verkabelung der Geräte	49
5.3.2	Einrichten der Datenbank	50
5.3.3	Basteln des Schreibtisches	50
6	Anwendungsbeispiel^{RB}	51
6.1	Am Morgen	51
6.2	In der Arbeit	51
6.3	Am Abend	52

7	Bedienungsanleitung^{RB}	53
7.1	Erste Einrichtung	53
7.1.1	Erstellen der Räume	53
7.1.2	Modul-Konfiguration	53
7.2	Verwaltungstools	56
7.2.1	Erstellen	56
7.2.2	Löschen	57
7.2.3	Bearbeiten	57
7.3	Regeln	58
8	Fazit und Persönliche Erfahrungen	59
8.1	Fazit	59
8.2	Persönliche Erfahrungen	60
A	Diverse Anhänge	61
A.1	Projektstrukturplan	61
A.2	Relais-Schaltplan	62
A.3	Relais-Skizze	63
A.4	Modulbasis-Schaltplan	64
A.5	Modulbasis-Skizze	65
A.6	Projekttagebuch	66
A.6.1	Baischer	66
A.6.2	Brachmaier	66
	Literaturverzeichnis	67
	Abbildungsverzeichnis	68
	Quelltextverzeichnis	69
	Autoren	71

Vorwort

An dieser Stelle möchten wir uns bei einigen Personen, die uns bei unserer Projektarbeit tatkräftig unterstützt haben, bedanken. Angefangen mit Herrn Direktor Mag. Hans Blocher für die Möglichkeit, vorhandene Praxiseinheiten für die Projektarbeit sowie die Diplomarbeit zu verwenden. Weiters bedanken wir uns bei Frau Ing. Margit Fuchs sowie Herrn DI. Christian Wagner für die Organisation und Bereitstellung eines Miniaturhauses zur Installation des SmartHome Systems. Im Projektmanagement hat uns Frau Fuchs auch sehr viel über zeitliches Management in einem Projekt beigebracht, was uns die gesamte Planung sehr erleichtert hat.

Nicht zuletzt möchten wir uns bei unserem Projektleiter Herrn DI. Gerhard Waser bedanken für die Hilfestellungen beim Heranführen an das Projekt, die Unterstützung bei der laufenden Entwicklung und vor allem auch für die Bereitstellung des Servers und der dazugehörigen API. Herr Waser hat uns immer zeitnah und ausführlich bei Problemen zur Seite gestanden, dafür wollen wir uns besonders bedanken.

Ein solch großes Projekt über ein halbes Jahr intensiv anzugehen, war anfangs zwar eine große Herausforderung, hat uns aber als Softwareentwickler eine Lektion gelehrt. Ein Projekt in seiner groben Struktur zu Beginn durchzudenken ist sehr wichtig, doch man muss sich gleichzeitig bewusst sein, dass man nach einem abgeschlossenen Projekt immer Ideen hat, wie man dieses besser angehen hätte können. Diese Erfahrung mussten wir auch mit unserem SmartHome machen.

Auch darf man nicht davor zurückschrecken, Fehler in der Organisation zu finden und gegebenenfalls Ideen und Code zu verwerfen und neu zu schreiben. Das ist zwar immer eine große Überwindung, hat uns aber in jedem Fall große Fortschritte gebracht, über die man im Nachhinein sehr glücklich ist.

Alles in allem war dieses Projekt eine erfolgreiche Erfahrung die uns fachlich sehr weitergebracht hat und wir sind stolz darauf, was wir geleistet haben.

Zusammenfassung

Dank der bis jetzt vorangeschrittenen Digitalisierung hat nahezu jeder Mensch Zugang zum Internet auf mobilen Geräten wie Smartphones und Tablets. Auf diese Bequemlichkeit "in der Hosentasche" wird mit dem Universal SmartHome aufgebaut. Immer mehr Haushaltsgeräte sind digitalisiert und benötigen Strom. Daraus entstand die Idee, ein SmartHome zu entwickeln, welches diese Geräte optimiert, um trotz technischem Fortschritts energieeffizient und nachhaltig zu leben. Dieser Schritt zum SmartHome soll für Menschen aus allen Lebenssituationen zugänglich sein. Großer Vorteil bei einem selbst entwickelten SmartHome ist, dass die Hardwaremodule zur Steuerung der Elektronik mit einem im jeweiligen Haus installierten Server eingerichtet werden. Dadurch bleiben alle hausbezogenen Daten beim Benutzer und werden nicht auf großen Serverfarmen gespeichert. Dieser Aspekt ist sehr vielen Menschen wichtig, da der Schutz der eigenen Daten im Internet immer mehr zum Thema wird. Die Website ist einfach und intuitiv gestaltet, was es auch älteren Menschen, die eventuell bettlägerig einem Pfleger die Tür öffnen wollen, ermöglicht, vollen Zugriff auf das Haus zu haben. Für ältere Menschen ist der Umstieg auf neue Technologien oft sehr schwer, um nicht zu sagen unmöglich. Hierfür haben wir uns mit der Entwicklung eines visuellen Video-Livestreams beschäftigt, da so der Zugang für Senioren erleichtert wird nach dem Prinzip: "Wenn ich sehe, dass etwas passiert, dann hat es funktioniert". Alles in allem bietet ein so umfassendes SmartHome System für jeden in der einen oder der anderen Weise etwas Ansprechendes, weshalb so etwas in einer immer weiter digitalisierten Welt großen Anklang findet. Für die Entwicklung wurde Blazor, ein neues Microsoft Framework, verwendet, weshalb das Universal SmartHome auch technisch auf dem neusten Stand ist.

Abstract

Thanks to the progress of digitization up to this day, almost everyone has access to the internet on mobile devices such as smartphones and tablets. The Universal SmartHome builds on this convenience "in your pocket". More and more household appliances are digitalized and require electricity. This gave rise to the idea of developing a SmartHome that optimizes these devices in order to live a energy-efficient and sustainable life despite technical progress. This step towards the SmartHome should be accessible to people no matter the circumstances. The big advantage of a self-developed SmartHome is that the hardware modules for controlling the electronics are set up with a server installed in the respective house. This means that all house-related data remains with the user and is not stored on large server farms. This aspect is very important to many people, as the protection of their own data on the internet is becoming more and more of an issue. The website is designed to be simple and intuitive, which also allows older people who may want to open the door to a caregiver when bedridden to have full access to the house. For older people, switching to new technologies is often very difficult, not to say impossible. For this purpose we have been working on the development of a visual video livestream, as this facilitates access for senior citizens according to the principle: "If I see something happening, it has worked". All in all, such a comprehensive SmartHome system offers something appealing to everyone in one way or another, which is why such a system is so popular in an increasingly digitalized world. Blazor, a new Microsoft framework, was used for the development, which is why the Universal SmartHome is also technically up to date.

1 Aufgabenstellung/Pflichtenheft

1.1 Thema SmartHome

Mit dem Begriff *SmartHome* können sich nach und nach mehr Menschen ein Bild vor Augen führen, sei es das Steuern von Haushaltsgeräten mit dem eigenen Smartphone oder auch nur ein Staubsauger-Roboter, der selbstständig den Boden säubert. Mit ständig neuer Vernetzung kommen auch immer mehr Möglichkeiten auf uns zu, um diese Bilder weiter auszumalen. Das eigenen Haus von der Arbeit aus zu überwachen, im Urlaub zu überprüfen, ob man nicht doch ein Licht angelassen hat oder das bequeme Öffnen der Haustür vom Schlafzimmer aus, sind schon lange keine Zukunftsvorstellungen mehr. Doch man darf sich von all dem Glanz der Vorstellungen nicht blenden lassen, denn all das, was man selbst kann, kann bei nicht ausreichender Sicherheit auch jeder Fremde. Viele SmartHome Systeme, die bereits auf dem Markt sind, wurden entwickelt, um dem Benutzer wenige Unannehmlichkeiten bei der Benutzung und vor allem auch bei der Einrichtung solcher Systeme zu bereiten, was auf Kosten der Sicherheit geht. Um sich selbst und sein Haus von außen zu schützen, ist derzeit noch ein gewisses Know-How nötig, um so ein System selbst von Grund auf sicher zu entwickeln und aufzusetzen. Überlegt man also, sich die Bequemlichkeiten eines SmartHomes anzueignen, sollte man sich gut informieren und am besten eine Fachmeinung heranziehen zur Beratung, um die eigene Haustür nicht den Falschen zu öffnen.

1.2 Aufgabenstellung

In den letzten Jahren wurde im Rahmen von Diplomarbeiten ein universelles SmartHome-System entwickelt. In beispielsweise einem Gebäude sind Steuermodule mit verschiedenen I/O's installiert, welche über WLAN mit einem Home-Server verbunden sind. Dieser bietet übers Internet eine verschlüsselte HTTP-REST-Schnittstelle zur Konfiguration und Steuerung. Zusätzlich existieren IP-Kameras für Live-Videostreams.

Zu dieser Ausgangslage soll eine neue Webapplikation (Website) für alle gängigen Endgeräte entstehen. Für die Entwicklung wird Blazor verwendet, ein brandneues Microsoft Framework, welches auf C# sowie Razor basiert und MVVM (Model - View - Viewmodel) als Anwendungsarchitektur verwendet. Live-Videostreams sollen nach dem Augmented Reality Prinzip angezeigt und zur graphischen Steuerung verwendet werden können.

1.3 Pflichtenheft

- Steuerung elektronischer Geräte mittels WLAN-fähigen Modulen und Schaltrelais.
- Vereinfachung der Haus-Steuerung mit Hilfe einer Webapplikation

1.3.1 Benutzer

Mit einem Benutzerkonto soll folgendes möglich sein:

- Ansicht von virtuellen Räumen mit Auflistung aller schaltbaren Elemente
- Kamera-Livestream mit Augmented Reality
- Übersicht aller einzelnen Ein- und Ausgänge

1.3.2 Administrator

Als Administrator sind folgende Funktionen vorgesehen:

- Erstellen, Löschen und Bearbeiten jeglicher Ansichten und schaltbaren Elemente
- Auflistung aller Module und jeweilige Einzelansicht
- Regeln erstellen, um Zeitschaltungen und logische Zusammenhänge zwischen Elementen zu definieren

1.3.3 Optionale Features

- SignalR zur Darstellung von Livedaten
- Vollständiges MultiLanguage
- Startseite mit verschiedenen Widgets

2 Tools

Dieser Abschnitt dient zur Beschreibung von verwendeten Tools, Frameworks und NuGet-Paketen.

2.1 Blazor^{JB}

Grundstein und Anlass der Entwicklung des Universal SmartHomes in neuer Auflage ist das 2019 von Microsoft veröffentlichte Webframework *Blazor*^[1]. Dieses Projekt soll, durch den Ausbau von Webentwicklung in C# statt alt eingesessenem JavaScript, Innovation in die Webentwicklung bringen. Weiters war einer der größten Punkte für Microsoft das Ausführen von Webanwendungen direkt im Browser des Clients anstatt am Server wie bisher.



2.1.1 Blazor Syntax

In Kombination mit Microsofts bewährtem Razor-Syntax zur Erstellung von HTML-Seiten bietet Blazor eine für C#-Entwickler einfache Möglichkeit, dynamische Websites zu erstellen, ohne auf JavaScript zurückgreifen zu müssen. Der neue Syntax erlaubt hier, einfache Codestrukturen wie Schleifen, If-Statements oder Switch-Cases in den HTML-Code einzubinden, um einfache logische Abhängigkeiten beim Laden einer Webpage zu erreichen.

Mit dem @ - Syntax kann man mitten im HTML gewohnten C# Syntax verwenden, Methoden definieren oder spezielle Attribute setzen. So ist es zum Beispiel möglich, mit dem @ref Attribut Objektreferenzen von HTML-Elementen auf C# Objekte zu binden. Auch kann man mit neuen Eventhandlern ganz einfach C# Methoden auf Events von verschiedenen Elementen zuweisen (z.B. @onclick).

```
1 @{
2     switch (type)
3     {
4         case "DigitalInput":
5             <h1>@Globals.MultiLanguage.Digital_Inputs</h1>
6             break;
7         case "DigitalOutput":
8             <h1>@Globals.MultiLanguage.Digital_Outputs</h1>
9             break;
10        case "AnalogInput":
11            <h1>@Globals.MultiLanguage.Analog_Inputs</h1>
12            break;
13        case "AnalogOutput":
14            <h1>@Globals.MultiLanguage.Analog_Outputs</h1>
15            break;
16    }
17 }
18
19 @if (loading_done == false)
20 {
21     <div class="container text-center">
22         <div class="spinner-grow text-primary"></div>
23         <div class="spinner-grow text-primary"></div>
24         <div class="spinner-grow text-primary"></div>
25     </div>
26 }
27 else
28 {
29     ...
30 }
```

Listing 2.1: Blazor Syntax

2.1.2 Einführung in Blazor Projekte

Nachdem nun klar ist, welche neuen Möglichkeiten man mit dem neuen Blazor - Syntax hat, geht es daran zu verstehen, wie all das zusammenwirkt, um vollständige Funktionalitäten in Blazor zu ermöglichen. Mit dem Erstellen eines neuen Blazor - Projekts erhält man gleichzeitig eine Vorlage von Microsoft einer einfachen Anwendung. Diese Vorlage ist ohne weiteres Zutun sofort ausführbar und übernimmt so die Aufgaben wie das Erstellen einer Startup - Logik.

```

1
2 public class Program
3 {
4     public static void Main(string[] args)
5     {
6         CreateHostBuilder(args).Build().Run();
7     }
8
9     public static IHostBuilder CreateHostBuilder(string[] args) =>
10         Host.CreateDefaultBuilder(args)
11             .ConfigureWebHostDefaults(webBuilder =>
12             {
13                 webBuilder.UseStartup<Startup>();
14             });
15 }

```

Listing 2.2: Startup Logik

In der Vorlage befindet sich auch bereits ein kleines Code-Beispiel für das Zusammenspiel von Oberfläche mit dem dahinter liegendem Code. Hier wird auf einer der Seiten ein Button angezeigt, welcher beim Klicken einen Zähler auf der Seite nach oben zählen lässt. Hier wird gezeigt, wie die Oberfläche voll dynamisch mit Objekten aus dem Code (in diesem Fall ein Integer) aufgebaut werden kann und wie die Oberfläche mit dem Code interagieren kann (@onclick - Event).

```

1
2 @page "/counter"
3
4 <h1>Counter</h1>
5
6 <p>Current count: @currentCount</p>
7
8 <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
9
10 @code {
11     private int currentCount = 0;
12
13     private void IncrementCount()
14     {
15         currentCount++;
16     }
17 }

```

Listing 2.3: Counter Beispiel

Hierbei ist noch zu sagen, dass sich der Code für die Methode "IncrementCount()" nicht im selben File wie die Oberfläche befinden muss, diese kann für bessere Trennung von Oberfläche und Code - Behind auch ausgelagert werden (siehe 4.2.2 Code- & Filestruktur).

2.1.3 Blazor und Webassembly

Ein großes Ziel von Blazor ist es, die mittlerweile sehr leistungsfähigen Ressourcen der verschiedenen Browser auszunutzen, um bereits vorhandene C#-Anwendungen nahezu reibungslos in den Webbereich zu bringen. Hier vermeidet Microsoft jegliche Umwege über JavaScript und übersetzt den Blazor-Code direkt in Webassembly. Webassembly ist Bytecode, der direkt im Browser aufgeführt werden kann und wurde entwickelt, um Performance von anderen Websprachen zu optimieren und die Benutzererfahrung durch schnelleres Laden von Websites zu verbessern. Für Blazor war dies das Tor zum clientseitigen Ausführen von Applikationen, anstatt diese auf einem Server arbeiten zu lassen und Daten mit dem Browser auszutauschen wie bisher.

Die clientseitige Entwicklung von Blazor-Anwendungen steht bis jetzt noch in den Kinderschuhen, man rechnet hier im ersten Halbjahr 2020 mit dem ersten Full-Release. Seit September 2019 kann man jedoch schon mit Preview Versionen der Blazor-Webassembly arbeiten. Gemeinsam mit dieser Preview wurde auch die Serverside Variante veröffentlicht, diese wird von Visual Studio 2019 schon vollständig unterstützt mit dazugehörigem Debugging, welches für Webassembly Blazor bislang noch nicht vorhanden ist. Syntaktisch weisen die beiden Varianten kaum Unterschiede auf, weshalb entwickelte Serverside-Anwendungen bei Release nahezu reibungslos in Client-Anwendungen konvertiert werden können.

2.2 ASP.NET Core^{JB}

ASP.NET Core^[2] ist ein plattformübergreifendes Open-Source-Framework von Microsoft zur Entwicklung von performanceoptimierten Webanwendungen. Auch Blazor basiert auf ASP.NET Core, was man an den übernommenen Features wie schnelle Testbarkeit merkt. DOTNET Core ist prinzipiell ein leistungsorientierter Teil der .NET Plattform mit Fokus auf Modernisierung vor allem im Webbereich. Mit dem 2019 veröffentlichten .NET Core 3.0 kam auch der Release von Blazor, welches auf diesem basiert, um als leistungsorientiertes Framework in der Webentwicklung zu strahlen. Zwei Monate später, Ende 2019, wurde .NET Core 3.1 als Stabilitätsverbesserung veröffentlicht, in welchem die derzeitige Version des Universal SmartHomes arbeitet.

2.3 SignalR^{JB}

SignalR^[3] ist Teil des ASP.NET Frameworks und wird verwendet, um Livedaten anzuzeigen und dadurch Echtzeit - Applikationen zu erstellen. Von Chatrooms über Liveticker, bis hin zur Synchronisierung mehrerer Websites ist alles möglich. Letzteres wird beim Universal SmartHome verwendet, um mehreren Anwendern zu jedem Zeitpunkt die absolut gleichen Daten anzuzeigen, um keinen veralteten Zustand von Geräten anzuzeigen, was möglicherweise Probleme aufwerfen kann. Mehr dazu im Punkt *4.2.17 Live Daten mit SignalR*.

2.4 Newtonsoft JSON^{JB}

Newtonsoft^[4] stellt eine .NET Erweiterung zur Verfügung zum Serialisieren und Deserialisieren von JavaScript Object Notation (Siehe *4.2.4 JSON Codierung*).

2.5 Bootstrap^{RB}

Ein großer Schritt in der Gestaltung von Webseiten war die Entwicklung von CSS (Cascading Style Sheets). Mit dieser Stylesheet-Sprache ist es möglich, aus einfachen HTML-Elementen dynamische und gut aussehende Bausteine zu kreieren. Grundlegende Elemente wie Tabellen, Knöpfe, Dropdown-Menüs oder Eingabeformulare musste man sich hier jedoch jedes Mal aufs Neue ausschmücken oder aus alten Projekten kopieren. Aus diesem Grund entwickelte Twitter ein Framework namens Bootstrap^[5]. In diesem Framework sind die gängigsten HTML-Elemente bereits voll dynamisch und ansprechend fürs Auge definiert. Mit einfacher Zuweisung bestimmter Eigenschaften über das *class* - Attribut bekommen die sonst statischen Bausteine verschiedene Stile und Funktionalitäten zugewiesen. Buttons können beispielsweise mithilfe der *.disabled* Klasse als deaktiviert erscheinen. So ändern sie mit nur einem Schlüsselwort ihre Funktionalität sowie auch ihr Aussehen.



Dadurch ist es möglich, diese einfachen Bausteine ohne großen Aufwand in seine Webanwendung einzubinden. Diese sind zwar einheitlich vordefiniert, jedoch über CSS frei gestaltbar, wie jedes andere HTML-Element. Man findet hier auf der offiziellen Bootstrap Website Übersichten der verfügbaren Klassen, sowie eine einfache Anleitung, wie man diese verwenden kann.

Displays für internetfähige Geräte gibt es mittlerweile in allen verschiedenen Größen und Formen, in der Webentwicklung wird hier nun ein dynamisches Design der Oberfläche benötigt. Bootstrap ermöglicht, Elemente in ein Raster zu geben, welches je nach Auflösung skaliert wird. Zusätzlich gibt es eigene Längeneinheiten, diese stehen in Abhängigkeit zu der Bildschirmgröße und skalieren mit der Größe des Anzeigebereichs. Kombiniert man diese Aspekte geschickt, kann man eine Oberfläche entwickeln, welche auf verschiedenen Displays einigermaßen gleich ansprechend aussieht.

2.5.1 Layout

Das Raster ist der Grundstein von Bootstrap, dieses wird in 12 Spalten eingeteilt, welche mittels einer CSS-Flexbox realisiert sind.

```
1 <div class="container">
2   <div class="row">
3     <div class="col-sm">
4       One of three columns
5     </div>
6     <div class="col-sm">
7       One of three columns
8     </div>
9     <div class="col-sm">
10      One of three columns
11    </div>
12  </div>
13 </div>
```

Listing 2.4: Bootstrap-Raster

Die Einteilung kann nun für die ganze Website geschehen oder auch nur für einen Teil davon. Wie der ganze Code funktioniert wird hier beschrieben:

- *.container* - Diese Klasse wird verwendet um einen dynamischen Bereich innerhalb der Website zu definieren. Die Breite des Containers ist jedoch nicht 100% des möglichen Bereichs, sondern nur ca. 60%. Will man die volle Breite eines Bereichs verwenden, kann man dies mit der *.container-fluid* Klasse bewerkstelligen.
- *.row* - Diese Klasse ist die "Hülle" um die einzelnen Zellen und gibt an, wo das verwendete Raster anfängt und aufhört.
- *.col* - Diese Klasse kann anhand von 2 Arten definiert werden: Durch die Angabe der Teile, die der verwendete Abschnitt verwenden soll oder durch automatische Aufteilung. Wenn nun wie in dem angeführten Beispiel 3 Spalten vorhanden sind, teilt Bootstrap diese bei der automatischen Aufteilung in jeweils 12/3 Abschnitte auf (4 Teile pro Spalte).

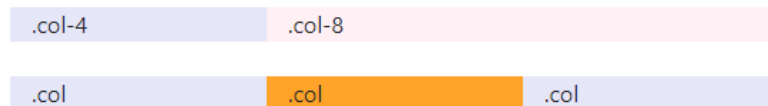


Abbildung 2.1: Bootstrap-Spalten

Bootstrap verwendet gleichzeitig noch sogenannte *responsive breakpoints*. Bei Anwendung wird definiert, ab welcher Displaygröße die Spaltengröße unverändert bleibt. Mit CSS3 wurde diese Funktionalität mittels *Media Queries* eingeführt, welche verschiedene Eigenschaften der verwendeten Geräte abfragen können und auf diese reagieren. Somit ist es möglich, ab gewissen Bildschirmgrößen gänzlich anderen, an die Größe angepassten, Inhalt darzustellen. Die folgende Tabelle zeigt genauere Angaben:

KLASSE	X-SMALL (<576PX)	SMALL (>576PX)	MEDIUM (>767PX)	LARGE (>991PX)	X-LARGE (>1200PX)
<i>.container-sm</i>	100%	540px	720px	960px	1140px
<i>.container-md</i>	100%	100%	720px	960px	1140px
<i>.container-lg</i>	100%	100%	100%	960px	1140px
<i>.container-xl</i>	100%	100%	100%	100%	1140px

2.5.2 Gegenüberstellung

Komponenten, die im einfachen HTML erstellt wurden, haben oft eine feste Pixelgröße. Das führt dann jedoch beim Anzeigen auf verschiedenen Displays zu unschönen Ergebnissen. Bootstrap schafft Abhilfe mithilfe von Viewport-Größen, diese sind abhängig von Höhe und Breite des Anzeigefensters. Gleichzeitig wird durch Verwendung des bereits erwähnten Rasters, die Website eingeteilt und dynamisch mitskaliert.

Ein weiterer Vorteil, den Bootstrap mit sich bringt, ist die Verwendung von einheitlichen Designs. Jegliche Komponenten, die verwendet werden, wirken modern und farblich abgestimmt. Somit ist eine gute Grundlage für den Designer gegeben, kann aber gleichzeitig auch noch individuell gestaltet werden.

Bootstrap fordert kein neues Lernen einer Sprache. Anhand von einfacher Verwendung des Klassen-Attributs werden Bootstrap-Elemente definiert. Dadurch ist eine sehr einfache Zuweisung von verschiedensten Eigenschaften oder Funktionalitäten möglich. Beispiele dafür wären: Farbe ändern, Ladeanimation hinzufügen, Inhalte zentrieren, Textformatierung (Bold, Italic, Einfärben) oder auch eine Scrollbar hinzufügen. Die komplette Aufführung der Möglichkeiten würde den Rahmen dieser Diplomarbeit sprengen und ist auf der offiziellen Website^[5] genauer aufgeführt.

2.5.3 Komponenten

Wie bereits erwähnt, gibt es eine Vielzahl an Attributen. Das Gleiche gilt auch für die Komponenten, deshalb werden in diesem Abschnitt die wichtigsten genauer erwähnt.

Tabellen werden oft verwendet zum Anzeigen von Daten mit inhaltlichem Zusammenhang. Bei Verwendung der einfachen HTML-Tabellen fällt schnell auf, dass diese sehr schlicht und eintönig gestaltet sind. Bootstrap schafft mit Farbkontrast sowie der Verwendung von Rahmenlinien besser Übersicht. In folgendem Bild wird dies noch verdeutlicht:

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

Abbildung 2.2: Tabellen-Unterschied

Bootstrap-Knöpfe sind mit Animationen sowie verschiedenen Farben ausgestattet. Gleichzeitig kann mithilfe des *.btn-fluid* Attributs dessen Breite auf 100% des Anzeigebereichs eingestellt werden. Ein großer Vorteil in Bootstrap sind sogenannte *.btn-group* Elemente, mittels dieser können mehrere Knöpfe direkt aneinander gesetzt werden.

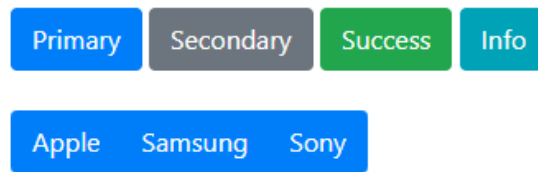


Abbildung 2.3: Bootstrap-Knöpfe

Um einfach thematisch gleiche Auswahlmöglichkeiten anzuzeigen, werden oft Dropdown - Menüs verwendet. Bootstrap verschönert die allgemeine Anzeige und stattet diese zusätzlich mit Animationen aus. In den bereits erwähnten *.btn-group* Elementen können diese ebenfalls hinzugefügt werden.

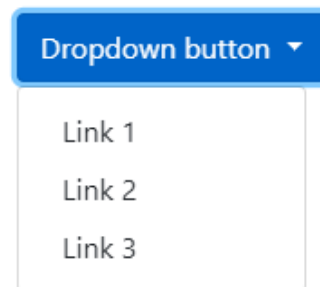


Abbildung 2.4: Bootstrap-Dropdowns

Für die Datenverarbeitung gibt es verschiedene Arten diese einzulesen, wobei in HTML oftmals Forms verwendet werden. Innerhalb einer *.form-grp* werden die verschiedenen Eingaben definiert: Textfelder, Checkboxes, Dropdown - Menüs, etc. Durch ein ansprechendes, zeitgemäßes Design sind diese für den Benutzer gut anzusehen, da vor allem alles, was nicht nur reiner Text ist, sonst schnell eintönig und veraltet aussieht.

Email:

Password:

☐ Remember me

Abbildung 2.5: Bootstrap-Forms

Eine weitere Komponente, bei der Bootstrap sehr viel Arbeit erspart, sind Animationen. Animationen in CSS zu erstellen ist sehr aufwendig und nach großem Aufwand doch meist nur sehr simpel. Bootstrap schafft hier bei zum Beispiel Ladeanimation schnell Abhilfe. Diese sind auf Websites mittlerweile zum Standard geworden und auch sehr wichtig für das Feedback gegenüber dem Benutzer. In Bootstrap sind diese sehr einfach realisiert anhand der *.spinner-grow* oder *spinner-border* Klasse. Der "grow" - Spinner zeigt eine kleine Blase, die an immer mehr Transparenz gewinnt. Der "border" - Spinner stellt den typischen Ladekreis dar.

2.6 MultiLanguage^{RB}

Englisch ist eine der wichtigsten Sprachen in der heutigen Zeit, jedoch gibt es noch sehr viele Menschen, die nur in ihrer Muttersprache begabt sind. Um eine Webapplikation zu erstellen, die keine Sprachbarrieren aufweist, ist MultiLanguage ein wichtiger Bestandteil in der Entwicklung. Das NuGet-Paket `Toolbelt.Blazor.I18nText`^[6] ist eine Visual Studio Erweiterung, welches die Entwicklung von mehrsprachigen Oberflächen erleichtert.

Jede verwendete Sprache benötigt ein JSON-Dokument. Die Sprachdateien sollen eine gleiche Struktur vorweisen, dass bedeutet der Identifier des jeweiligen Texts ist eindeutig in jedem Dokument vorhanden und der Wert gibt den Text in der jeweiligen Sprache an. Das Tool wird dann als Service hinzugefügt und in die jeweiligen Oberfläche *injected* (siehe 4.2.7 *Inheritance & Dependency Injection*). Statt der Verwendung von einfachen Strings wird nun eine Variable der Klasse MultiLanguage verwendet. Diese besitzt zu jedem Text eine Property mit dem jeweiligen Identifier. Anstatt dieser Properties wird beim Erstellen der Seite der jeweilige übersetzte Text dynamisch je nach Spracheinstellung des Browsers eingefügt.

```

1 {
2     "Active": "Aktiv",
3     "Add": "Hinzufügen",
4     "Add_Augmented_IO": "Augmented I/O hinzufügen!",
5     "Add_Camera": "Kamera hinzufügen!",
6     "Add_Room": "Raum hinzufügen!",
7     "Add_Rule": "Regel hinzufügen!",
8     "Address": "Adresse",
9     "Analog_Input": "Analoger Eingang",
10    "Analog_Inputs": "Analoge Eingänge",
11    "Analog_Input_ID": "ID Analoger Eingang",
12    "Analog_Output": "Analoger Ausgang",
13    "Analog_Outputs": "Analoge Ausgänge",
14    "Analog_Output_ID": "ID Analoger Ausgang",
15
16    ...
17 }
```

Listing 2.5: MultiLanguage JSON-Datei Deutsch

```

1 {
2   "Active": "Active",
3   "Add": "Add",
4   "Add_Augmented_IO": "Add Augmented I/O!",
5   "Add_Camera": "Add Camera!",
6   "Add_Room": "Add Room!",
7   "Add_Rule": "Add Rule!",
8   "Address": "Address",
9   "Analog_Input": "Analog Input",
10  "Analog_Inputs": "Analog Inputs",
11  "Analog_Input_ID": "Analog Input ID",
12  "Analog_Output": "Analog Output",
13  "Analog_Outputs": "Analog Outputs",
14  "Analog_Output_ID": "Analog Output ID",
15
16  ...
17 }

```

Listing 2.6: MultiLanguage JSON-Datei Englisch

```

1 @if (loading_done == false)
2 {
3   <h1>@Globals.MultiLanguage.Home</h1>
4   <div class="container text-center">
5     <div class="spinner-grow text-primary"></div>
6     <div class="spinner-grow text-primary"></div>
7     <div class="spinner-grow text-primary"></div>
8   </div>
9 }
10 else
11 {
12   <div class="row">
13     <div class="col-sm-4">
14       <h1>@Globals.MultiLanguage.Home</h1>
15     </div>
16
17     ...
18   </div>
19 }

```

Listing 2.7: MultiLanguage im Code

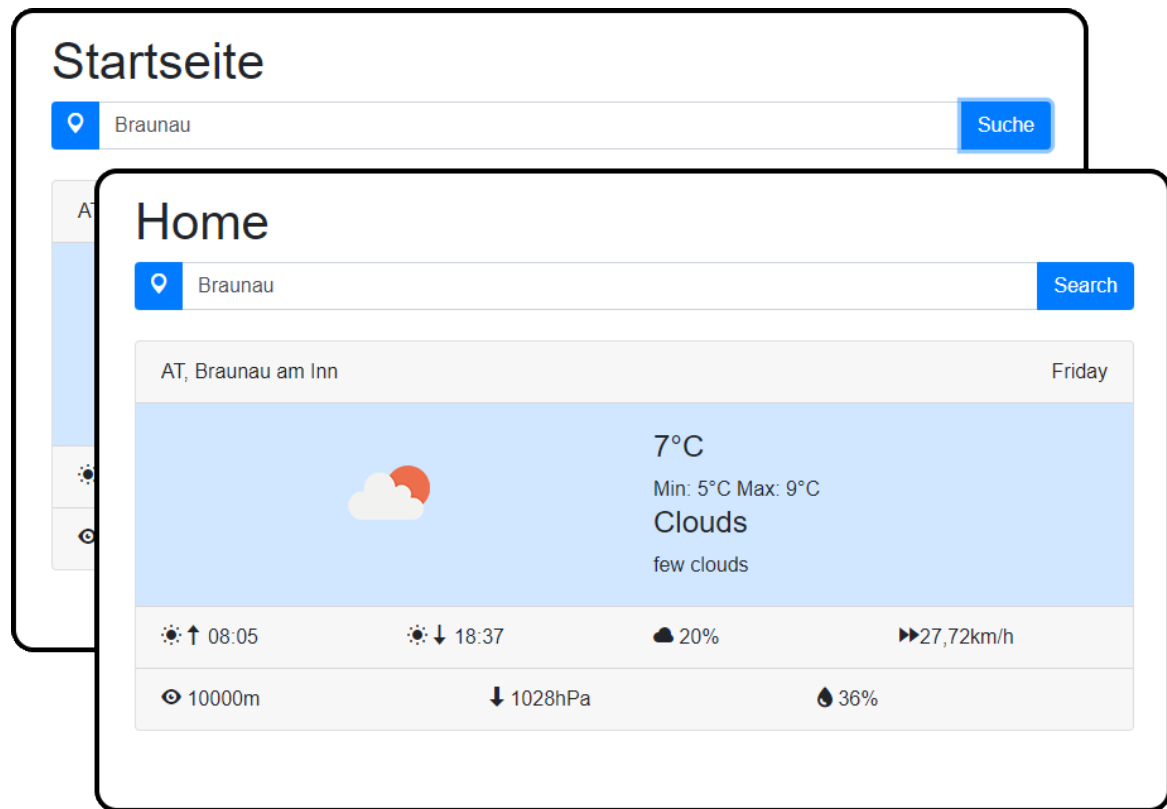


Abbildung 2.6: Multilanguage Vergleich

2.7 Toasts^{RB}

Das NuGet-Paket Blazored.Toasts^[7] ist eine Erweiterung für Visual Studio von *Chris Sainty*. Diese dient zur einfachen Darstellung von Benachrichtigungen. Vier verschiedene Arten von Toasts sind möglich: Fehler, Infos, Erfolg und Warnungen. Mit Hilfe der Klasse *ToastService* wird eine Variable erstellt, welche vier Methoden, eine pro Typ (*showInfo*, *showError*, ...), zur Verfügung stellt, um eine Nachricht anzuzeigen. Um diese Variable verwenden zu können, muss das Tool als Service hinzugefügt werden, danach implementiert man *Blazored.Toast* und *Blazored.Toast.Services* im Code. Im *MainLayout* des Blazor-Projekts kann man die Toasts je nach Anliegen konfigurieren. Für die richtige Formatierung benötigt man noch das richtige CSS, dieses wird in der Datei *_Host.cshtml* (ServerSide) oder *index.html* (ClientSide) hinzugefügt. Will man nun einen Toast anzeigen, ruft man von der eingebundenen Variable eine der vier Funktionen auf, wobei man den Text als Parameter übergeben kann.

3 Benutzeroberfläche^{RB}

Das Projekt "SmartHome" gibt es schon sehr lange an der HTL-Braunau, dadurch waren bestimmte Aspekte der Arbeit schon dargelegt. Diese Projekte sind alle abgeschlossen, jedoch ist in der heutigen Zeit ein Produkt schon nach einigen Wochen nicht mehr auf den neusten Stand. Neuerungen in Soft- oder Hardware kommen rapide, gleichzeitig gibt es immer mehr Menschen, die in diese Welt eintauchen. Das bedeutet gleichzeitig verschiedenste Benutzer mit allerlei Anforderungen. Aus diesem Grund war das alte Projekt weniger eine Grundlage, sondern eher eine Möglichkeit zur Verbesserung und Neugestaltung.

3.1 Konzept

Beim Erstellen einer Benutzeroberfläche ist ein Konzept ein wichtiger Bestandteil. Man hat so immer ein Ziel vor Augen, kann jedoch noch im Verlauf des Projekts weitere Ideen einbringen.

3.1.1 Erstes Design

Das erste Design hilft dabei, ein Grundgerüst für die Oberfläche zu erstellen. Die ersten Ideen wurden hierbei noch auf Papier skizziert, das Konzept sah wie folgt aus:

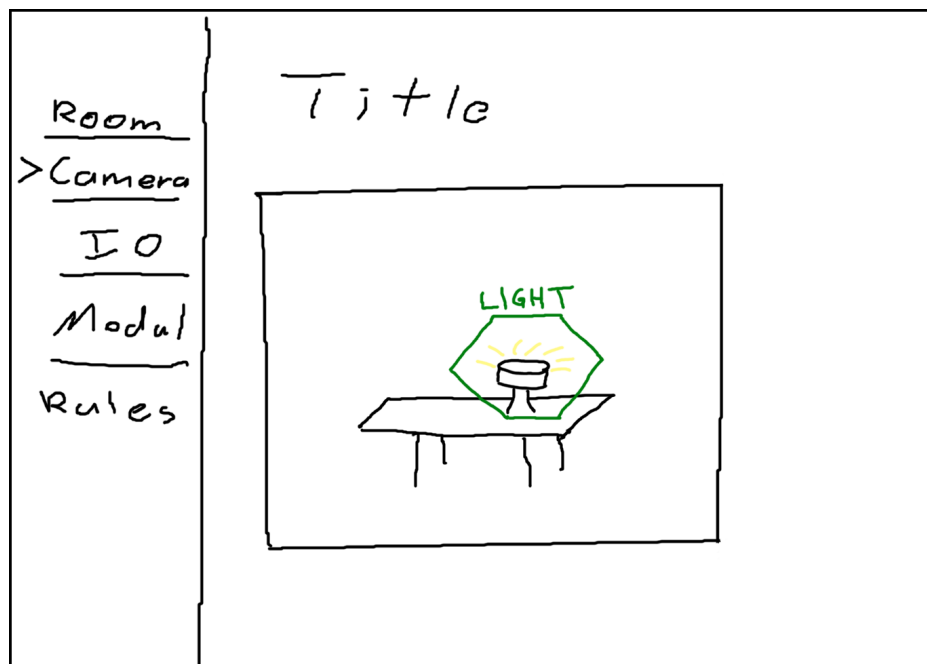


Abbildung 3.1: Erstes Konzept

Wie man auf dem Bild erkennen kann, soll es eine Navigationsleiste geben, mit der das einfache Wechseln zwischen Bereichen möglich ist. Die verschiedenen Abschnitte sollen die Möglichkeit bieten, das SmartHomes zu steuern, sei es über den Raum direkt oder sogar über eine Kamera. Das Augmented Reality Prinzip soll hierbei für die Steuerung der einzelnen Geräte per Kamera verwendet werden.

3.1.2 Finales Design

Im Laufe des Projekts gab es viele Ideen zum Design der Oberfläche, ein Großteil davon wurde auch umgesetzt. Eine Startseite mit Wettervorhersage und eine Spritpreisübersicht waren ein Teil davon. Vieles aus dem ersten Konzept wurde mitgenommen. Die Navigationsleiste auf der linken Seite, die Einteilung in Räume, Kameras, I/O's, Module und Regeln wurden so übernommen. Das Augmented Reality Prinzip wurde oft überarbeitet, aber schaffte es zum Abschluss noch in das Projekt. Im weiteren Verlauf der Diplomarbeit wird auf die einzelnen Elemente der Oberfläche noch genauer eingegangen.

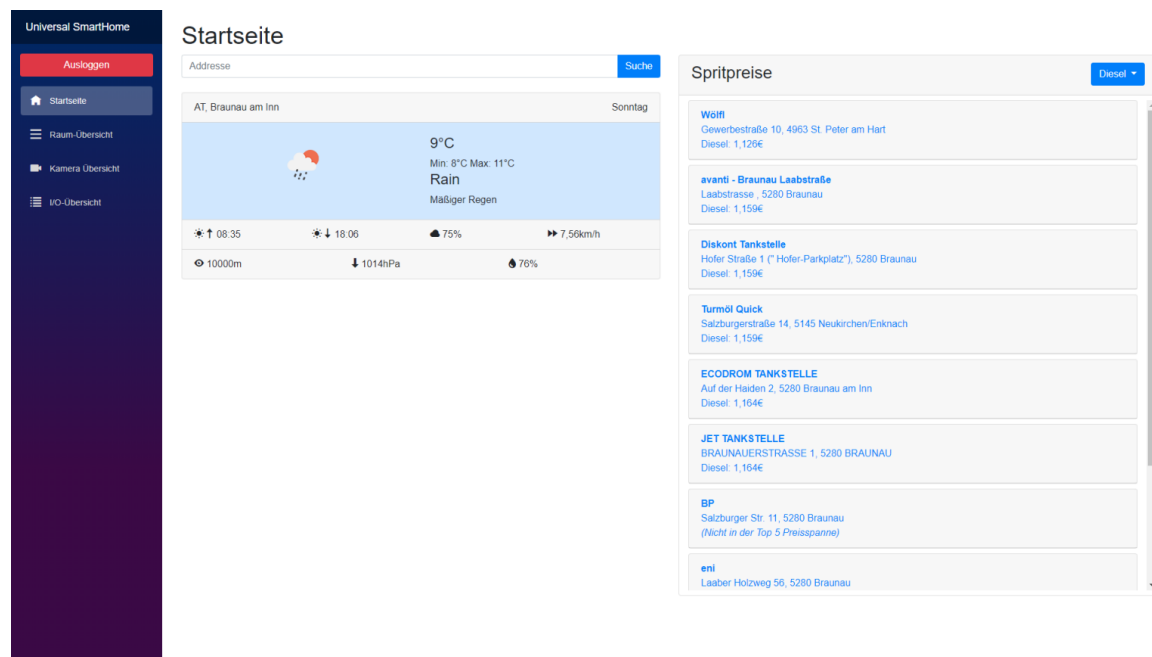


Abbildung 3.2: Startseite

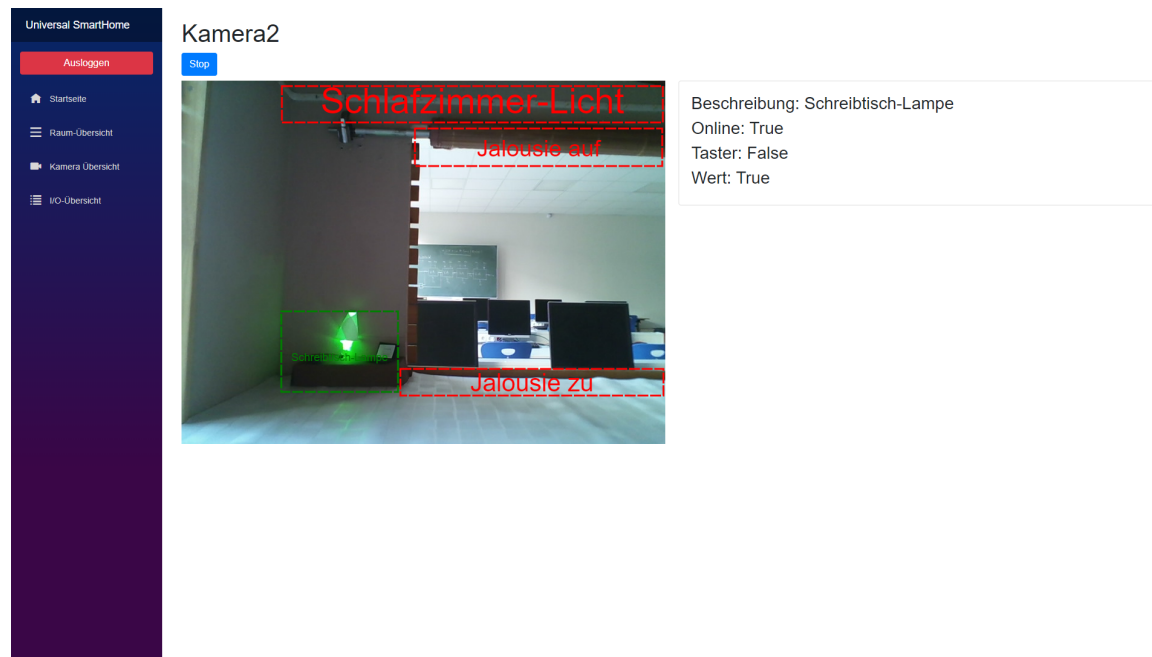


Abbildung 3.3: Kamera Website

3.2 Design und Ablauf

3.2.1 Blazor Vorlage

Blazor erzeugt beim Erstellen einer Anwendung ein Vorlage, diese Grundstruktur hilft zu Beginn herauszufinden, wie genau nun Blazor die zwei Sprachen HTML und C# zusammenführt. Die Applikation ist in drei verschiedene Teilen geteilt: Home, Counter und Fetch data. Diese Abschnitte sind mithilfe einer Navigationsleiste einfach zu erreichen. Jede Seite beinhaltet eine Überschrift und einen kleinen Text zur Beschreibung des Inhalts. Die Home Seite hat keine besonderen Funktionen und dient nur zum besseren Verständnis. *Counter* hat als Besonderheit einen Knopf, der eine Variable hochzählen lässt. Dem Bedienelement wird hierbei eine C#-Methode zugewiesen. In *Fetch data* werden Informationen angezeigt, die erst nach Abschluss des Ladevorgangs sichtbar sind, dies soll das Laden von externen Daten simulieren.

Das Navigieren zwischen Seiten, zuweisen von C#-Methoden oder auch das Laden und Anzeigen von Daten ist eine gute Basis für einen Teil der Aufgaben. Die Vorlage verwendet Bootstrap für die Benutzeroberfläche, wie zum Beispiel die Navigationsleiste, den Knopf oder auch die Tabelle. Bootstrap, ein starkes Tool in der Webentwicklung, ist ein wichtiger Bestandteil des Projekts und ein Muss für jede fortschrittliche Applikation.

3.2.2 Stil festlegen (Bootstrap Einarbeitung)

Um eine Website zu gestalten, gibt es allerlei Möglichkeiten dies zu realisieren, man kann zum Beispiel mithilfe von reinem CSS das Aussehen und Verhalten jedes Elements selbst definieren. Dieser Prozess ist jedoch sehr mühselig und auch eher veraltet. Für diesen Zweck werden mittlerweile Frontend-Frameworks verwendet. Diese unterscheiden sich im Stil oder auch in der Sprache zum Beispiel: Angular basiert auf TypeScript, Bootstrap jedoch auf CSS.

Für welches man sich nun entscheidet, liegt am Benutzer selbst, in diesem Projekt wurde Bootstrap verwendet. Die Beweggründe dafür waren die bereits vorhandene Erfahrung und gleichzeitig verwendet Blazor dieses Framework schon in der Vorlage. Im Internet gibt es viele Informationen zu Bootstrap, die zwei wichtigsten Webseiten hierbei sind die offizielle Website^[5] sowie w3schools^[8].

3.2.3 Login Seite

Die Universal SmartHome API für die Steuerung der verschiedenen Modulen benötigt eine Authentifizierung. Für Testzwecke gibt es zwei verschiedene Benutzer, diese sollen den Unterschied zwischen Admin und User darstellen und tragen auch diesen Namen (Siehe 3.2.4 *Unterschied Admin/User*). Nach der Anmeldung bekommt der jeweilige Benutzer ein Token, der 30 Minuten lang aktiv ist und danach wieder neu generiert werden muss. Dieser Prozess war der erste Schritt in der Entwicklung der Oberfläche. Bevor nun die eigentlichen Funktionen der Webapplikation verwendbar sind, soll eine Log-in - Seite angezeigt werden.

Die Anmeldung wurde mit Hilfe von verschiedenen Bootstrap-Elementen realisiert. Der Benutzername und das Passwort werden jeweils über ein Textfeld eingegeben, diese Felder bestehen aus einer Form-Group, die ein Input-Element beinhalten. Um nun die Eingabe auslesen zu können, wird das Element auf eine string - Variable gebunden (siehe 4.2.5 *Databinding*). Die Log-in - Daten können abgespeichert werden anhand einer Checkbox, in HTML wird hierbei die gleiche Struktur wie bei den Textfeldern verwendet. Zusätzlich wird dem Input-Element die Klasseneigenschaft *form-check-input* zugewiesen, ein weiterer Unterschied ist das Databinding auf eine boolesche Variable.

Die Eingabe der Anmeldedaten wird mit einem Button bestätigt, die korrespondierende C#-Methode zum Einloggen bekommt die Zuweisung auf das *@onclick* - Event. Bei falscher Eingabe wird mithilfe eines Toasts angezeigt, dass die Daten fehlerhaft sind. Gibt es keine Komplikationen wird weitergeleitet auf die Startseite.

3.2.4 Unterschied Admin/User

Die Unterteilung von Benutzer und Administrator ist ein wichtiger Bestandteil des SmartHomes. Beim ersten Einrichten werden zum Beispiel die verschiedenen Räume erstellt, die Ports der Module zugewiesen oder die Augmented Reality - Ansicht der Kamera aufgesetzt (siehe 7 *Bedienungsanleitung*^{RB}). Die erwähnten Aufgaben sind der Rolle des Admins zugewiesen. Das Aussehen der Benutzeroberfläche wird daher stark beeinflusst durch den Unterschied Admin/User.

Der Admin kann jegliche Elemente erstellen, bearbeiten oder auch löschen. Weitere Features sind eine Übersicht der verschiedenen Hardware - Module als auch der erstellten Regeln (z.B. Zeitschaltungen). Abschnitte, die der Benutzer erreichen kann, werden soweit vereinfacht, dass nur mehr die Steuerung und nicht das Bearbeiten der Geräte möglich ist. Diese sind die Startseite, Raum-Übersicht, Kamera-Übersicht und I/O-Übersicht. In diesen Abschnitten hat ein Admin wie erwähnt verschiedenste Verwaltungsmöglichkeiten, auch werden wichtige zusätzliche Informationen angezeigt. Im Fall der Kamera sind das zum Beispiel: ID, Model, IP, Status oder auch die Auflösung. Die genauen Abweichungen werden in den jeweiligen Abschnitten genauer beschrieben.

3.2.5 Startseite

Die Startseite ist die erste Ansicht nach dem Login, hier werden Informationen angezeigt wie Wetter oder aktuelle Spritpreise. Genauer gesagt gibt es hier 3 Widgets: Aktuelles Wetter, Wettervorhersage und Spritpreise. Jedes dieser Widgets ist in HTML als *card* definiert, wobei die Daten per API - Call geladen werden. Wenn der Benutzer es zulässt, wird der derzeitige Standort automatisch abgefragt, andernfalls ist es möglich, mittels eines Textfelds die Adresse anzugeben. Nach Bestätigung der Suche, werden mittels Wetter API^[9] Daten empfangen. Ist die Adresse jedoch falsch wird eine Fehlermeldung als *Toast* ausgegeben. Die Anzeige der Spritpreise^[10] ist nur auf Österreich beschränkt, somit werden keine Daten geladen, sobald sich der Standort außerhalb befindet.

Das aktuelle Wetter enthält Informationen über Temperatur (Minimal-, Durchschnitts- und Maximalwert), Beschreibung des Wetterzustands, Sonnenzyklus, Wolkendichte, Windgeschwindigkeit, Sichtweite, Druck und Luftfeuchtigkeit.

Die Wettervorhersage funktioniert sehr ähnlich wie das aktuelle Wetter, der Unterschied ist hier, dass durch das Anzeigen von mehreren Tagen für jeden Tag weniger Informationen angezeigt werden, um den Bildschirm nicht zu überhäuften. Hierbei sieht man für den aktuellen Tag die verschiedenen Temperaturunterschiede, sowie das Wetter und die Temperatur (Minimal-, Durchschnitts- und Maximalwert) der nächsten drei Tage.

Das letzte Widget auf der Startseite ist eine Anzeige für verschiedene Spritpreise. Der Anwender kann zwischen Diesel, Super95 oder auch Erdgas wählen. Die API ermittelt die Preise in der Nähe und gibt eine Liste aus, sortiert nach billigstem Preis. Jedes Element in der Liste ist wieder eine *card*, angezeigt werden die drei wichtigsten Informationen jeder Tankstelle: Name, Adresse und Spritpreis (in Euro). Für weitere Informationen wie Öffnungszeiten oder Selbstbedienung können diese Komponenten aufgeklappt werden.

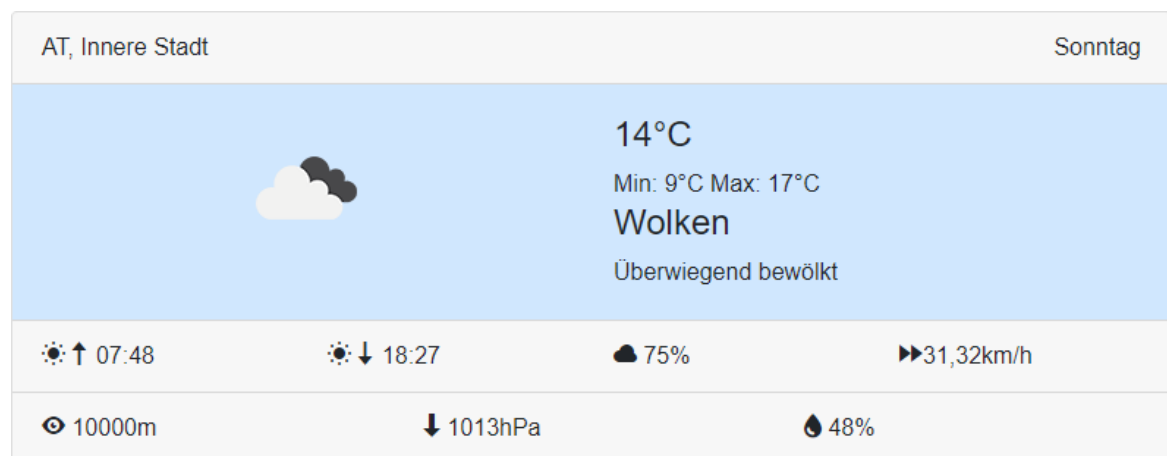


Abbildung 3.4: Wetter Widget

3.2.6 Übersichts-Seiten

Die Steuerung des SmartHomes ist über verschiedenste Ansichten möglich. Der Benutzer hat hierbei die Auswahl, ob er nur innerhalb eines Raums steuern will, mittels einer Kamera oder über die Anzeige aller I/O's einer bestimmten Gattung. Für den Admin gibt es zusätzlich eine Steuerung anhand der einzelnen Module. All diese Möglichkeiten beinhalten eine Vielzahl an Elementen, die eine Visualisierung benötigen. Die Übersichtsseiten dienen zur

Zusammenfassung und Anzeige, erreichbar sind diese mittels der Navigationsleiste. Für jedes Element wird ein Button in HTML erstellt, der auf die jeweilige Seite verlinkt. Als Admin hat man in jeder dieser Übersichtsseiten zusätzliche Verwaltungselemente: Erstellen, Bearbeiten oder Löschen. Zu beachten ist jedoch, dass Module nicht vom Admin erstellt werden. Da es sich bei einem Modul um ein Hardware - Element handelt, kann es nicht durch den Admin erstellt werden, sondern meldet sich sobald es eingeschaltet wird selbstständig beim Home - Server und wird somit automatisch in die Datenbank eingetragen, wodurch es auch auf der Übersichtsseite aufscheint.

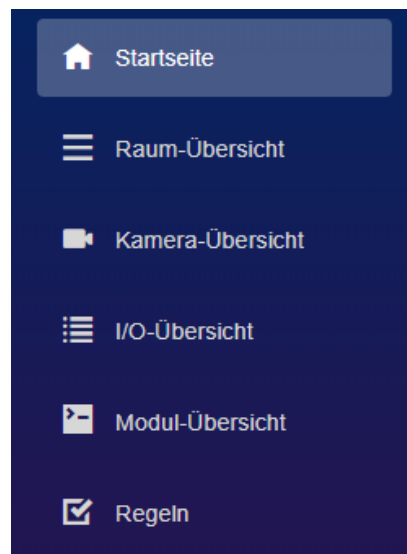


Abbildung 3.5: Navigationsleiste

3.2.7 Raum-Einzelansicht

Die Einzelansicht der Räume vereinfacht die Steuerung von angeschlossenen Geräten. Nach Einteilung der Räume und der zugehörigen Modul-Ports ist es dem Benutzer möglich, diese zu bedienen. Hierbei wird unterschieden zwischen Analog oder Digital und ob es ein Eingang oder Ausgang ist. Digitale Ausgänge können zusätzlich als Schalter oder Taster gesehen werden, ein Taster besitzt hier noch optional eine Pulsdauer, diese definiert die Dauer, wie lange der Taster "gedrückt" bleibt.

Für die Benutzeransicht ist ein Taster als einzelner Knopf realisiert, ein regulärer Schalter besteht aus einer *button-group* mit zwei Knöpfen jeweils zum Ein- und Ausschalten. Als Administrator sieht man für jeden Ausgang beide Steuermöglichkeiten, gleichzeitig können die verschiedenen I/O's bearbeitet oder aus dem Raum gelöscht werden.

3.2.8 Kamera/Augmented Reality

Dieser Teil der Diplomarbeit ist die neuste Erweiterung des SmartHome-Projekts und war gleichzeitig auch die aufwendigste. Elektronische Geräte, die sich innerhalb des Kamerabilds befinden werden mittels Augmented Reality gesteuert. Aus dem Englischen übersetzt bedeutet das so viel wie erweiterte Realität. Einfach gesagt vermischt man digitale Informationen mit der Wirklichkeit, um zusammen mit einem Echtzeitbild wichtige Daten anzuzeigen. Ein simples Beispiel hierfür ist das Aufnehmen von Fotos mit einer Handykamera. Wenn ein

Gesicht in das Aufnahmebild läuft, wird es durch ein Rechteck umrahmt und wodurch die Realität auf dem Bildschirm mit digitalen Mitteln erweitert wurde.

In unserem Projekt wird das Bild mit einer IP-Kamera erzeugt. Das Model ist hierbei frei wählbar, nach der Konfiguration erscheint das Gerät im System. Die Verwaltungstools spielen hier eine wichtige Rolle. Der Admin definiert einen Bereich für ein Steuerelement, in dem es sichtbar ist. Sind alle AR-Rahmen festgelegt, kann der Benutzer die Geräte, die dahinter liegen, mittels einfachen Mausklicks steuern. Das Erstellen und Löschen wird mithilfe von Dropdown - Menüs erleichtert.



Abbildung 3.6: Kamera-Ansicht

3.2.9 I/O-Liste einer Gattung

Jedes Modul besitzt vier verschiedene Signaltypen, diese werden unterschieden zwischen Eingang oder Ausgang. Gleichzeitig kann es sich bei jedem I/O um entweder analoge oder digitale Werte handeln. Somit werden die I/O's aufgeteilt in: analoge Ausgänge, analoge Eingänge, digitale Ausgänge und digitale Eingänge.

Hat man eine der vier Gattungen gewählt, wird man auf die zutreffende Seite weitergeleitet. Alle Steuerelemente werden alphabetisch sortiert aufgelistet, wenn ein Gerät jedoch offline ist, ist die Steuerung deaktiviert. Der Aufbau ist somit strukturell für jede der vier Gattungen gleich, wodurch für den Aufbau der Seite der selbe Code verwendet werden kann. Die jeweiligen Elemente an sich werden hier je nach Typ verschieden angezeigt. Ein digitaler Eingang zeigt hierbei nur seinen Zustand (True oder False / Ein oder Aus) an. Der dazugehörige digitale Ausgang kann mittels zwei verschiedene Arten gesteuert zu werden: Ein- oder Ausschalter beziehungsweise als Taster mit definierter Zeitspanne. Analoge Ein- oder Ausgänge beschränken sich nur auf einen bestimmten Signalwert, der Eingang stellt diesen auf zwei Kommastellen genau dar. Der Ausgang bekommt einen Minimal- und Maximalwert zugeteilt und kann mit einem Slider eingestellt werden.

3.2.10 Modul-Einzelansicht

Diese Seite ist nur für den Admin verfügbar, gleichzeitig aber auch ein wichtiger Teil in der Konfiguration. Wenn ein Modul sich innerhalb des Systems gemeldet hat und in der Übersicht erscheint, kann es noch nicht sofort verwendet werden. Die verschiedenen I/O's benötigen Namen sowie Zuweisungen auf einen Raum. Jedes Modul besitzt hierbei zwei digitale Ausgänge und jeweils ein I/O der anderen Gattungen. Diese werden aufgelistet und können mittels Verwaltungstools bearbeitet werden.

3.2.11 SmartHome Regeln einrichten

Regeln sind ein weiteres Tool des Administrators. Diese dienen zum zeitlichen Steuern von Geräten sowie dem logischen Verknüpfen von Geräten. Zum Beispiel kann definiert werden, dass zum Sonnenaufgang die Jalousien hochgefahren werden sollen oder wenn ein Temperatursensor eine gewisse Temperatur feststellt, die Heizung automatisch ausgeschaltet wird. Es werden alle Regeln nacheinander aufgelistet (sortiert nach ID) und angezeigt, ob diese gerade aktiv oder inaktiv sind. Verwaltungstools sind hierbei das Erstellen, Löschen oder auch Bearbeiten. Wie diese genau definiert sind, wird in der Bedienungsanleitung beschrieben (siehe 7 *Bedienungsanleitung*^{RB}).

Regeln

#	Regel	Aktiv	
7	EVENT "Labor Waser#LED" = "Labor Waser#Tor" && "Sonstiges#M3_DigIn"	False	 
17	EVENT "Labor Waser#Glühbirne" = ! "Labor Waser#Tor" && "Sonstiges#M2_AnIn" > 7	False	 
18	TIMER "Wohnzimmer#Wohnzimmer-Licht" = 15:17#Mo#Di#Mi#Do#Fr#Sa#So 15:18#Mo#Di#Mi#Do#Fr#Sa#So	False	 
27	NOTIFICATION "Das ist eine Testnachricht" = "Labor Waser#Tor" && "Sonstiges#M3_DigIn"	False	 
1014	EVENT "Labor Waser#LED" = "Labor Waser#Tor" < 2	False	 
2020	EVENT "Außenbereich#Außenbeleuchtung" = "Außenbereich#Klingel"	True	 
2021	TIMER "Wohnzimmer#Wohnzimmer-Licht" = 10:30#Mo#Di#Mi#Do#Fr#Sa#So 10:45#Mo#Di#Mi#Do#Fr#Sa#So	True	 
2023	TIMER "Schlafzimmer#Schlafzimmer-Licht" = 10:30#Mo#Di#Mi#Do#Fr#Sa#So 10:45#Mo#Di#Mi#Do#Fr#Sa#So	True	 

Abbildung 3.7: Übersicht der Regeln

3.2.12 Rendering

Eine moderne Webapplikation bedeutet auch das automatische Anzeigen aktueller Daten. Für die Benutzeroberfläche ist der Begriff *Rendering* hierbei sehr wichtig. *Rendern* kann "übersetzen" oder auch "wiedergeben" bedeuten, im Falle des User-Interface handelt es sich um das Laden eines Abschnitts oder der ganzen Website. Bei wiederholtem Ladevorgang spricht man von *Rerendering*.

Der erste Versuch Daten dynamisch anzuzeigen war mithilfe eines *RenderFragments*. Hierbei wird innerhalb des Razor-Files eine *RenderBody* definiert, dieser fungiert als Platzhalter für die eigentlichen Daten. Im Code-File wird anhand eines Builders das HTML-Element definiert, welches den *RenderBody* ersetzen soll. Der ganze Vorgang erweist sich jedoch als sehr mühselig und die Struktur war unübersichtlich.

Die aktuelle Methode zum Aktualisieren von Daten ist nun übersichtlich und einfach in der Anwendung. Wenn neue Daten zum Anzeigen verfügbar sind, wird innerhalb des Code-Files die Funktion *base.StateHasChanged()* aufgerufen. Dadurch wird das *Rerendern* signalisiert und durchgeführt. Diese Methode ist eine der vielen Lifecycle - Methoden von Blazor (siehe 4.2.10 *Blazor Lifecycle Methoden*).

3.3 Fehlerbehandlung graphisch

Um den Benutzer über mögliche fehlerhafte Eingaben oder auch systematische Fehler zu informieren, werden Toasts verwendet (siehe 2.7 *Toasts^{RB}*). Diese dienen neben dem Bekanngeben von aufgetretenen Fehlern auch zum Anzeigen verschiedenster Informationen. Im Falle der Fehlerbehandlung, zum Beispiel bei falschen Eingaben oder auch Verbindungsproblemen, wird hier eine Meldung am Bildschirm ausgegeben. Dem Benutzer wird damit deutlich gezeigt, ob er selbst einen Fehler gemacht hat oder ob das System daran schuld ist. Gleichzeitig informieren die Toasts den Benutzer wenn er zum Beispiel einen Taster erfolgreich schaltet, da dieser bei bestimmten Ansichten ansonsten kein visuelles Feedback hat.

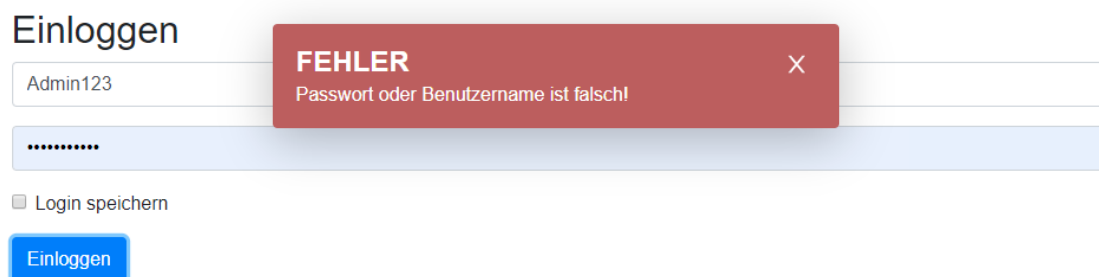


Abbildung 3.8: Fehler-Toast

4 Implementierung^{JB}

4.1 Software-Architektur

Nachdem die Anforderungen an das Projekt klar waren, ging es an die Entwicklung einer Software-Architektur für die Datenverarbeitung.

4.1.1 Erstes Design

Sehr früh war klar, dass bei einem Projekt in diesem Umfang Code gut durchdacht und sinnvoll gekapselt sein sollte, um von Anfang bis Ende den Überblick zu behalten. Die erste Idee war, für jede sichtbare Webpage ein dazugehöriges Code-File zu erstellen, welches die gesamte Logik sowie die angezeigten Daten dieser Page beinhaltet. Dies war naheliegend, da in Blazor jede Webpage von einem zu Grunde liegendem Code-File abgeleitet werden kann (siehe *4.2.7 Inheritance & Dependency Injection*).

Schnell wurde jedoch klar, dass Funktionalitäten die eine Seite besitzt, eine andere Seite ähnlich, oder teilweise genau so auch benötigt. Letztendlich wurde die Struktur zu einer leicht anderen Kapselung umgebaut.

4.1.2 Finales Design

Das Finale Design wurde um zwei C# - Files erweitert, um gemeinsames Verhalten von Webpages besser abzubilden. Zum einen wurde ein strikt statisch gehaltenes File hinzugefügt, um vor allem unveränderliche Daten wie URLs von APIs festzulegen. Zum anderen wurde die Kommunikation zum Server und verschiedenen APIs in ein gemeinsames File mit statischen Methoden ausgelagert. Da mehrere Seiten oft ähnliche oder auch gleiche Daten abfragen und anzeigen, macht hier eine logisch zusammengehörige Kapselung mehr Sinn als eine sichtbare. So ist letztendlich der Zugriff von den Blazor-Seiten auf die jeweiligen eigenen Daten sehr intuitiv durch Ableitung gegeben, und die Kommunikation mit dem Server über eine gemeinsame Schnittstelle sehr einfach gehalten.

4.2 Design und Ablauf

4.2.1 Blazor Vorlage

Einen guten Einstieg in die Entwicklung mit Blazor liefert die Vorlage, die beim Erstellen einer Anwendung vorliegt. In dieser findet man grundlegende Funktionalitäten wie das Navigieren zwischen verschiedenen Seiten, oder auch den Einsatz von C# Syntax inmitten von HTML-Seiten vor (siehe 2.1.2 *Einführung in Blazor Projekte*). Für die Erstellung einer Struktur für den *Code-Behind* ging es daran zu sehen, was die Vorlage vorschlägt, aber auch was sonst noch möglich ist. Schnell fällt hier auf, dass sich die meisten Funktionalitäten auf einer einzigen Seite befinden und wenig Trennung vorliegt, es wurde eher Wert auf den schnellen und vor allem bequemen Syntax gelegt.

Was die grundlegende Struktur einer Webpage, die Anwendungslogik zum Starten der Seite, oder Error-Handling bei falschen Adressen anbelangt, gibt die Vorlage bereits alles Nötige vor. Dies erweist sich als sehr nützlich, da hier sonst gut und gerne einige Stunden in die Implementation solcher Funktionalitäten hineinfließen können. Auch wird ein Beispiel einer Navigationsleiste gezeigt, welches für Anwendungen mit mehreren Seiten gut übernommen werden kann. Alles in allem bietet Microsoft hier einen einfachen und schnellen Einstieg in die Entwicklung mit Blazor.

4.2.2 Code- & Filestruktur

Um den Überblick über alle Ressourcen des Projekts zu behalten ist gutes Filemanagement nötig. Hier hält man sich am besten an gewohnte Konventionen und fasst Dateien mit gleichen Endungen in Ordnern zusammen. Ab einer gewissen Anzahl sollte man diese auch noch logisch kapseln mit Unterordnern oder auch verschiedene Namespaces, wie es sich in C# anbietet. Beim Universal SmartHome sind hier Razor-Files (Webpages), Code-Behind, JavaScript und Stylesheets in jeweils eigenen Ordnern. Weiters sind die Razor-Files noch unterteilt in wirkliche Seiten und Layouts oder Templates.

Was den Code an sich betrifft, existiert für jede sichtbare Webpage ein dazugehöriges Code-Behind-File. So ist sichergestellt, dass auf die anzuzeigenden Daten intuitiv zugegriffen wird und keine fehlerhaften Überschneidungen auftreten. Zu Grunde liegt dem das Blazor System der Ableitung (Inheritance), hier kann nämlich jeweils ein Razor-File von einem C#-File abgeleitet werden, um dessen Eigenschaften und Methoden zu verwenden, ohne darauf statisch zugreifen zu müssen. Der Vorteil darin besteht, dass all diese Daten gemeinsam mit dem Aufruf der Seite entstehen und so niemals veraltete Daten vom letzten Aufruf übriggeblieben sein können. Die Instanz der abgeleiteten Klasse wird also zusammen mit dem Verlassen der Seite sauber aufgeräumt. Daraus folgt auch, dass in einem objektorientiertem Stil mit den Daten umgegangen werden kann, da die Methoden der Klasse auf die eigenen Member vollen Zugriff haben, ohne statisch

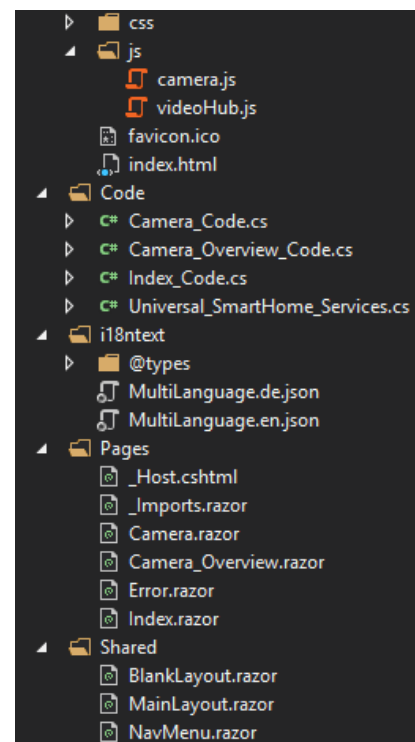


Abbildung 4.1: Filestruktur

deklariert werden zu müssen. Führt man diesen Gedanken weiter, merkt man, dass wenn man die Daten einer Seite behalten will, man wie bei jedem Objekt nur eine Referenz auf dieses behalten muss. So ist es zum Beispiel möglich die gesamte Seite oder auch nur Teile dieser (wie die Navigationsleiste) im Nachhinein zu bearbeiten. Mehr dazu in den Abschnitten 3.2.4 *Unterschied Admin/User* und 4.2.5 *Databinding*.

4.2.3 Universal SmartHome API

Herzstück des Projektes ist die hauseigene Universal SmartHome API, welche eine Schnittstelle für alle relevanten Daten des SmartHomes darstellt. Mithilfe der API werden Daten von der zu Grunde liegenden Datenbank abgefragt und hinzugefügt. Ein Großteil der Daten, wie Raumübersichten, steuerbare Elemente oder Zustände der Geräte kommen hier von der API. Weiters werden für die Steuerung der Geräte verschiedenste API Requests verwendet, welche serverseitig verarbeitet werden und in weiterer Folge die Kommunikation mit den Steuerrelais übernehmen. Die API bietet auch Funktionen, um die Daten des SmartHomes zu bearbeiten, um zum Beispiel einen neuen Raum hinzuzufügen.

Die Kommunikation mit der API erfolgt über HTTP-Methoden wie GET, POST, PUT und DELETE. Alle Daten werden nach HTTP Konvention im JSON-Format übertragen (siehe 4.2.4 *JSON Codierung*). Für die Implementierung der Serverkommunikation wird clientseitig der in .NET Core vorhandene *HttpClient* verwendet. Mit diesem können HTTP-Requests jeglicher Art abgesendet werden.

```
1 public async static Task<List<CRoom>> GetRooms ()
2 {
3     try
4     {
5         client.DefaultRequestHeaders.Authorization =
6             new AuthenticationHeaderValue("bearer", Globals.AccessToken);
7
8         return JArray.Parse(await client.GetStringAsync(Globals.BASE_URL + "api/
9             Rooms?sort=Description")).ToObject<List<CRoom>>();
10    }
11    catch (Exception ex)
12    {
13        Console.WriteLine(ex);
14        new ErrorHandler().HandleException(ex);
15    }
16    return new List<CRoom>();
17 }
```

Listing 4.1: HTTP-Request

Um ein Gefühl dafür zu entwickeln, wie mit diesen API-Calls umzugehen ist, folgen hier einige der häufigsten Abfragen an die Schnittstelle. Dabei ist zu sagen, dass die Kommunikation mit dem Server geschützt abläuft. Das bedeutet, dass man nur mit einem bestimmten, zeitlich begrenztem Schlüssel (Token) berechtigt ist, Daten vom Server abzufragen.

Um diesen Schlüssel zu erhalten, werden zu Beginn der Webanwendung vom Benutzer Einlogdaten abgefragt. Mit diesen Daten wird dann vom Server ein neuer Token verlangt. Der Token aus der Antwort wird für spätere Anfragen als Authentifizierung im *Header* mitgeschickt.

Example Request

```
POST URL/Token
```

Request Header

```
Content-type: application/x-www-form-urlencoded
```

Example Request Body

```
grant_type=password&username=User&password=xxxxxxxxxx
```

Example Response Body (Administratoranmeldung)

```
{
  "access_token":"CnQUooiUDgNGl...U582cYQd6F-8K4S0",
  "token_type":"bearer",
  "expires_in":1199,
  "role":"Administrator",
  ".issued":"Thu, 15 Oct 2015 09:30:57 GMT",
  ".expires":"Thu, 15 Oct 2015 09:50:57 GMT"
}
```

Wie mit der Antwort des Servers jeweils umgegangen wird, ist unter *4.2.9 Aufbereiten der API Daten* beschrieben. Sollten bei der Abfrage der Daten Fehler auftreten (z.B. falsch eingegebenes Passwort oder falsche URL) so wird dem Client vom Server eine Fehlermeldung zurückgeliefert:

Example Response Body (HTTP-Statuscode 404 (Not Found))

```
{
  "error":"Ungültige Port_ID",
  "error_description":"Es wurde kein Port mit der ID 1 gefunden."
}
```

Diese Art von Fehlerbehandlung gilt für alle verwendeten API-Requests.

War der Login erfolgreich, so ist es in weiterer Folge möglich, reale Daten des SmartHomes zu beziehen. Für die Navigation zu den einzelnen Geräten wird dem Benutzer eine Übersicht der Räume angezeigt. Diese Liste kann man wiederum vom Server abfragen:

Example Request

```
GET URL/api/Rooms?sort=Description
```

Example Result

```
[
  {
    "Room_ID":1,
    "Description":"Labor Waser"
  },
  {
    "Room_ID":2,
    "Description":"EDV 1"
  }
  ...
]
```

Immer wenn Daten vom Server zum Anzeigen im Browser abgefragt werden, laufen diese Anfragen meist ähnlich ab. Zum besseren Verständnis folgt noch ein Beispiel, wie man den Server per API - Request dazu bringt, beispielsweise digitale I/O's der Module zu schalten:

Example Request

```
PUT URL/api/DigitalOutputs/1/Value
```

Request Header

```
Content-type: application/json
```

Example Request Body

```
{
  "Value":true
}
```

4.2.4 JSON Codierung

Da wie in 4.2.3 *Universal SmartHome API* erwähnt die gesamte Datenübertragung mit der API im JSON-Format von statten geht, ist es nun an der Zeit über den Aufbau dieses Textformates zu sprechen und wie diese Kommunikation implementiert ist.

JSON (JavaScript Object Notation) ist ein vereinheitlichtes Datenformat, das entwickelt wurde um JavaScript Objekte abzubilden und zu versenden. Bald schon wurde das Format in anderen Sprachen als dynamische Alternative zu eher strengeren Formaten wie *XML* verwendet. Es haben sich über die Zeit mehrere Parser in verschiedensten Sprachen entwickelt, um mit JSON zu arbeiten, wobei sich in C# *Newtonsoft.JSON* bewährt hat. Mit dieser Erweiterung beschränkt sich das Serialisieren und Deserialisieren von Daten im JSON Format meist auf eine Zeile.

```
1 | StringContent content = new StringContent (Newtonsoft.Json.JsonConvert.
    |     SerializeObject(room), Encoding.UTF8, "application/json");
```

Listing 4.2: JSON-Codieren

```
1 | return JArray.Parse(await client.GetStringAsync(Globals.BASE_URL + "api/Rooms?
    |     sort=Description")).ToObject<List<CRoom>>();
```

Listing 4.3: JSON-Decodieren

4.2.5 Databinding

Es gibt mehrere Ansätze um in einer Webanwendung Eingaben des Benutzers zu verarbeiten, wie altbewährte HTML-Formulare oder JavaScript-Funktionen, welche die Daten von Eingabefeldern auslesen. Blazor bietet hier die Möglichkeit von *Databinding*.

Jedes Inputelement bekommt durch den neuen Blazor Syntax ein *@Bind* - Attribut, mit dem der Inhalt von zum Beispiel einem Eingabefeld im Hintergrund auf einen String gebunden werden kann. Im Hintergrund ist das Ganze als Two-Way-Binding implementiert, was bedeutet, wenn man im Code-Behind den Inhalt des Strings bearbeitet, sieht man die Änderung auch im Eingabefeld. So ist die Kommunikation zwischen Benutzeroberfläche und Code sehr elegant und intuitiv gelöst.

```
1 | <input type="text" class="form-control" id="usr" placeholder="@Globals.
    |     MultiLanguage.Username" name="user" @bind="@User" />
```

Listing 4.4: @Bind Attribut

```
1 | public string User { get; set; }
```

Listing 4.5: @Bind Attribut Code Behind

4.2.6 Der Server

Im Hintergrund der Website agiert ein Server, welcher für die Datenverarbeitung, Kommunikation mit Steuermodulen und der Verwaltung der Daten zum Haus zuständig ist. Auf diesem Server befindet sich die dazugehörige Datenbank und die API, über welche mit dem Server kommuniziert wird (siehe 4.2.3 *Universal SmartHome API*). Der Server verarbeitet eingehende HTTP-Requests vom Client und bearbeitet je nach dem zum Beispiel Einträge in der Datenbank. Wird beispielsweise ein neues Gerät zu einem Raum hinzugefügt, wird diese Verknüpfung in die Datenbank eingetragen.

Auch das eigentliche Steuern der WLAN-Module übernimmt der Server nach dementsprechendem HTTP-Request. Wird also ein Gerät auf der Website gesteuert, sendet der Client diese Anfrage an den Server und dieser sendet dementsprechende Signale an das richtige Modul.

Mittels SignalR (siehe 2.3 *SignalR^{JB}* und 4.2.17 *Live Daten mit SignalR*) kann sich der Client auf vom Server zur Verfügung gestellte Events anmelden, wodurch dieser das Senden von Live-Daten übernehmen kann.

All diese Funktionalitäten zusammen mit der Datenbank (ausgenommen SignalR) sind von

früheren Versionen des Universal SmartHomes übernommen worden und sind somit nicht Teil der Arbeit gewesen.

4.2.7 Inheritance & Dependency Injection

Blazor bringt mit seinem neuen Syntax Programmierkonstrukte wie Ableitung und Dependency Injection^[12] auf seine Razor-Pages. Mit dem Schlüsselwort *@inherits* kann eine Razor Seite von einer C# - Klasse abgeleitet werden, falls diese selbst von der Klasse *ComponentBase* abgeleitet ist. So ist es möglich, die Daten einer Seite als Klasse zu behandeln, um Daten im objektorientierten Stil zu bearbeiten. Events von Userinterface-Elementen können ganz einfach auf Methoden der Klasse gesetzt werden, Eigenschaften der Klasse können ohne Umwege auf der Seite verwendet werden, wobei diese beim Aufrufen der Seite initialisiert und beim Verlassen bestmöglich aufgeräumt werden. Gibt es beim Verlassen noch weitere Schritte, die für richtiges Verhalten geschehen müssen, kann man durch diesen Aufbau bekannte Ressourcenfreigabe-Mechanismen von Objekte anwenden (Siehe 4.2.18 Freigeben von Ressourcen (*IDisposable*)).

Dependency Injection wird in Blazor unterstützt, wodurch es möglich ist, Dienste (Klassen) die an einer zentralen Stelle konfiguriert sind, auf verschiedensten Seiten zu verwenden. Paradebeispiel ist hier der *NavigationManager* welcher in Blazor verwendet wird, um zwischen verschiedenen Seiten zu navigieren. Dieser ist als Singleton an einer Stelle erzeugt worden, mit dem Schlüsselwort *@inject* bekommt man eine Referenz auf das Objekt. Will man in einem C# - File eine Referenz eines solchen Dienstes, so erstellt man einfach eine Variable des Dienstes und gibt dieser das *[Inject]* - Attribut. Dadurch erhält man auch wieder eine Referenz auf dasselbe Objekt, womit man dann arbeiten kann.

Will man selbst solch einen Dienst erstellen, kann man ganz einfach in der *ConfigureServices* - Methode der Klasse *Startup* von Blazor einen eigenen Dienst hinzufügen und diesen wie die anderen Default-Dienste injecten.

```
1 | @inherits Universal_SmartHome.Code.Room_Code
```

Listing 4.6: Inheritance

```
1 | @inject IToastService toastService
2 |
3 |
4 | [Inject]
5 | private IToastService toastService { get; set; }
```

Listing 4.7: Dependency Injection

```
1 | public void ConfigureServices(IServiceCollection services)
2 | {
3 |     services.AddRazorPages();
4 |     services.AddServerSideBlazor();
5 |     services.AddI18nText<Startup>();
6 |     services.AddBlazoredToast();
7 | }
```

Listing 4.8: Dienst Hinzufügen

4.2.8 Asynchrone Programmiermodelle

Sobald in einer Anwendung User-Interface existiert muss die gesamte Datenverarbeitung und vor allem der Datentransfer asynchron ablaufen, um die Oberfläche nicht "einfrieren" zu lassen.

Blazor bietet hierzu ein System von *Lifecycle* - Methoden (siehe 4.2.10 *Blazor Lifecycle Methoden*) an, um Datenverarbeitung zu bestimmten Zeitpunkten ganz leicht asynchron zu bewältigen.

Weiters sind viele Funktionalitäten wie HTTP-Requests mittels *HttpClient* in C# bereits asynchron vorhanden, was die Verwendung um einiges erleichtert. So können Daten einfach asynchron bezogen und verarbeitet werden, um sich am Schluss bei der Oberfläche wieder zu melden um die Daten anzuzeigen.

4.2.9 Aufbereiten der API Daten

Zum Speichern der angeforderten Daten waren einige Überlegungen notwendig in Bezug auf Datenstrukturen. Grundsätzlich gibt es zwei Möglichkeiten mit JSON-Daten nach dem Decodieren umzugehen. Man kann das dabei entstandene Objekt auf eine Variable vom Typ *dynamic* speichern, was keinen zusätzlichen Code benötigt oder das entstandene Objekt als C# - Klasse modellieren und auf dieses zuweisen. Bei der zweiten Methode muss man jedoch für jede Datenstruktur, die als Antwort des Requests zurückkommen kann, eine dementsprechende Klasse schreiben. Da für das Erstellen einer Benutzeroberfläche Intellisense (=Schreibervervollständigung) sehr hilfreich und fast notwendig ist, bietet sich die zweite Methode besser an. Für ein *dynamic* - Objekt ist kein Intellisense vorhanden, weil der tatsächliche Typ erst zur Laufzeit feststeht.

Handelt es sich nun bei dem Objekt um etwas Einzigartiges wie beispielsweise den aktuellen Raum, kann es schlicht auf eine Variable des passenden Typs gespeichert werden. Handelt es sich jedoch um eine Ansammlung von Daten, wie die Geräte in einem Raum, muss man sich überlegen, welche Datenstruktur am besten geeignet ist. Für alle Auflistungen (z.B. Raumübersicht) werden *Lists* verwendet, da die Daten von der API beispielsweise bereits alphabetisch aufsteigend geordnet zur Verfügung gestellt werden. Für Daten wie die I/O's eines Raums bietet sich ein *Dictionary* gut an, da die IDs der I/O's direkt als Keys verwendet werden können. Weiters werden I/O's und deren Werte eher direkt abgefragt, wo hingegen bei der Raumübersicht die Daten durchlaufen werden, weshalb sich diese Strukturen jeweils gut eignen.

```

1 public class CDigitalInput
2 {
3     public int? DigitalInput_ID { get; set; }
4     public int? PortNr { get; set; }
5     public int? Module_ID { get; set; }
6     public string Description { get; set; }
7     public int? Room_ID { get; set; }
8     public string Shortcut { get; set; }
9     public string ShortcutOn { get; set; }
10    public string ShortcutOff { get; set; }
11    public bool? Online { get; set; }
12
13    public CDigitalInput()
14    {
15
16    }
17
18    public CDigitalInput(dynamic DI)
19    {
20        this.DigitalInput_ID = DI.DigitalInput_ID != null ? (int)DI.
            DigitalInput_ID : 0;
21        this.PortNr = DI.PortNr != null ? (int)DI.PortNr : 0;
22        this.Module_ID = DI.Module_ID != null ? (int)DI.Module_ID : 0;
23        this.Description = DI.Description != null ? DI.Description : "";
24        this.Room_ID = DI.Room_ID != null ? (int)DI.Room_ID : 0;
25        this.Shortcut = DI.Shortcut != null ? DI.Shortcut : "";
26        this.ShortcutOn = DI.ShortcutOn != null ? DI.ShortcutOn : "";
27        this.ShortcutOff = DI.ShortcutOff != null ? DI.ShortcutOff : "";
28        this.Online = DI.Online != null ? DI.Online : false;
29    }
30 }

```

Listing 4.9: Klasse CDigitalInput

4.2.10 Blazor Lifecycle Methoden

Blazor bietet für das zeitliche Managen des Ablaufs der Anwendungslogik sogenannte *Life-cycle Methods*^[13] (Lebenszyklus Methoden) an. Mit diesen kann man gewisse Arbeitsschritte zu bestimmten Zeitpunkten des "Lebens" einer Anwendung ausführen. So ist es zum Beispiel möglich, Daten für eine Website vor dem ersten Rendern anzufordern und diese erst anzuzeigen, wenn sie vorhanden sind. Weiters kann man darauf reagieren, wenn eigene Parameter von außen gesetzt oder verändert werden, um diese beispielsweise anzuzeigen oder auszuwerten.

Zur Verwendung dieser Methoden überschreibt man in dem jeweiligen *Component* (z.B.: in einer Page) die entsprechende Methode und führt in dieser die gewünschten Arbeitsschritte aus. Hier bemerkt man, dass nicht wie bei anderen Plattformen diese Lebenszyklus Methoden für die gesamte Anwendung überschrieben werden, sondern für jede Komponente einzeln. So entstehen keine Kollisionen zwischen den Komponenten durch Verwendung von ein und derselben Methode. Wenn also beim Aufrufen verschiedener Seiten ganz unterschiedliche Schritte für richtiges Verhalten nötig sind, stellt das in Blazor kein Problem dar.

```
1 public List<CRoom> rooms = null;
2
3 protected override async Task OnInitializedAsync()
4 {
5     rooms = await Universal_SmartHome_Services.GetRooms();
6 }
```

Listing 4.10: OnInitializedAsync Lifecycle Methode

4.2.11 URL Parameter: /{var}

Um für ähnliche Seiten nicht immer ein neues Layout erstellen zu müssen, kann man Teile der Daten auch variabel halten. Hierfür kann man in Blazor das *[Parameter]* - Attribut verwenden. Variablen, welche mit diesem Attribut definiert werden, können als Parameter in der URL mit übergeben werden bzw. gesetzt werden. So kann für z.B. die Übersichtsseite der verschiedenen I/O's der Typ des I/O's als Parameter übergeben werden und die Daten somit dynamisch je nach Übergabe geladen und angezeigt werden, ohne an der Razor-Page etwas zu ändern.

```
1 @page "/io/{type}"
2
3
4
5 public class IO_Code : ComponentBase, IDisposable
6 {
7     [Parameter]
8     public string type { get; set; }
9
10    ...
11 }
```

Listing 4.11: URL-Parameter

4.2.12 Das Problem mit deutschem Visual Studio

In diesem Abschnitt geht es kurz darum, bei mehrsprachiger Entwicklung besonders vorsichtig zu sein. Die Darstellung von Daten wie *double* - Werten variiert von Region zu Region, so wird eine Kommazahl im englischsprachigen Raum mit einem '.' (Punkt) und im deutschsprachigen Raum mit einem , (Beistrich, Komma) getrennt. Da bei deutschen Entwicklungsumgebungen hier also ein Komma als Trennzeichen gilt, wird dieses von englischen Parsern falsch interpretiert.

Beim Festlegen des Wertebereich eines Schiebereglers tritt genau hier ein Problem auf: Der Parser ignoriert das Komma und verwendet die Nachkommastellen als Vorkommastellen. In diesem Fall kann man sich mit einfacher Textersetzung Abhilfe verschaffen, doch ähnliche Probleme treten meist ungeahnt auf. Daraus sollte man lernen, nicht nur im eigenen Code nach Fehlerquellen zu suchen, sondern auch die Umgebung, in der sich der Fehler ereignet, miteinzubeziehen.

4.2.13 SmartHome Regeln

Das Universal SmartHome besitzt eine Möglichkeit, Abhängigkeiten zwischen Modulen zu modellieren und sogar Zeitsteuerung zu ermöglichen. Durch *Regeln* kann man verschiedene I/O's miteinander koppeln, um beispielsweise die Heizung einzuschalten wenn, eine Messeinheit eine zu niedrige Temperatur misst. Weiters kann man einstellen, dass Geräte zu bestimmten Uhrzeiten ein-, oder ausgeschaltet werden. Hier kann man auch Schlüsselwörter wie *sunrise* oder *sunset* verwenden, um nach dem variablen Verlauf der Sonne zum Beispiel Geräte nur eingeschaltet zu lassen, solange Sonne vorhanden ist, um Photovoltaikanlagen zu betreiben.

Wie diese Regeln erstellt werden können wird in *3.2.11 SmartHome Regeln einrichten* beschrieben. Eingetragene Regeln sind dann am Server hinterlegt, welcher diese auswertet (*parst*) und verarbeitet, um die gewünschte Funktionalität zur Verfügung zu stellen.

```
1 | TIMER "Schlafzimmer#Schlafzimmer-Licht" = 10:30#Mo#Di#Mi#Do#Fr#Sa#So 10:45#Mo#Di
   | #Mi#Do#Fr#Sa#So
```

Listing 4.12: SmartHome Regeln

4.2.14 Livestreams & Augmented Reality

Die Erweiterung des bestehenden Universal SmartHomes um einen funktionierenden Video-Livestream und diesen mit Augmented Reality auszustatten, war einer der größten Punkte des Projekts. Die Benutzeroberfläche des Livestreams und allgemeines über Augmented Reality wird in *3.2.8 Kamera/Augmented Reality* beschrieben.

Für die Implementierung des Livestreams war anfangs wie bei vorherigen Projekten geplant, sich mit der Anwendung auf einen Websocket zu verbinden, um den rohen Bytestrom der Kamera zu empfangen. Websockets sind eine Möglichkeit zwischen Server und Client eine TCP Verbindung aufzubauen, um möglichst latenzfrei Daten zu übertragen. Mit vorhandenen JavaScript - Libraries wurde hier clientseitig decodiert und als *MJPEG (Motion - JPEG)* direkt auf einem *Canvas* - Element angezeigt. Hierbei entstanden einige Probleme, da das Decodieren des Bytestroms teilweise für den Client sehr aufwendig war und dadurch der Livestream sehr verzögert, oder teilweise auch gar nicht angezeigt werden konnte. Die Lösung zu diesem Problem ist, die Daten bereits am Server zu decodieren und nur noch die fertigen Bilder im JPEG - Format an den Client zu senden. Dadurch wird dem Client die Aufgabe abgenommen, alle eingehenden Daten zu verarbeiten, wodurch sich mögliche Verzögerungen bei der Verarbeitung nicht wie zuvor aufsummieren können. So entsteht, auch wenn der Livestream geöffnet bleibt, maximal eine Verzögerung um die Latenzzeit der Übertragung.

Die Verbindung zum Server wird im neuen System über einen *SignalR - Hub* realisiert, womit es möglich ist, dem Server Methoden der Clients zur Verfügung zu stellen, welche er selbst aufrufen kann, um zum Beispiel das Anzeigen neuer Daten anzuregen (mehr dazu in *4.2.17 Live Daten mit SignalR*).

Zum Anzeigen wird wegen der noch mangelnden Alternativen der Umweg über JavaScript verwendet, um die jeweiligen JPEG - Frames auf einem *Image - Element* anzuzeigen (siehe *4.2.15 JavaScript - C# Brücke*). Hier wird das Bild einfach als *src* (Source) des Elements gesetzt und immer wenn ein neuer Frame vorhanden ist, wird sie aktualisiert.

```
1 public async void StartVideo()
2 {
3     var stream = this.hubConnection.StreamAsync<byte[]>("StartVideoStream",
4         selected_camera.IP, this.videoCancellationTokens.Token);
5
6     try
7     {
8         await foreach (var frame in stream)
9         {
10             ...
11             await _jsRuntime.InvokeAsync<string>("showFrame", buffer);
12             ...
13         }
14     }
15     catch (Exception ex)
16     {
17         ...
18     }
19 }
20 }
```

Listing 4.13: Video - Stream starten

```
1 window.showFrame = (frame) => {
2     var image = new Uint8Array(frame);
3     if (document.getElementById('img_videoStream') != null) {
4         document.getElementById('img_videoStream').src = URL.createObjectURL(new
5             Blob([image], { type: "image/jpeg" }));
6     }
7 };
```

Listing 4.14: Frame anzeigen

Um das graphische Steuern von Geräten zu ermöglichen, benötigt man neben dem Livestream auch noch eine Benutzeroberfläche. Erneut wird hier, wegen des geringen bisherigen Supports von Blazor, JavaScript verwendet, um einfache 2D - Graphiken mittels eines *Canvas* zu zeichnen. Hier arbeitet im Hintergrund eine Event-basierende Logik für die Maus, um über im Video vorhandene Geräte als Admin ganz einfach Rechtecke zu ziehen und diese mit dem Gerät zu koppeln. Sind die Rechtecke fertig eingerichtet, wird bei weiteren Mausklicks erkannt, ob man sich innerhalb eines Rechtecks befinden und dementsprechend ein Gerät aus-, bzw. eingeschaltet.

```

1 //Mousedown
2 document.getElementById('augmentedRealityCanvas').addEventListener('mousedown',
    function (e) {
3     if (rectangleDrag == false) {
4         return;
5     }
6
7     rect.left = parseInt(e.clientX - canvasx);
8     rect.top = parseInt(e.clientY - canvasy);
9     mousedown = true;
10 });
11
12 //Mouseup
13 document.getElementById('augmentedRealityCanvas').addEventListener('mouseup',
    function (e) {
14
15     ...
16
17     dotNetRuntime.invokeMethodAsync('CreateNewAugmentedIO', rect);
18 });
19
20 //Mousemove
21 document.getElementById('augmentedRealityCanvas').addEventListener('mousemove',
    function (e) {
22     if (rectangleDrag == false) {
23         let position = { x: parseInt(e.clientX - canvasx), y: parseInt(e.clientY
            - canvasy) };
24
25         dotNetRuntime.invokeMethodAsync('OnCanvasHover', position);
26
27         return;
28     }
29
30     if (mousedown) {
31
32         augmentedctx.clearRect(0, 0, augmentedCanvas.width,
            augmentedCanvas.height); //clear canvas
33         augmentedctx.beginPath();
34
35         ...
36
37         augmentedctx.stroke();
38     }
39 });
40
41 //Mouseclick
42 document.getElementById('augmentedRealityCanvas').addEventListener('click',
    function (e) {
43     if (rectangleDrag == false) {
44         let position = { x: parseInt(e.clientX - canvasx), y: parseInt(e.clientY
            - canvasy) };
45
46         dotNetRuntime.invokeMethodAsync('OnCanvasClick', position);
47
48         return;
49     }
50 });

```

Listing 4.15: Mouse Events

```

1 public async void OnCanvasClick(JsonElement mouse)
2 {
3     int mousex = mouse.GetProperty("x").GetInt32();
4     int mousey = mouse.GetProperty("y").GetInt32();
5
6     foreach(CAugmentedDigitalOutput DO in AugmentedDigitalOutputs)
7     {
8         if(AugmentedIOContainsPoint(mousex, mousey, DO))
9         {
10             if (DO.Button)
11             {
12                 Universal_SmartHome_Services.PulseDigitalOutput(DO);
13             }
14             else
15             {
16                 if (io_values["DigitalOutput"][DO.DigitalOutput_ID.ToString()].
17                     Equals("True"))
18                 {
19                     await SetAugmentedDigitalOutput(DO, "DigitalOutput", "false"
20                     );
21                 }
22                 else
23                 {
24                     await SetAugmentedDigitalOutput(DO, "DigitalOutput", "true"
25                     );
26                 }
27             }
28         }
29     }
30 }

```

Listing 4.16: Augmented Reality steuern

Bei der Implementierung der Funktionalitäten für den Augmented Reality - Livestream wurde großer Wert auf Einfachheit gelegt, da der ganze Rahmen der Bedienung sonst sehr schnell sehr technisch werden kann. Zum Verknüpfen eines visuellen Rechtecks mit einem Gerät wird beispielsweise die ID dieses Gerätes benötigt. Diese ist als Benutzer jedoch eher umständlich herauszufinden, weshalb hier intern von der ID auf den Textbezeichner (Namen) des Gerätes für die Auswahl geschlossen wird. Letztendlich wird dieser wieder zurück auf eine ID geführt für den korrekten API - Call. Das alles bedeutet zwar zusätzlichen, nicht unbedingt nötigen Aufwand, erleichtert jedoch die Handhabung als Benutzer ungemein.

Auch werden beim *hovern* über Rechtecke Informationen über das zu Grunde liegende Gerät angezeigt, um für diese nicht extra das Gerät in einer anderen Liste suchen zu müssen. Ganz intuitiv wird hier der derzeitige Zustand nicht nur in diesen Informationen angezeigt, sondern ist auch als Farbe des Rechtecks ersichtlich. Ist beispielsweise eine Lampe eingeschaltet, erscheint das Rechteck grün, ist sie aus, wird es rot angezeigt. All das in Kombination ergibt eine für den Anwender einfache und angenehme Benutzererfahrung.

4.2.15 JavaScript - C# Brücke

Microsoft ist sich mit der Entwicklung von Blazor ganz klar bewusst, dass ein Neuankömmling, der das System revolutionieren möchte, in der Webentwicklung kaum eine Chance hat. Deshalb kommt zusammen mit Blazor auch die Möglichkeit, auf bereits verwendete JavaScript - Libraries zurückzugreifen. In der Anwendung kann man eine als Singleton - Service realisierte JavaScript - Runtime durch *Dependency Injection* sowohl in Razor - Pages als

auch in Code - Behind - Files verwenden (siehe 4.2.7 *Inheritance & Dependency Injection*). JavaScript - Funktionen können mit diesem Runtimeobjekt ganz einfach asynchron aufgerufen werden, wobei man auf die Antwort der Funktionen wie gewohnt mit dem *await* - Schlüsselwort warten kann. Beim Universal SmartHome wird bei noch nicht unterstützten Funktionen, wie dem Zeichnen auf einem *Canvas* oder dem Abfragen von Standortinformationen im Browser, auf diese Weise über JavaScript gearbeitet.

Um auch Abläufe wie C# - Methoden als Callback für JavaScript - Libraries zu realisieren, ist in Blazor auch der umgekehrte Weg, nämlich C# - Methoden in JavaScript aufzurufen, möglich. Hier kann die Blazor - Anwendung eine *dotNET* - *Runtime* erzeugen, welche man auf eine JavaScript Variable speichern kann. Mithilfe dieser kann man wie zuvor Methoden asynchron aufrufen und auch wieder je nach Bedarf darauf warten. Methoden die für einen Aufruf aus JavaScript vorgesehen sind, müssen mit dem [JSInvokable] - Attribut versehen sein.

```
1  
2 await _jsRuntime.InvokeVoidAsync("updateAugmentedIOs", new object[] {  
    AugmentedDigitalInputs, AugmentedDigitalOutputs, AugmentedAnalogInputs,  
    AugmentedAnalogOutputs, io_values});
```

Listing 4.17: JavaScript Interoperationen

4.2.16 Dynamisches Programmieren: *ExpandoObject*

Entstehend aus der Aufteilung eines WLAN-Moduls in jeweils digitale I/O's und analoge I/O's entstehen 4 verschiedene Abbilder von Daten, die relativ ähnlich angezeigt werden können. Vor allem beim Einrichten einer Seite als Admin sind oftmals dieselben Eigenschaften, beispielsweise ID eines Geräts, zu setzen, diese haben aber verschiedene Namen. Um hier für die ähnlichen Datenstrukturen nicht 4 verschiedene Klassen erstellen zu müssen, wird eine höchst dynamische Datenstruktur, das *ExpandoObject*, verwendet.

Ein *ExpandoObject* ist vergleichbar mit, in Scriptsprachen gängigen, *var* - Objekten. Wird in diesen Sprachen auf Membervariablen, die beim Erstellen des Objekts noch nicht existieren etwas zugewiesen, besitzt das Objekt nach dieser Zuweisung die Variable und all das läuft ohne Probleme ab. Ähnlich verhält sich das *ExpandoObject* in einer solchen Situation, was man ausnutzen kann, um Eigenschaften zur Laufzeit zum Objekt hinzuzufügen und somit das oben genannte Problem zu umgehen. Wichtig zu erwähnen ist, dass das Objekt grundsätzlich als *dynamic* deklariert wird, wodurch leider keine Schreibervervollständigung für dieses Objekt vorhanden sein kann.

```

1 public static dynamic selected_io { get; set; }
2
3
4
5 <button type="button" class="btn btn-warning" @onclick='() => {
6     Globals.selected_io = new System.Dynamic.ExpandoObject();
7     Globals.selected_io.io = new CDigitalInput(d_input);
8     Globals.selected_io.type = "DigitalInput";
9     }' data-toggle="modal" data-target="#editIODialog">
10 </button>
11
12
13
14 <button type="button" class="btn btn-warning" data-dismiss="modal"
15     @onclick="async () => { await Universal_SmartHome_Services.EditIO(
16         Globals.selected_io.io, Globals.selected_io.type); UpdateIOs(); }">

```

Listing 4.18: ExpandoObject

4.2.17 Live Daten mit SignalR

Bei ersten Tests der Universal SmartHome Applikation trat das Problem auf, dass wenn mehrere Browser die exakt selbe Seite öffnen und jemand beispielsweise das Licht einschaltet, alle anderen Clients nichts davon mitbekommen und man somit nie die Gewissheit hat, ob die angezeigten Daten gerade aktuell sind. Dies mag zu Beginn nicht besonders problematisch klingen - wenn man etwas einschaltet, das bereits eingeschaltet ist, hat man am Ende dasselbe Ergebnis - jedoch können bei Geräten wie bidirektionalen Motoren teilweise gefährliche ungewollte Schaltungszustände entstehen.

Dieses Problem kann mit der Verwendung von *SignalR - Hubs* behoben werden. SignalR bietet die Möglichkeit serverseitig Event-ähnliche Hubs zu erstellen, auf welche sich jeder Client verbinden kann. Ein Client (Browseranwendung) meldet sich hier, beim Universal SmartHome sogar mit Authentifizierung, mit einer eigenen Callback - Methode an. Der Token für die Anmeldung kann wie in 4.2.3 *Universal SmartHome API* beschrieben, durch API-Requests angefordert werden.

```

1
2 public async static Task ConnectToSignalRIOHub()
3 {
4     if (IOHubConnection == null)
5     {
6         IOHubConnection = new HubConnectionBuilder()
7             .WithUrl(Globals.SignalR_IO_baseURL, options =>
8             {
9                 options.AccessTokenProvider = () => Task.FromResult(Globals.
10                     SignalR_IO_AccessToken);
11             })
12             .WithAutomaticReconnect()
13             .Build();
14         await IOHubConnection.StartAsync();
15     }
16 }

```

Listing 4.19: SignalR-Hub Anmeldung

Will man nun dem Server Methoden zum Aufrufen zur Verfügung stellen, registriert man diese durch das `HubConnection` Objekt. Hier gibt man als Schlüssel das jeweilige Event an, auf das man sich registrieren will und dazu eine Callback-Funktion mit den passenden Parametern (in diesem Fall *int* und *string*).

```
1
2 public static void SetSignalREventHandler(Action<int, string> Callback)
3 {
4     if(IOHubConnection == null)
5     {
6         //No Connection established
7         return;
8     }
9
10    try
11    {
12        IOHubConnection.Remove("ModuleIOEvent");
13
14        IOHubConnection.On<int, string>("ModuleIOEvent", (moduleID, message) =>
15        {
16            //Callback on ModuleChangedEvent
17
18            Callback(moduleID, message);
19        });
20    }
21    catch (Exception ex)
22    {
23        Console.WriteLine(ex.Message);
24    }
25 }
```

Listing 4.20: SignalR-Hub Callback Registrierung

Diese Methode kann ab diesem Zeitpunkt vom Server selbst aufgerufen werden, um den Client beispielsweise über Änderungen in der Datenbank zu informieren. Der Server ist in diesem Fall so mit den WLAN-Modulen verbunden, dass diese Rückmeldung geben, falls sich etwas an ihnen geändert hat (d.h. etwas geschaltet wurde). Der Server ruft daraufhin die von allen verbundenen Clients bereitgestellte Callback - Methode auf, wodurch diese auf die Änderung reagieren, Daten aktualisieren und alles wieder richtig auf der Oberfläche anzeigen können. So ist gewährleistet, dass zu jedem Zeitpunkt die Clientanwendungen auf dem aktuellsten Stand sind und Fehler wie eingangs erwähnt vermieden werden können.

Um wertvolle Ressourcen und Rechenleistung einzusparen, wird nicht bei jeder Änderung der Zustand aller Geräte abgefragt, sondern es ist in den Übergabeparametern der Callback - Methoden eine Codierung vorhanden, wodurch jeder Client herausfinden kann, ob das betroffene Geräte tatsächlich gerade auf der Oberfläche angezeigt wird oder nicht. Dies ist möglich, da die Anmeldung am SignalR - Hub jeweils erst beim Aufruf einer Seite passiert und hier jeweils eine passende Callback - Methode zur Verfügung gestellt wird.

Die Information, welches Gerät gerade geschaltet wurde, ist im Parameter *message*. Hierbei handelt es sich um einen string mit 7 Zeichen:

1. Zeichen - Lesen/Schreiben/Pulsen

L...Lesen
 S...Schreiben (bzw. High-Puls mit fixer Zeit)
 P...High-Puls mit variabler Zeit
 E...Event vom Modul ausgelöst

2. Zeichen - Porttyp

0...Digitaler Ausgang
 1...Digitaler Eingang
 2...Analoger Ausgang
 3...Analoger Eingang

3. Zeichen - Portnummer

0...9

4.-7. Zeichen - Wert

0000...Digital 0
 1111...Digital 1
 1100...High-Puls mit fixer Zeit
 XXXX...Analoger Wert (ModuleRange!)
 bzw. Zeit (in ms) für High-Puls mit variabler Zeit

Um diese Codierung für die spätere Verwendung verständlicher aufzubereiten, wird mit dieser Zeichenkette ein eigenes C# - Objekt (CModuleIOEventMessage) erstellt, welches die Informationen als verständliche Eigenschaften enthält:

```

1
2 public class CModuleIOEventMessage
3 {
4     public enum Operationtype { Read, Write, High_Puls, Event}
5     public enum IOtype { Digital_Output, Digital_Input, Analog_Output,
6         Analog_Input }
7     public enum Valuetype { Zero, One, High_Puls, Analog_Value}
8
9
10    public Operationtype Operation { get; set; }
11    public IOtype IO { get; set; }
12    public int PortNumber { get; set; }
13    public Valuetype TypeOfValue { get; set; }
14    public int Value { get; set; }
15    public bool IsInErrorCase { get; set; } = false;
16
17
18    public CModuleIOEventMessage(string message)
19    {
20        string s_operation = message.Substring(0, 1);
21        string s_IO = message.Substring(1, 1);
22        string s_PortNumber = message.Substring(2, 1);
23        string s_TypeOfValue = message.Substring(3, 4);
24
25        //Operation
26        switch (s_operation.ToUpper())
27        {

```

```

28         case "L":
29             this.Operation = Operationtype.Read;
30             break;
31
32         case "S":
33             this.Operation = Operationtype.Write;
34             break;
35
36         case "P":
37             this.Operation = Operationtype.High_Puls;
38             break;
39
40         case "E":
41             this.Operation = Operationtype.Event;
42             break;
43
44         default:
45             this.IsInErrorCase = true;
46             break;
47     }
48
49     //IOtype
50     this.IO = (IOtype)int.Parse(s_IO);
51
52     //PortNumber
53     this.PortNumber = int.Parse(s_PortNumber);
54
55     //Value
56     if ((this.IO == IOtype.Analog_Input) || (this.IO == IOtype.Analog_Output))
57         //Analog Value
58     {
59         this.TypeOfValue = Valuetype.Analog_Value;
60         this.Value = int.Parse(s_TypeOfValue);
61     }
62     else //Digital Value
63     {
64         switch (s_TypeOfValue.ToUpper())
65         {
66             case "0000":
67                 this.TypeOfValue = Valuetype.Zero;
68                 this.Value = 0;
69                 break;
70
71             case "1111":
72                 this.TypeOfValue = Valuetype.One;
73                 this.Value = 1;
74                 break;
75
76             case "1100":
77                 default:
78                     this.TypeOfValue = Valuetype.High_Puls;
79                     this.Value = int.Parse(s_TypeOfValue);
80                     break;
81         }
82     }
83 }

```

Listing 4.21: Modul Codierung

```

1 public void OnDatabaseChanged(int moduleID, string message)
2 {
3     CModuleIOEventMessage eventMessage = new CModuleIOEventMessage(message);
4     if (eventMessage.IsInErrorCase)
5     {
6         //SignalR Error
7         return;
8     }
9
10    switch (type)
11    {
12        case "DigitalOutput":
13            if(eventMessage.IO == CModuleIOEventMessage.IOtype.Digital_Output)
14                //check if IOs on Page have same type as eventMessage
15            {
16                foreach (dynamic IO in ios) //find IO of eventMessage
17                {
18                    if ((IO.Module_ID == moduleID) && (IO.PortNr == eventMessage
19                        .PortNumber))
20                    {
21                        switch (eventMessage.TypeOfValue)
22                        {
23                            case CModuleIOEventMessage.Valuetype.Zero:
24                                io_values["DigitalOutput"][(int) IO.
25                                    DigitalOutput_ID] = "False";
26                                base.StateHasChanged();
27                                break;
28                                ...
29                        }
30                    }
31                }
32            }
33            break;
34
35        case "DigitalInput":
36            ...
37            break;
38
39        case "AnalogOutput":
40            ...
41            break;
42
43        case "AnalogInput":
44            ...
45            break;
46    }
47 }

```

Listing 4.22: SignalR Callback Methode

4.2.18 Freigeben von Ressourcen (IDisposable)

Vor allem im Webbereich ist mit Ressourcen sehr sorgfältig umzugehen, da ein Browser schlicht nicht die gleiche Speicherkapazität und Rechenleistung aufbringen kann wie eine native Desktopanwendung. Umso wichtiger ist es, nicht mehr verwendete Ressourcen freizugeben. Praktischerweise ist bei einer Website oft ganz leicht zu erkennen, wann etwas nicht

mehr gebraucht wird: Wird es nicht auf der Webpage angezeigt, wird es gerade nicht benötigt. Genau für diese Denkweise gibt es in C# das `IDisposable` Interface. Eine Komponente, die beim Verlassen der Website Ressourcen freigeben soll, implementiert *IDisposable* und die dazugehörige Methode *Dispose()*.

Beim Universal SmartHome kommt dieses Konstrukt beim Verlassen der Kamera - Livestreamseite zum Einsatz. Die SignalR - Verbindung zum Empfangen der anzuzeigenden Bilder läuft in einer Endlosschleife ab, weshalb die Methode beim Verlassen nach kurzer Zeit eine *Exception* wirft, was zwar in Ordnung wäre, aber besser mit dem `IDisposable` Interface gelöst werden sollte. Am wichtigsten ist jedoch den Audiostream, welcher über eine JavaScript - Verbindung läuft zu stoppen, da dieser sonst weiterhin Audio abspielt, obwohl man sich nicht mehr im Livestream befindet. Hier wird in der *Dispose* - Methode die JavaScript Funktion zum Stoppen des Audiostreams aufgerufen, wodurch alle nötigen Schritte abgeschlossen sind, um die Livestream - Seite sauber zu verlassen.

```
1 public class Camera_Code : ComponentBase, IDisposable
2 {
3
4     ...
5
6     public void Dispose()
7     {
8         _jsRuntime.InvokeVoidAsync("disposeVideoHub", null);
9     }
10 }
```

Listing 4.23: IDisposable

4.2.19 Internet Information Services (IIS)

Für Webentwicklung bietet Visual Studio ganz einfache Möglichkeiten seine Website zu testen. Dort, wo man sonst seine Desktopanwendung starten kann, wird bei Webanwendungen ein mit IIS^[14] (Internet Information Service) gehosteter Browser gestartet. So kann ganz einfach das Aussehen und die Funktionalität einer Webseite in Visual Studio getestet werden. Will man nun für Präsentationen oder ähnliches die Applikation auf verschiedenen Endgeräten vorzeigen, muss die Webseite unter einer definierten IP - Adresse aufrufbar sein. Hier kann man sich wiederum mit IIS Abhilfe verschaffen. Windows bietet diesen Dienst an, um Webseiten auf einem Gerät mit Windows - Betriebssystem zu hosten, wobei die Webseite dann unter der IP - Adresse des Geräts erreichbar ist. Für das Einrichten einer Website mit IIS folgt man am besten der Dokumentation der offiziellen Microsoft Seite.

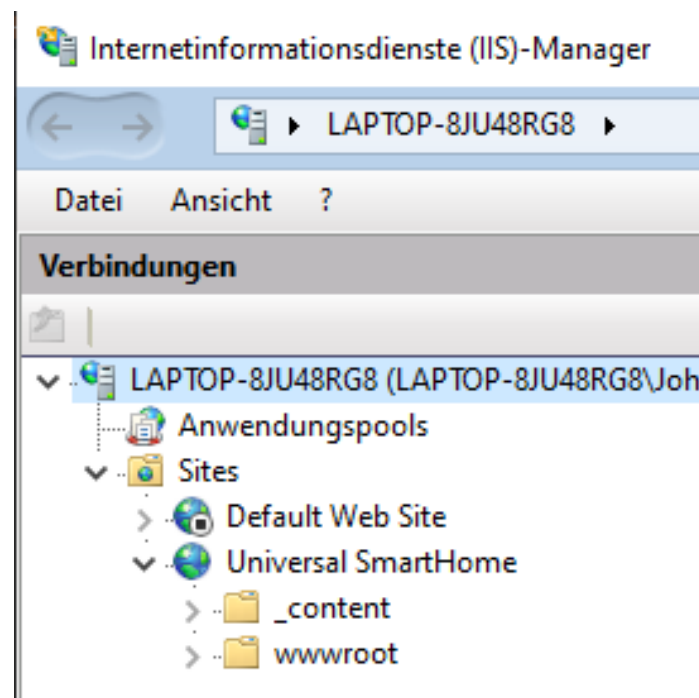


Abbildung 4.2: IIS - Manager

4.2.20 Wetter-, Sprit-, und GeoCoding-API

Da die meisten Funktionalitäten der eigentlichen Universal SmartHomes auf eigene Komponenten ausgelagert sind und die Navigation durch diese über eine Navigationsleiste am Rand erfolgt, bleibt für die Startseite der Anwendung viel Spielraum für freie Gestaltung. Die Überlegung war hier zu Beginn, mit welchen Informationen man auf den ersten Blick versorgt werden möchte. Nach kurzer Recherche war klar, dass hier Wetterinformationen bzw. Wettervorhersagen^[9] und Informationen über aktuelle Spritpreise^[10] sinnvoll sind. Um genauere Ergebnisse für die Standorterkennung zu erzielen, wurde in diesem Zusammenhang auch die Google GeoCoding - API^[11] verwendet. Allgemeine Infos zu den verwendeten APIs und zur Benutzeroberfläche der Startseite gibt es unter *3.2.5 Startseite*.

Die Vorgehensweise ist hier im Prinzip ähnlich wie in *4.2.9 Aufbereiten der API Daten* beschrieben. Für jede API wird ein neues File angelegt, welches Basis - URL und Datenstrukturen für die API - Antworten beinhaltet. Zu den Universal SmartHome Service - Methoden wird für jeden API - Call eine neue Methode hinzugefügt und die zurückkommende JSON-Antwort decodiert. Um es für die Entwicklung der Benutzeroberfläche so einfach wie möglich zu halten, werden für die jeweiligen Antworten C# - Klassen erstellt, um mit Schreibvollständigkeit arbeiten zu können. Die Daten für die Wettervorhersage werden im Code im Nachhinein noch in mehrere Listen umstrukturiert, um den Aufbau der Razor - Seite durch wenige Schleifen möglichst einfach zu halten.

Sowohl die Wetter API als auch die Spritpreis API benötigen für die Abfrage Standortinformationen. Erteilt der Benutzer die Berechtigung den Standort der Gerätes abzufragen wird der aktuelle Standort verwendet, ansonsten der Entwicklungsstandort des Universal SmartHomes Braunau am Inn, wobei man hier die Adresse auf der Startseite selbst eingeben kann. In jedem Fall wird für die API-Abfragen nicht die Adresse sondern die geographischen Koor-

dinaten verwendet. Um die Adresse in Koordinaten umzuwandeln, wird die *Google GeoCoding* - API verwendet.

```

1 public async static Task<CCurrentWeather> GetCurrentWeatherByAddress(string
   address)
2 {
3     try
4     {
5         GoogleGeocoding.Location location = await GoogleGeocoding.Geocode(
           address);
6
7         // Geocoding failed
8         if(location == null)
9         {
10             return new CCurrentWeather();
11         }
12
13         string lat = location.Latitude.ToString().Replace(',', ' ');
14         string lon = location.Longitude.ToString().Replace(',', ' ');
15
16         client.DefaultRequestHeaders.Clear();
17
18         return JObject.Parse(await client.GetStringAsync(OpenWeatherMap.
           OWM_BASE_URL +
19             "weather?APPID=" + OpenWeatherMap.OWM_API_KEY +
20             "&units=metric" +
21             "&lang=" + Globals.MultiLanguage.Culture_Info +
22             "&lat=" + lat + "&lon=" + lon)).ToObject<
           CCurrentWeather>());
23     }
24     catch (Exception ex)
25     {
26         Console.WriteLine(ex);
27         new ErrorHandler().HandleException(ex);
28     }
29     return new CCurrentWeather();
30 }

```

Listing 4.24: Wetter API Call

4.3 Fehlerbehandlung im Code

Sobald eine Anwendung von äußeren Einflüssen abhängig ist, muss man damit rechnen, dass durch diese Einflüsse Probleme entstehen. Da die Anwendung aber bei solchen Problemen nicht auseinanderbrechen soll, muss man sich auf diese vorbereiten. Wie dem Benutzer auftretende Fehler mitgeteilt werden ist unter *3.3 Fehlerbehandlung graphisch* beschrieben. Die größte Fehlerquelle bei einer Webanwendung ist logischerweise die Internetverbindung. Deshalb wird jeder fehlgeschlagene API - Request mit einem *try - catch* - Konstrukt abgefangen und jeweils default - Werte als Rückgabe zurückgeliefert. Dadurch werden bei Listen und Dictionaries Datenstrukturen mit Länge 0 zurückgegeben und dadurch von den Schleifen bei der Erstellung der Benutzeroberfläche übersprungen. Handelt es sich um nur ein Objekt wird entweder stattdessen *null* zurückgeliefert, worauf in der Benutzeroberfläche abgefragt werden kann oder wieder der Standardwert des Objektes zurückgegeben, was meistens oberflächlich keine Auswirkungen hat. Weiters wird bei jedem auftretendem Fehler bereits im Services-

File Errorhandling betrieben. So wird beispielsweise bei erkanntem *Unauthorized* - Error der Benutzer auf die Loginseite geleitet, um einen neuen *Authentication Token* zu erhalten. Grundsätzlich sind all diese Maßnahmen nur Konventionssache, da eine solche Anwendung ohne funktionierendes Internet auch keine Anwendung findet, nichtsdestotrotz sollte man eine Anwendung deswegen bei Problemen nicht "gegen die Wand laufen lassen".

```
1 public async static Task<List<CRoom>> GetRooms ()
2 {
3     try
4     {
5         ...
6     }
7     catch (Exception ex)
8     {
9         Console.WriteLine(ex);
10        new ErrorHandler().HandleException(ex);
11    }
12    return new List<CRoom> ();
13 }
14
15
16
17 private class ErrorHandler
18 {
19     public void HandleException(Exception ex)
20     {
21         switch (ex.HResult)
22         {
23             // Not Authorized Exception
24             case -2146233088:
25                 Console.WriteLine("Unauthorized Exception");
26                 Console.WriteLine(ex.HResult);
27                 UriHelper.NavigateTo("/");
28                 break;
29
30             default:
31                 Console.WriteLine(ex.HResult);
32                 break;
33         }
34     }
35 }
```

Listing 4.25: Errorhandling im Code

5 Test und Inbetriebnahme^{JB}

Dieser Abschnitt beschäftigt sich mit dem Testen und der Inbetriebnahme aller Komponenten des Universal SmartHomes.

5.1 Der Server

Für die Kommunikation zwischen Webanwendung und den Hardwaremodulen ist der Universal SmartHome Server zuständig. Mittels REST-API ermöglicht er dem Client die Konfigurationen des eingerichteten SmartHomes zu verändern und Befehle zum Steuern der Hardwaremodule zu senden. Was die Inbetriebnahme betrifft, musste wenig bis keine Arbeit verrichtet werden, da der Server bereits von vorherigen Projekten fertig eingerichtet in Betrieb genommen wurde. Nach den ersten erfolgreichen API - Anfragen war klar, dass hier alles fehlerfrei funktioniert.

Manche Funktionalitäten, wie das zeitliche Schalten von Modulen, wurde erst bei dieser Version der Serveranwendung hinzugefügt, da sie bei vorherigen Projekten noch nicht zum Einsatz kamen. Auch die Schnittstelle für den SignalR - Hub wurde neu hinzugefügt (siehe *4.2.17 Live Daten mit SignalR*). Diese Neuheiten wurden zu Beginn jeweils in separaten Projekten getestet und danach in die Webanwendung mit eingebunden. Allgemeines zur Verwendung des Servers findet man unter *4.2.3 Universal SmartHome API*.

5.2 Die Module

Die Hardwarekomponenten zum Steuern eines Hauses sind die eigens entwickelten WLAN - Schaltrelais. Diese Module werden mit Netzspannung angeschlossen und nach dem Hochfahren und Bespielen mit der Software des Herstellers des WLAN - Sockels melden sich diese Module bei der eingerichteten Schnittstelle, sobald sie online sind. Diese können danach mit bestimmten Befehlen gesteuert werden. Das Umsetzen von HTTP - Requests auf Befehle, die die Module verstehen, übernimmt der Universal SmartHome Server. Da die Erstellung der serverseitigen Anwendung nicht Teil der Aufgabenstellung des Projektes war, kann hier leider nicht näher auf die genauen Abläufe der Serverseite und die Kommunikation mit den Modulen eingegangen werden.

5.2.1 Pinbelegung der Module

Für den fertigen Testaufbau wurden die fünf zur Verfügung stehenden Module verwendet, wobei zu sagen ist, dass jedes Modul nur begrenzte I/O's besitzt. So sind jeweils auf jedem Modul ein digitaler Eingang zum Messen von digitalen Signalen (0V / 5V), ein analoger Eingang zum Messen von analogen Signalen (Temperaturmessung), ein analoger Ausgang zum Steuern von analogen Geräten und zwei Schaltrelais zum Durchschalten bzw. Ausschalten von angeschlossenen Leitungen (Licht ein-, ausschalten). Das Verbinden der einzelnen Pins

ist recht simpel, man muss hier nur den richtigen Pin am Modul finden (siehe *A Diverse Anhänge*), einen der digitalen Ausgänge zu verwenden, bedarf ein paar mehr Überlegungen.

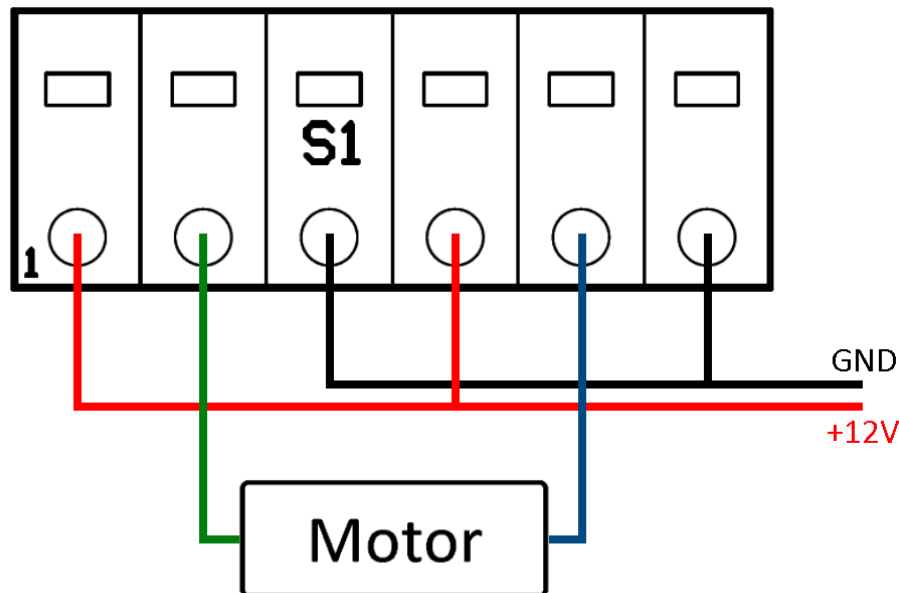


Abbildung 5.1: Anschluss Motor

In Abbildung 5.1 *Anschluss Motor* ist stellvertretend der Anschluss eines bidirektionalen Motors, wie er für die Jalousien verwendet wird, gezeigt. Hier werden beide digitalen Ausgänge eines Moduls verwendet, um den Motor in beide Richtungen zu steuern. Hier ist auch zu erkennen, dass jeweils die ersten drei und die letzten drei Anschlüsse zusammen ein Schaltrelais bilden, wobei der mittlere Anschluss entweder nach rechts oder nach links durchgeschaltet werden kann. Um beispielsweise einen Lichtschalter mit diesen Modulen zu realisieren, kann die Zuleitung zum Licht durch eine der Relaisgruppen geführt werden, wobei hier beispielsweise der Rechte der drei Anschlüsse nicht angeschlossen bleibt, und der linke Anschluss die Zuleitung zum Licht durch verbindet.

5.3 Das Haus

Das von der HTBLA Brauneau zur Verfügung gestellte Haus, zum Testen der SmartHome Anwendung, war einer der interessantesten und inspirierendsten Punkte des Projekts. Viele der neu hinzugefügten Funktionalitäten, wie das zeitliche Schalten von Ausgängen, sind auf die Herausforderungen und Aufgaben, die beim Verkabeln und Steuern eines solchen Hauses entstehen, zurückzuführen.



Abbildung 5.2: Das Haus

5.3.1 Verkabelung der Geräte

Die meiste Zeit der Arbeit am Haus ist in das Verkabeln der bereits verbauten Geräte geflossen. Im unteren Stockwerk befindet sich, wie in Abbildung 5.2 *Das Haus* zu sehen ist, ein Drehmotor, der sich je nach Stromrichtung nach links oder nach rechts dreht. Weiters befindet sich an der Oberkante des Raums ein LED - Streifen, welcher ebenfalls verkabelt werden muss. Im oberen Stock befindet sich wiederum ein Drehmotor mit einer Holzmatte, welche als Jalousie fungiert. Im oberen Stock befindet sich neben einem weiteren LED-Streifen auch die IP - Kamera für den Augmented Reality Livestream. Außen am Haus befindet sich auch noch eine LED - Beleuchtung und eine herkömmliche Klingel, welche jedoch in diesem Aufbau ohne Lautsprecher verkabelt wird.

Da das Haus an sich am gewöhnlichen 230V/50Hz Netz angeschlossen ist, die Geräte aber nur bis maximal 24V vertragen, ist hier im Haus ein Spannungswandler verbaut. Mit einem *LED Driver* wird hier von 230V auf 12V - 50W gewandelt. Das reicht sowohl zum Treiben der Motoren als auch zum Ansteuern der LED - Streifen aus. Die Schaltmodule an sich besitzen jeweils einen eigenen Spannungswandler von 230V auf 3.3V/5V und können somit direkt an das Netz gesteckt werden.

Angeschlossen werden die verschiedenen Geräte nun an die jeweiligen Spannungen, wobei diese mit einfachen Klemmen verteilt werden. Die meisten Geräte verwenden die digitalen Ausgänge eines Moduls, welche wie in Abbildung 5.1 *Anschluss Motor* zu sehen ist, angeschlossen werden. Eine Ausnahme bietet die Klingel, welche an den digitalen Eingang eines Moduls gelötet wird, um zu erkennen, wann die Klingel gedrückt wird und somit Spannung an diesem Pin anliegt. Mit einer Regel (siehe 3.2.11 *SmartHome Regeln einrichten* und 4.2.13 *SmartHome Regeln*) wird dann hier zur Veranschaulichung außen das Licht eingeschaltet.

5.3.2 Einrichten der Datenbank

Hinter einem physikalischen Haus steht immer eine Datenbank, welche die Beziehungen zwischen verschiedenen Geräten und Räumen im Haus modelliert und die Zustände und Einstellungen dieser Geräte speichert. Für jedes neue Haus muss auch diese Datenbank entsprechend eingerichtet werden bezüglich Räume, Bezeichnungen, Funktionen und anderen hausbezogenen Daten. Nachdem in der Universal SmartHome Webanwendung auch die Administratorsteuerung angeboten wird, kann man hier ganz einfach das Einrichten des Hauses vornehmen. Die Verwendung der Website zum Einrichten eines Hauses wird unter dem Kapitel 7 *Bedienungsanleitung*^{RB} beschrieben.

5.3.3 Basteln des Schreibtisches

Ein kleiner Teil, jedoch nicht unbedeutend für die Veranschaulichung, war das Basteln eines kleinen Schreibtisches mit Leselampe für den oberen Stock, besser gesagt das Schlafzimmer. Hier befindet sich im Testaufbau nämlich die IP - Kamera für das Steuern des Hauses per Augmented Reality Livestreams (siehe 3.2.8 *Kamera/Augmented Reality* und 4.2.14 *Livestreams & Augmented Reality*). Der Schreibtisch soll hier dazu dienen, das Steuern per Video - Livestream besser verständlich zu machen, durch das Einbinden von maßstabsgetreuen Elementen. Hier wird einfach eine LED als Schreibtischlampe verwendet, welche mit einem Vorwiderstand an ein Schaltrelais angeschlossen wird. Im Augmented Reality Livestream ist dieser Schreibtisch dann zu sehen und kann auch als eines der Hauptelemente gesteuert werden.

6 Anwendungsbeispiel^{RB}

In diesem Abschnitt der Diplomarbeit werden ein paar Beispiele aufgeführt, wie nun die verschiedenen Teile des Universal SmartHomes im Alltag verwendbar sind.

6.1 Am Morgen

Benutzer wird mittels Regel geweckt:

- Regel reagiert auf Sonnenaufgang
- Jalousien fahren nach oben
- Licht schaltet sich ein

Benutzer öffnet Applikation über Smartphone-Browser

- Schaltet Licht in der Küche ein
- Schaltet Kaffeemaschine ein

Benutzer fährt los zur Arbeit

- Schließt Garagentor aus dem Auto
- Benutzer ist sich unsicher, ob das Licht in der Küche ausgeschaltet ist
- Schaltet mittels Raum-Ansicht das Licht

6.2 In der Arbeit

Tochter schickt eine SMS, dass sie ihren Schlüssel vergessen hat

- Benutzer kann mittels Live-Stream sehen, wer vor der Tür steht
- Sieht nur seine Tochter vor der Tür stehen
- Öffnet Tür mittels AR-Ansicht (Augmented Reality)

6.3 Am Abend

Benutzer kommt von der Arbeit nach Hause

- Startet Applikation mit dem Smartphones
- Öffnet Garagentor mittels I/O-Ansicht
- Regel schaltet zusätzlich das Licht in der Garage ein

Benutzer will sich vor dem Schlafen entspannen

- Schaltet Licht im Badezimmer ein
- Schaltet Wasserhahn auf 30 °C ein

Auf der Startseite bemerkt der Benutzer, dass es morgen regnet

- Startet Administrator-Ansicht
- Deaktiviert Regel für die Gartensprinkler

7 Bedienungsanleitung^{RB}

Anhand dieser Anleitung wird dem Administrator beschrieben, wie die Konfiguration und Verwaltung der Webapplikation funktioniert. Die erste Einrichtung ist Pflicht für einen Großteil der Funktionalitäten des Universal SmartHomes.

7.1 Erste Einrichtung

7.1.1 Erstellen der Räume

Der erste Schritt erfolgt im Abschnitt der Raum-Übersicht. Anhand des grünen Plus-Knopfs kann ein Raum erstellt werden, wobei hier nur ein Name für den Raum benötigt wird. Wurde ein Raum mit falschem Namen erstellt, kann dieser gelöscht oder bearbeitet werden mittels der zwei rechten Knöpfe.

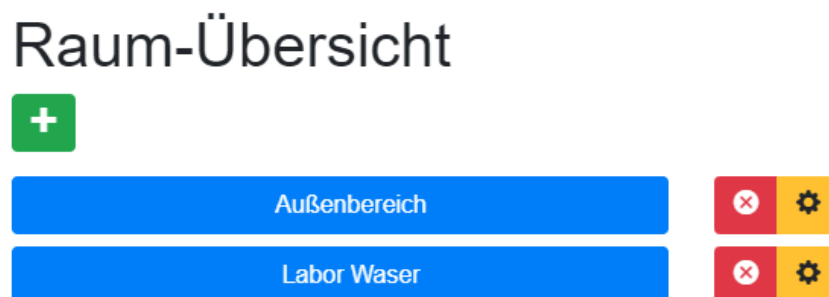


Abbildung 7.1: Anleitung-Räume

7.1.2 Modul-Konfiguration

Als nächsten Schritt wird jedes einzelne Modul konfiguriert, hierbei werden die Parameter der zugehörigen I/O's eingestellt. In den folgenden Tabellen werden die zu einzustellenden Eigenschaften genauer erklärt:

Digitaler Eingang:

PFLICHTFELD	PARAMETER	BESCHREIBUNG
JA	ID Digitaler Eingang	Identifikator des Eingangs (Selbstzuweisung)
	Beschreibung	Anzeigename gegenüber dem Benutzer
JA	Raum ID	Zuweisung auf zugehörigen Raum (Raum ID)
	Abkürzung	Bezeichnung innerhalb des Systems
	Abkürzung Aus	Anzeigetext wenn Signal <i>false</i> ist
	Abkürzung Ein	Anzeigetext wenn Signal <i>true</i> ist

Digitaler Ausgang:

PFLICHTFELD	PARAMETER	BESCHREIBUNG
JA	ID Digitaler Ausgang	Identifikator des Ausganges (Selbstzuweisung)
	Beschreibung	Anzeigename gegenüber dem Benutzer
JA	Raum ID	Zuweisung auf zugehörigen Raum (Raum ID)
	Abkürzung	Bezeichnung innerhalb des Systems
	Abkürzung Aus	Anzeigetext wenn Signal <i>false</i> ist
	Abkürzung Ein	Anzeigetext wenn Signal <i>true</i> ist
	Impulszeit	Pulsdauer bei Aktivierung des Tasters (Standardwert 0 : Pulsdauer 500ms)
JA	Schalter / Taster	Auswahl zwischen Schalter oder Taster

Analoger Eingang:

PFLICHTFELD	PARAMETER	BESCHREIBUNG
JA	ID Analoger Eingang	Identifikator des Eingangs (Selbstzuweisung)
	Portnummer	Signalport am Modul
	Modul ID	Zuweisung auf zugehöriges Modul (Selbstzuweisung)
	Beschreibung	Anzeigename gegenüber dem Benutzer
	Einheit	Einheit des Analogsignals
	Modul: min. Werteb.	Realer Minimalwert des Signals in Volt
	Modul: max. Werteb.	Realer Maximalwert des Signals in Volt
	Global: min. Werteb.	Minimalwert des Signals nach Gerät (z.B. Thermometer)
	Global: max. Werteb.	Maximalwert des Signals nach Gerät (z.B. Thermometer)
JA	Raum ID	Zuweisung auf zugehörigen Raum (Raum ID)
	Abkürzung	Bezeichnung innerhalb des Systems

Analoger Ausgang:

PFLICHTFELD	PARAMETER	BESCHREIBUNG
JA	ID Analoger Ausgang	Identifikator des Ausgangs (Selbstzuweisung)
	Beschreibung	Anzeigename gegenüber dem Benutzer
	Einheit	Einheit des Analogsignals
	Global: min. Werteb.	Minimalwert des Signals nach Gerät (z.B. Thermometer)
	Global: max. Werteb.	Maximalwert des Signals nach Gerät (z.B. Thermometer)
JA	Raum ID	Zuweisung auf zugehörigen Raum (Raum ID)

PFLICHTFELD	PARAMETER	BESCHREIBUNG
	Abkürzung	Bezeichnung innerhalb des Systems

7.2 Verwaltungstools

Alle Elemente werden verwaltet anhand drei verschiedener Arten: Erstellen, Löschen oder Bearbeiten. Die genaueren Möglichkeiten werden in diesem Abschnitt beschrieben.

7.2.1 Erstellen

Folgende Elemente können erstellt werden:

- Räume
- Kameras
- Augmented Reality I/O
- Regeln

Ein Raum benötigt nur den Namen, die Regeln werden in der gleichnamigen Überschrift genauer erläutert. Die Erstellung einer Kamera benötigt folgende Parameter:

PFLICHTFELD	PARAMETER	BESCHREIBUNG
JA	Identifikator	Datenbank-ID der Kamera
	Model	Modeltyp der Kamera
	Beschreibung	Anzeigenname gegenüber dem Benutzer
JA	IP	Netzwerkadresse der Kamera
JA	Raum ID	Zuweisung auf zugehörigen Raum (Raum ID)
JA	Auflösung X	Breite der Kamera-Auflösung (in Pixel)
JA	Auflösung Y	Höhe der Kamera-Auflösung (in Pixel)

Nach der Erstellung einer Kamera können in der Kamera - Einzelansicht Augmented Reality I/O's erstellt werden. Anhand eines Dropdown - Menüs wird die Gattung sowie das zu steuernde Gerät ausgewählt. Danach muss nur mehr der Rahmen über das dementsprechende Gerät im Bild gezogen werden.

7.2.2 Löschen

Folgende Elemente können gelöscht werden:

- Räume
- Kameras
- Augmented Reality I/O
- Module
- Regeln

Wichtig ist zu beachten, dass jedes Element auch in der Datenbank gelöscht wird, bedeutet im Falle eines Moduls ist der Wiederherstellungsprozess sehr umständlich.

7.2.3 Bearbeiten

Folgende Elemente können bearbeitet werden:

- Räume
- Kameras
- I/O's jeder Gattung
- Module
- Regeln

Die Beschreibung der verschiedenen Parameter ist im Abschnitt Modul-Konfiguration (siehe *7.1.2 Modul-Konfiguration*) zu finden.

7.3 Regeln

Regeln werden in drei verschiedene Typen unterteilt: *EVENT*, *TIMER* und *NOTIFICATION*. Der Auslöser wird mittels boolescher Algebra definiert. Ein *EVENT* steuert bei Veränderung des zugewiesenen Signals. Der *TIMER* ist eine Zeitsteuerung von Elementen und die *NOTIFICATION* sendet eine Nachricht an den Benutzer, wenn die dazugehörige Bedingung erfüllt ist.

Wenn eine Regel nun erstellt wird, benötigt diese einen String mit folgender Struktur:

<Typ> "<I/O-Zuweisung>" <Operator> "<Bedingung>"

Im Falle der *NOTIFICATION* ist die I/O-Zuweisung der String für die Nachricht. Der Operator kann entweder gleich oder ungleich sein (= / !=). Die Bedingung wird bei Verwendung des Timer-Typs zur Zeitangabe, mit folgender Struktur:

<Startzeit>#<Wochentag> <Endzeit>#<Wochentag> .

Die Wochentage sind im Kurzformat, für die Einstellung von mehr als einem Tag wird ein weiterer mittels Raute hinzugefügt. Ist der Typ *EVENT* oder *NOTIFICATION*, wird die Bedingung mittels logischer Operatoren definiert.

```
TIMER "Wohnzimmer#Wohnzimmer-Licht" = 15:17#Mo#Di#Mi#Do#Fr#Sa#So 15:18#Mo#Di#Mi#Do#Fr#Sa#So
```

```
NOTIFICATION "Das ist eine Testnachricht" = "Labor Waser#Tor" && "Sonstiges#M3_DigIn"
```

```
EVENT "Labor Waser#LED" = "Labor Waser#Tor" < 2
```

Abbildung 7.2: Regeltypen

8 Fazit und Persönliche Erfahrungen

8.1 Fazit

Am Ende der Entwicklung des Universal SmartHomes befindet sich ein vollständig konfigurierter Testaufbau eines Hauses mit diversen Möglichkeiten zur intelligenten Steuerung eines Haushaltes. Angefangen vom Aus- und Einschalten von Lichtern, über das Steuern von bidirektionalen Motoren bis hin zu Livevideos bestimmter Räume ist alles dabei. Die Funktionalitäten vorheriger Versionen nachzubilden, war einer der einfachsten, jedoch zeitaufwendigsten Teile der Arbeit. Letztendlich war jedoch auch hier das Steuern aller möglichen Geräte vollständig implementiert.

Große Erweiterungen gab es vor allem im Bereich Augmented Reality, wofür es jetzt erstmals eine einfache Möglichkeit gibt, hier Geräte einzurichten, wodurch das rein graphische Steuern des Hauses sehr viel leichter möglich ist. Einen der größten Erfolge stellt hierbei auch die Verwendung von SignalR für das Anzeigen von Livedaten dar. Diese Neuerung ermöglicht fortan, dass mehrere Clients, welche das SmartHome gleichzeitig steuern wollen, immer den selben Stand der Zustände der Geräte vor sich haben.

Bei den Video - Livestreams wurde auch schnell klar, dass die zuvor verwendete Methode zum Anzeigen des Videos sehr viele Schwierigkeiten (vor allem Performance Probleme) bereitet. Hier wurde erneut durch SignalR eine Lösung gefunden, da so die ressourcenaufwendige Arbeit des Umwandelns der Kameradaten in Bilder auf den Server ausgelagert werden kann. Durch effektives Arbeiten blieb am Ende auch noch Zeit, vor allem die Benutzerfreundlichkeit der Website sehr zu verbessern. So wurde jede überflüssige Interaktion vermieden, um dem Benutzer eine einfachere Oberfläche zu bieten. Weiters war Zeit, Probleme wie mögliche Sprachbarrieren zu umgehen, durch die Verwendung vieler Symbole anstatt Text. Der Text, den man nicht ersetzen konnte, wird durch Multilanguage - Tools in die Sprache übersetzt, in welcher der Browser aufgesetzt ist, um jedem in seiner Muttersprache Zugang zu solcher Technik zu verschaffen.

Um Probleme zu lösen, wurde hier meist der offensichtlichste Weg gewählt, welcher sich erstaunlicherweise meist als der einfachste erwiesen hat. Das hat auch einiges mit dem Ziel zu tun, die Oberfläche so einfach wie nur möglich zu halten. Hier ist der erste Lösungsweg, der einem in den Sinn kommt, wohl auch der Weg, den ein Benutzer bei der Bedienung am ehesten einschlagen würde.

Würde man dieses Projekt noch weiterführen, wäre eines der nächsten Ziele das Verschönern der Benutzeroberfläche. Hier ist noch sehr viel Spielraum, mehr Symbolik zu verwenden, größere Schaltflächen für möglicherweise betagtere Menschen einzuführen und generell die Benutzererfahrung bei der Verwendung zu verbessern.

8.2 Persönliche Erfahrungen

Projekte in solch einer Größe können zu Anfang sehr aufwendig wirken. Man hat viele Aufgaben vom Projekt an sich und zusätzlich noch Zeit, die man für die Diplomarbeit aufwenden muss. Das Wichtigste ist hier die Vorbereitung in welcher Form auch immer. Da wir bereits größere Projekte im Vorjahr(Werkstättenprojekt) angegangen sind, war uns von Anfang an klar, dass wir hier einen genauen Plan brauchen, um das Ziel nicht aus den Augen zu verlieren. Denn solange man weiß, worauf man hinarbeitet, kann man sich nach Niederschlägen wieder Richtung Ziel ausrichten. Zu unserem Glück haben wir im Unterricht mehrere Konzepte des Projektmanagements durchgemacht und wurden hier auch direkt beim Erarbeiten verschiedener Konzepte für dieses Management unterstützt. Der Bezug zum Projekt war hier immer gegeben, wodurch die eher trockene Theorie dahinter sehr leicht greifbar wurde.

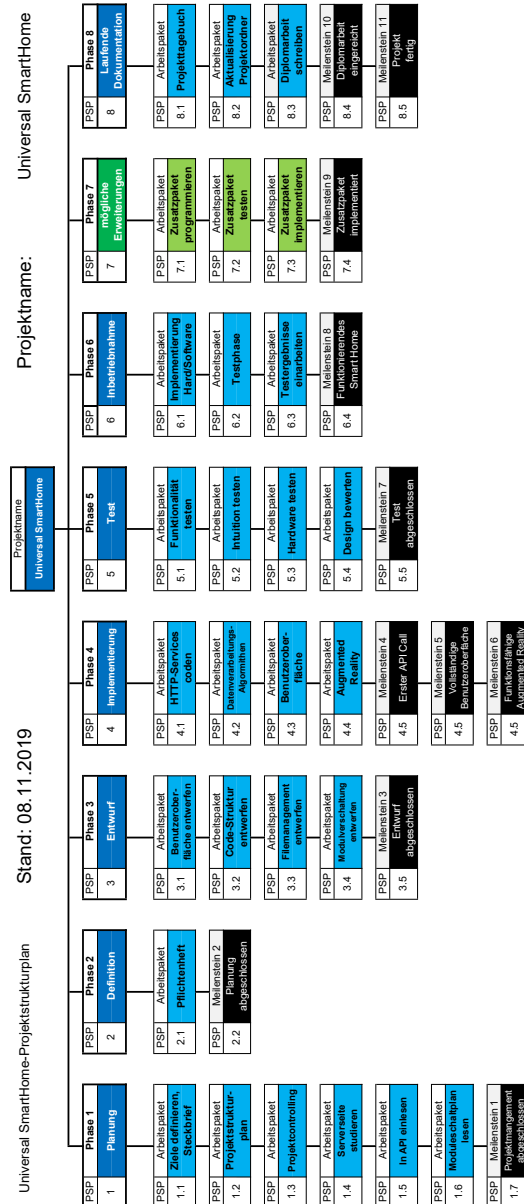
Der Projektstrukturplan definiert die groben Ziele genauer und gibt eine bessere Übersicht. Dadurch, dass wir mit dem Werkstättenprojekt schon ein Softwareprojekt über ein Jahr verfolgt haben, hatten wir bereits ein besseres Zeitgefühl für Aufgaben sowie Terminfristen. Die Einteilung der Ziele sowie das Setzen von Meilensteinen bringen im Laufe des Projekts immer wieder kleine Erfolgserlebnisse. Diese geben einem Motivation weiterzumachen und nach einiger Zeit sieht man wirkliche Fortschritte. Man sollte also nicht zurückschrecken und mit guter Vorbereitung in das Projekt gehen.

Auch wichtig ist zu erwähnen, dass wir es auch hier nicht geschafft haben, keine Fehler zu machen. Es gab immer wieder Momente, in denen klar wurde, dass geschriebener Code so einfach nicht sehr leicht zu handhaben war. Wir mussten uns oft dazu bringen Stunden von Arbeit zu verwerfen und gewisse Teile neu schreiben. Im Nachhinein war uns jedoch jedes Mal klar, dass es die richtige Entscheidung in Hinsicht auf die Zukunft war. Man darf also hier, obwohl es gegen das eigene Bauchgefühl geht, nicht davor zurückschrecken Fehler einzugestehen und sie am besten zeitnah zu beheben. Das ist vor allem in der Softwareentwicklung wichtig, denn umso komplexer ein Projekt wird, desto schwerer lassen sich solche Fehler beheben, was oft verheerende Folgen nach sich ziehen kann.

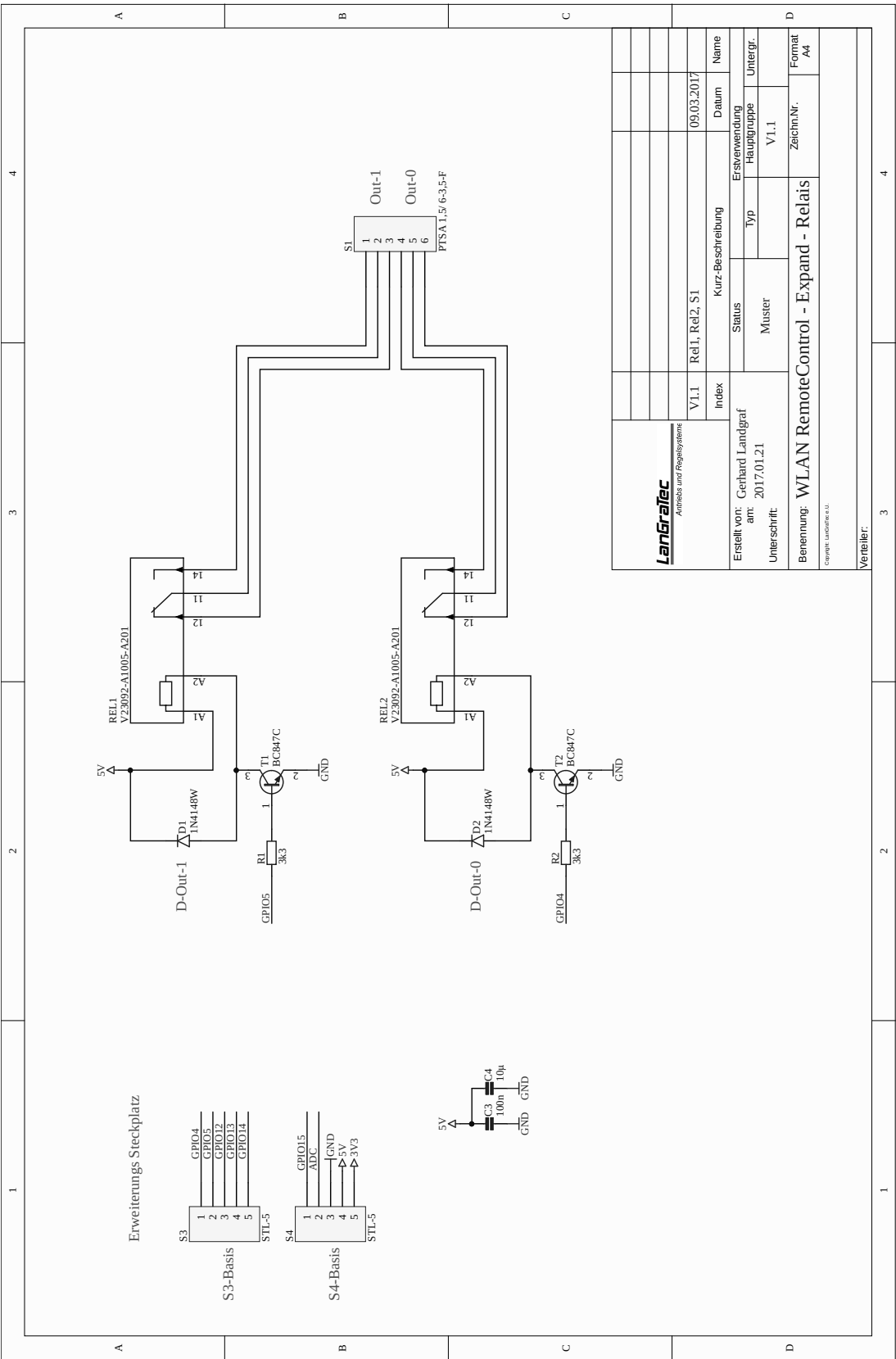
Abschließend ist noch zu sagen, dass obwohl man sich in der Abschlussklasse einer HTL befindet, man trotzdem noch lange nicht ausgelernt hat, was uns auch dieses Projekt wieder vor Augen geführt hat. Man lernt in der Softwareentwicklung enorm viel dazu, wenn man sich, wenn auch nur kurz, persönlich in einem Projekt damit beschäftigt. Man stößt auf Probleme, die man zuvor noch nie hatte und lernt diese zu lösen. Wir haben also nicht nur das Universal SmartHome zu mehr gebracht, als es bisher war, sondern auch umgekehrt.

A Diverse Anhänge

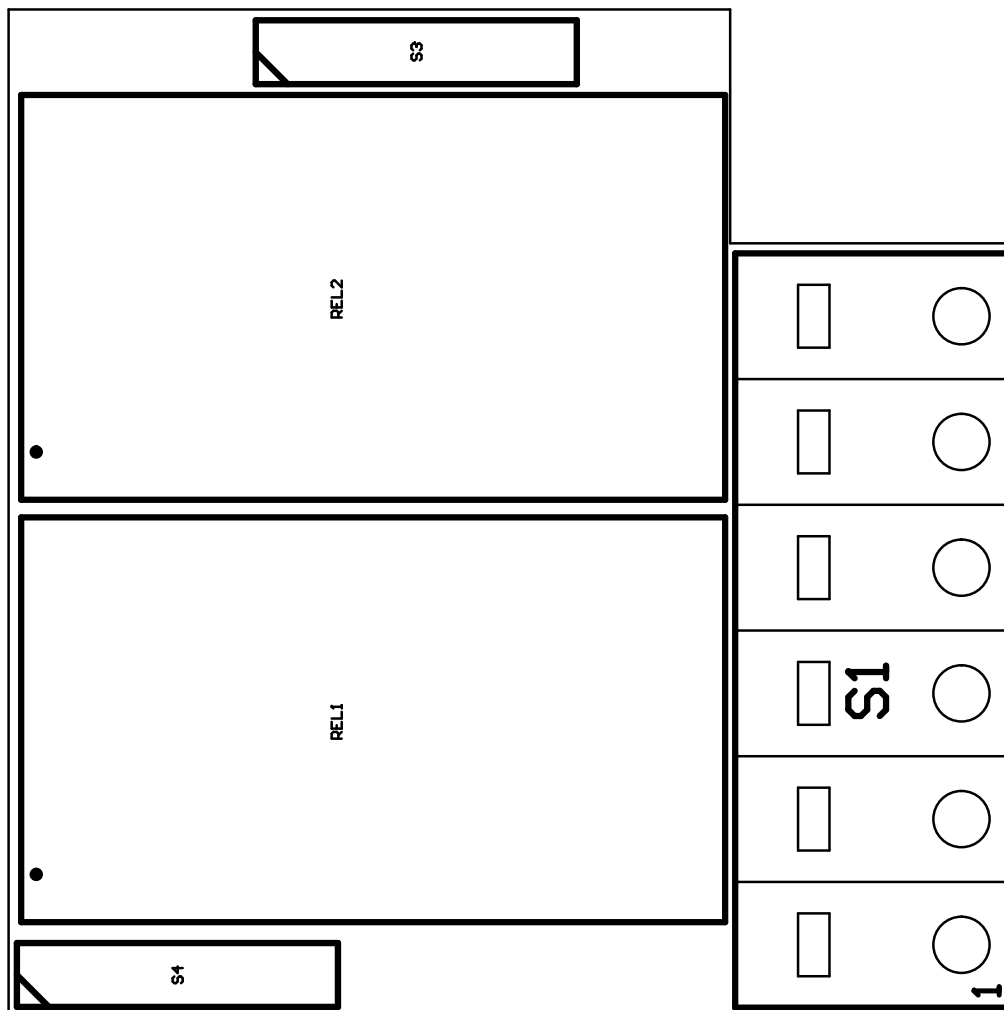
A.1 Projektstrukturplan



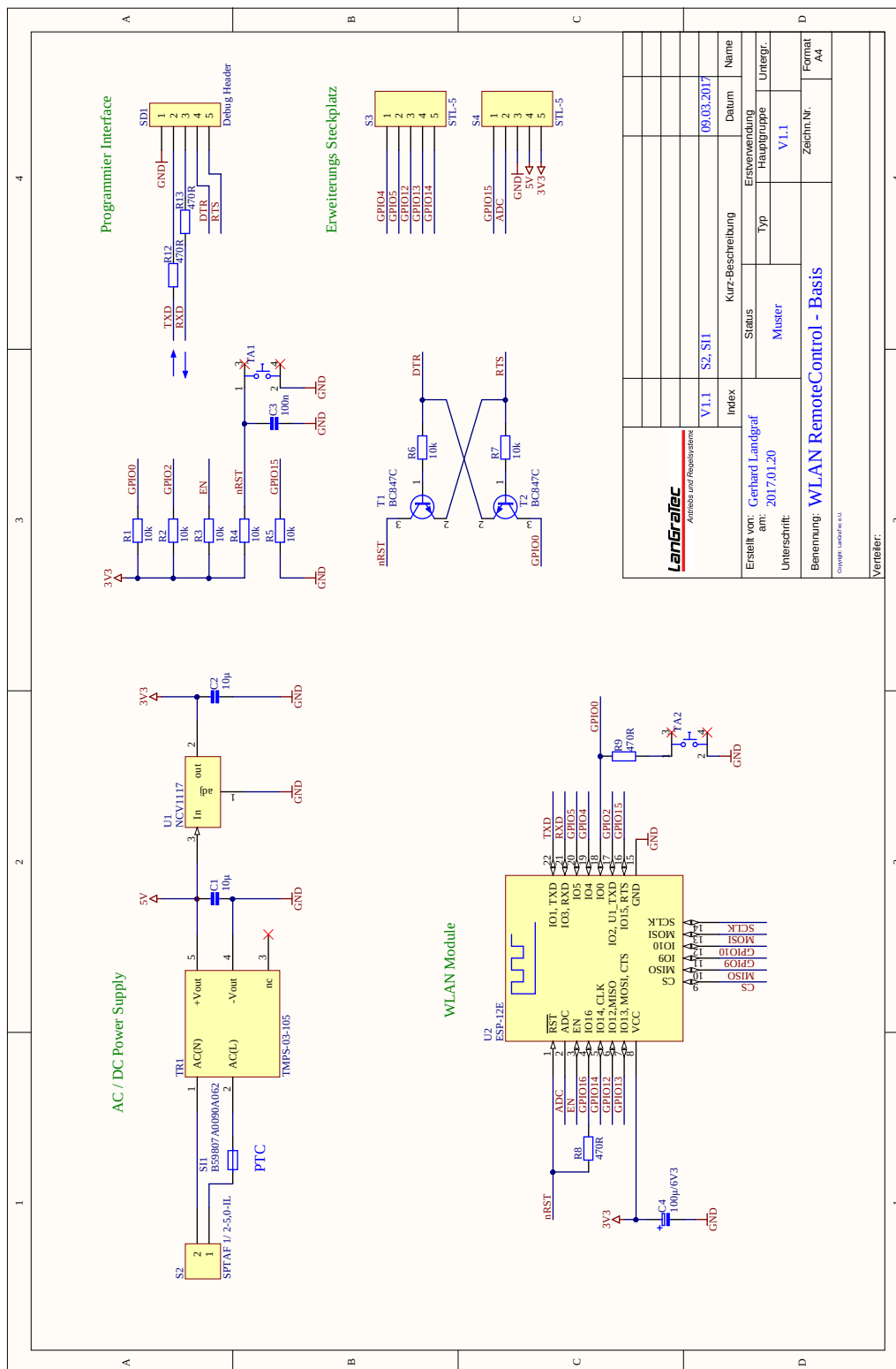
A.2 Relais-Schaltplan



A.3 Relais-Skizze



A.4 Modulbasis-Schaltplan



A.6 Projekttagbuch

A.6.1 Baischer

SCHULE	FREIZEIT	TÄTIGKEIT
	15	Blazor Einarbeitung
11	5	Code- & Filestruktur
23		Klassen-Implementierung
16	9	Universal SmartHome API
13	14	Livestreams & Augmented Reality
4	6	JavaScript - C# Brücke
3		Internet Information Service
7	3	SignalR
10		Haus-Installation
28	15	Projektmanagement
10	79	Diplomarbeit

Summe Schulstunden: 125

Summe Freizeitstunden: 146

A.6.2 Brachmaier

SCHULE	FREIZEIT	TÄTIGKEIT
	14	Blazor Einarbeitung
6	4	Bootstrap Einarbeitung
34	9	User-Interface
13	5	MultiLanguage
11		Unterschied Admin/User

SCHULE	FREIZEIT	TÄTIGKEIT
10		Haus-Installation
28	15	Projektmanagement
10	77	Diplomarbeit

Summe Schulstunden: 112

Summe Freizeitstunden: 124

Literaturverzeichnis

- [1] Microsoft Corporation, *Blazor*, <https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor>,
- [2] Microsoft Corporation, *ASP .NET Core*, <https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-3.1>,
- [3] Microsoft Corporation, *SignalR*, <https://docs.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>,
- [4] Newton Software, *Newtonsoft JSON*, <https://www.newtonsoft.com/json>,
- [5] Twitter Inc., *Bootstrap*, <https://getbootstrap.com/>,
- [6] J.Sakamoto, *MultiLanguage*, <https://www.nuget.org/packages/Toolbelt.Blazor.I18nText/>
- [7] Chris Sainty, *Toasts*, <https://www.nuget.org/packages/Blazored.Toast/>
- [8] Refsnes Data, *w3schools* <https://www.w3schools.com/>
- [9] Openweather Ltd., *Wetter API* <https://openweather.co.uk/>
- [10] E-Control, *Spritpreisrechner API* <https://spritpreisrechner.at/>
- [11] Google LLC, *Google GeoCoding* <https://developers.google.com/maps/documentation/geocoding/intro?hl=de>
- [12] Microsoft Corporation, *Dependency Injection* <https://docs.microsoft.com/de-at/aspnet/core/blazor/dependency-injection?view=aspnetcore-3.1>
- [13] Microsoft Corporation, *Lifecycle Methods* <https://docs.microsoft.com/de-de/aspnet/core/blazor/lifecycle?view=aspnetcore-3.1>
- [14] Microsoft Corporation, *Internet Information Service - IIS* <https://docs.microsoft.com/de-at/iis/extensions/introduction-to-iis-express/iis-express-overview>

Abbildungsverzeichnis

0.1	Bild der Startseite	iv
0.2	Picture of the Homepage	vi
2.1	Bootstrap-Spalten	8
2.2	Tabellen-Unterschied	9
2.3	Bootstrap-Knöpfe	10
2.4	Bootstrap-Dropdowns	10
2.5	Bootstrap-Forms	10
2.6	Multilanguage Vergleich	13
3.1	Erstes Konzept	14
3.2	Startseite	15
3.3	Kamera Website	16
3.4	Wetter Widget	18
3.5	Navigationsleiste	19
3.6	Kamera-Ansicht	20
3.7	Übersicht der Regeln	21
3.8	Fehler-Toast	22
4.1	Filestruktur	24
4.2	IIS - Manager	44
5.1	Anschluss Motor	48
5.2	Das Haus	49
7.1	Anleitung-Räume	53
7.2	Regeltypen	58

Listings

2.1	Blazor Syntax	4
2.2	Startup Logik	5
2.3	Counter Beispiel	5
2.4	Bootstrap-Raster	7
2.5	MultiLanguage JSON-Datei Deutsch	11
2.6	MultiLanguage JSON-Datei Englisch	12
2.7	MultiLanguage im Code	12
4.1	HTTP-Request	25
4.2	JSON-Codieren	28
4.3	JSON-Decodieren	28
4.4	@Bind Attribut	28
4.5	@Bind Attribut Code Behind	28
4.6	Inheritance	29
4.7	Dependency Injection	29
4.8	Dienst Hinzufügen	29
4.9	Klasse CDigitalInput	31
4.10	OnInitializedAsync Lifecycle Methode	32
4.11	URL-Parameter	32
4.12	SmartHome Regeln	33
4.13	Video - Stream starten	34
4.14	Frame anzeigen	34
4.15	Mouse Events	35
4.16	Augmented Reality steuern	36
4.17	JavaScript Interoperationen	37
4.18	ExpandoObject	38
4.19	SignalR-Hub Anmeldung	38
4.20	SignalR-Hub Callback Registrierung	39
4.21	Modul Codierung	40
4.22	SignalR Callback Methode	42
4.23	IDisposable	43
4.24	Wetter API Call	45
4.25	Errorhandling im Code	46

Autoren

Johannes Baischer

Geburtstag, Geburtsort: 08.11.2000, Oberndorf
Schulbildung: Volksschule Moosdorf
Musikhauptschule Eggelsberg
HTL Braunau
Praktika: AMO, 4 Wochen, Hardwareentwicklung
cts GmbH, 5 Wochen, Webentwicklung
Anschrift: Furkern 7
5141, Moosdorf
Österreich
E-Mail: johannes.baischer@gmail.com
johannes.baischer@htl-braunau.at



Rahael Brachmaier

Geburtstag, Geburtsort: 06.09.2000, Oberndorf
Schulbildung: Volksschule Ostermiething
Hauptschule Ostermiething
HTL Braunau
Praktika: cts GmbH, 5 Wochen, Webentwicklung
cts GmbH, 6 Wochen, Softwareentwicklung
cts GmbH, 4 Wochen, Hardwareentwicklung
Anschrift: Ziegelei 41
5121, Ostermiething
Österreich
E-Mail: raphael.brachmaier@gmail.com
raphael.brachmaier@htl-braunau.at

