

Raspberry Poolüberwachung

DIPLOMARBEIT

verfasst im Rahmen der

Reife- und Diplomprüfung

an der

Höheren Abteilung für Informatik

Eingereicht von
Sebastian Egger
Florian Wilflingseder

Betreuer:
Michael Wagner
Gerald Köck

Leonding, April 2022

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die Arbeit wurde bisher in gleicher oder ähnlicher Weise keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Leonding, April 2022

S. Egger & F. Wilflingseder

Zur Verbesserung der Lesbarkeit wurde in diesem Dokument auf eine geschlechtsneutrale Ausdrucksweise verzichtet. Alle verwendeten Formulierungen richten sich jedoch an alle Geschlechter.

Zusammenfassung



Abbildung 1: Grafische Darstellung des Projektes

Inhaltsverzeichnis

1	Einleitung	1
2	Umfeldanalyse	2
3	Technologien	3
3.1	Raspberry Pi	3
3.2	MQTT	5
3.3	.Net	7
3.4	Telegram	14
3.5	Begriffserklärungen	17
4	Projektumsetzung	18
4.1	Projektmanagement	18
4.2	Backend Projekt-Überblick	20
5	Zusammenfassung	43
	Literaturverzeichnis	IV
	Abbildungsverzeichnis	V
	Tabellenverzeichnis	VII
	Quellcodeverzeichnis	VIII
	Anhang	IX

1 Einleitung

Unsere Diplomarbeit wurde in das Leben gerufen, um einen Pool überwachen zu können. Aktuell sind mehrere Geräte für die Überwachung eines Pools erforderlich. Ziel unserer Diplomarbeit ist es, die Funktionalitäten von einem Trübungssensor, Wellengangssensor und Temperatursensor zuverlässig in einem Gerät kosteneffizient zusammenzuführen und über ein UserInterface der Single Page Application einen 360 Grad Blick auf die Geschehnisse im Pool zu ermöglichen.

2 Umfeldanalyse

Florian Wilflingseder

3 Technologien

3.1 Raspberry Pi

Das Projekt beinhaltet einen Raspberry Pi 4, welcher als MQTT Broker dient. Auf diesem Raspberry Pi läuft unser Backend mit DotNet, Docker und Samba. Der Raspberry Pi hat 4 Gigabyte RAM und eine 32 Gigabyte SSD. Die Verbindung zwischen dem Raspberry und der SSD wird mit einem USB-Adapter hergestellt. Die SSD wurde unter dem Raspberry mittels einer Platine und Schrauben befestigt. Der Raspberry braucht mindestens 3 und maximal 11 Watt.

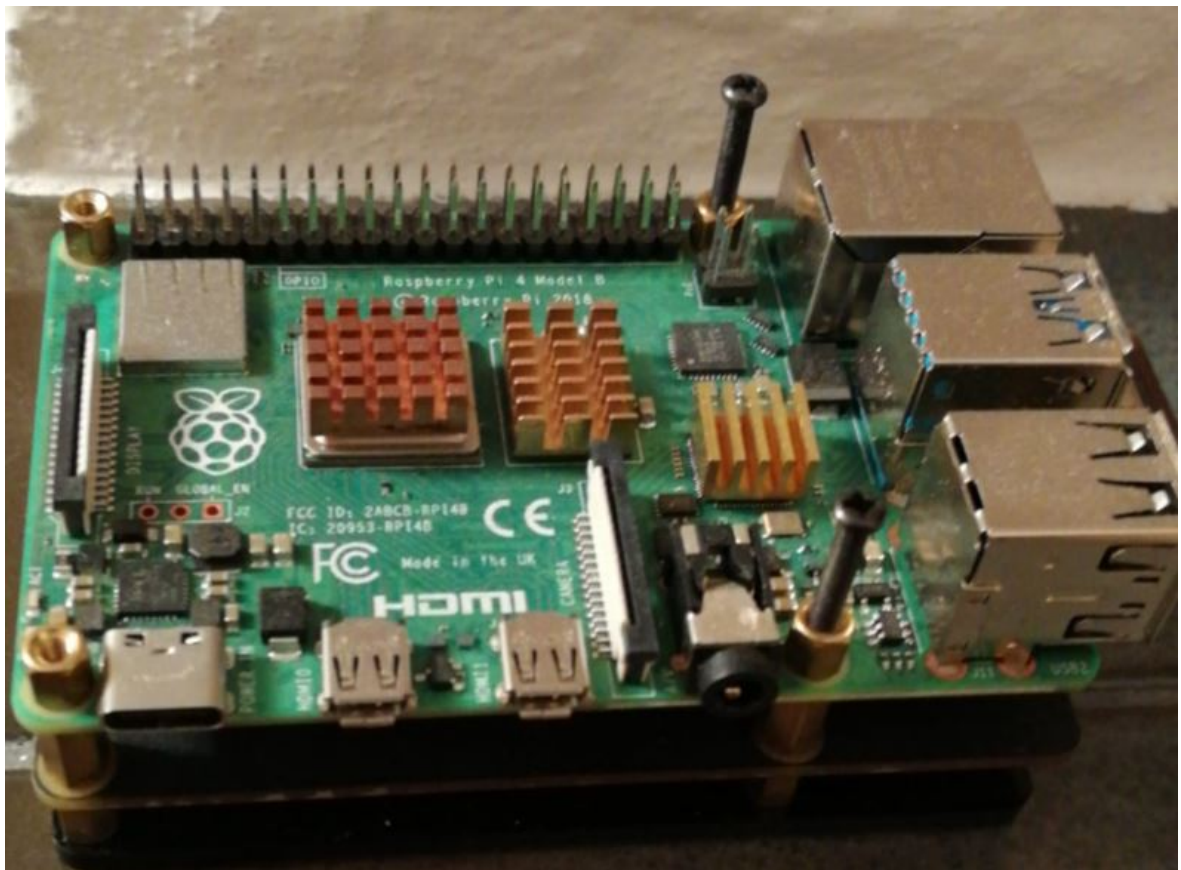


Abbildung 2: Raspberry Pi des Projektes

3.1.1 Samba

Der Raspberry dient weiters als File-Server. Für eine leichtere Datenübertragung zwischen Windows und Linux wird mit Hilfe von Samba über den Windows Explorer direkt auf den Raspberry Pi zugegriffen. Somit können Files oder Projekte direkt von einem Laptop oder Computer auf den Raspberry PI gelegt werden.

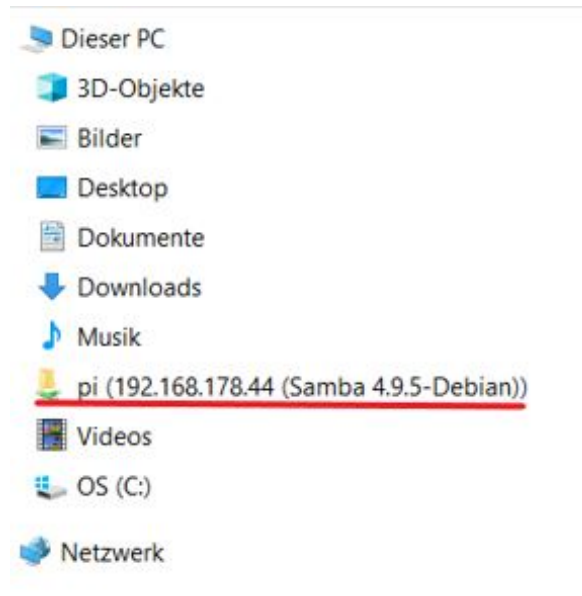


Abbildung 3: Samba auf Laptop

3.1.2 MQTT auf Raspberry Pi:

Weiteres dient unser Raspberry auch als MQTT-Broker, welcher Messwerte von Sensoren empfängt und an das Backend, welches als MQTT-Client dient, übermittelt. Genauerer wie MQTT aufgebaut ist und in welcher Verbindung der MQTT-Broker zu den MQTT-Clients steht wird im Kapitel MQTT beschrieben.

3.1.3 Installation und Verwendung von Docker auf dem Raspberry Pi:

Docker ist eine Software, welche das Management von Containern übernimmt. Ein Container enthält alle Dateien, die zum Ausführen einer Software notwendig sind. Die Installation von Docker wird über ein Skript durchgeführt. Dieses wird direkt von Docker zur Verfügung gestellt und führt alle Schritte automatisch ohne weitere Eingaben vom Benutzer durch. Nach wenigen Minuten ist Docker betriebsbereit. Weiters ist auch eine zentralisierte Servicebereitstellungsplattform für containerisierte Apps mit dem Namen

Portainer auf dem Raspberry Pi installiert, welche eine Liste aller Container und deren Informationen zur Verfügung stellt.

3.1.4 Remote Access auf einen Raspberry Pi:

Bei Verwendung des Raspberry Pi ohne direkt angeschlossenen Monitor kann mittels SSH (Secure Socket Shell) Protokoll von einem Laptop zugegriffen werden. Dabei muss die IP-Adresse des Raspberry's im Netzwerk bekannt sein. WLAN-Router (FRITZ!Box) bieten über ihre standard IP-Adresse eine Übersicht der verbundenen Geräte, wo auch unter anderem der verbundene Raspberry Pi angezeigt wird.

3.1.5 Was sind Ip-Adressen:

Im oberen Kapitel Remote Access ist oft das Wort Ip-Adresse gefallen, deswegen wird in diesem Unterpunkt eine kleine Einführung über Ip-Adressen und Ihre Verwendung gegeben. Eine Ip-Adresse ist eine Adresse in Computernetzen, welche von einem Router vergeben wird. Einem Gerät kann maximal eine Ip-Adresse zugewiesen werden, jedoch kann die Ip-Adresse auch wechseln, wenn sich das Gerät zum Router erneut verbindet. Im Router gibt es aber auch die Konfigurationsmöglichkeit, dass einem Gerät immer eine bestimmte Ip-Adresse zugewiesen wird.

3.2 MQTT

3.2.1 Was ist MQTT:

MQTT ausgesprochen Message Queuing Telemetry Transport ist ein Protokoll, welches Nachrichten von einer Maschine zu einer anderen Maschine schickt. Ein MQTT Netzwerk besteht aus mindestens einem MQTT-Broker und zwei MQTT-Clients. Wenn ein MQTT-Client eine Message an einen anderen MQTT-Client senden will, muss als erstes eine Message zu dem MQTT-Broker, welcher die Message zu einem sogenannten Topic zuweist, gesendet werden. Ein Topic ist ein Bereich, wo bestimmte Nachrichten aufgelistet werden. Ein Topic in unserem Fall lautet mqtt/noice für den Noice-Sensor. Wenn ein oder mehrere MQTT-Clients diese Nachricht empfangen wollen, dann subscriben diese auf das Topic. Durch das subscriben von den Clients werden diese, sobald eine neue Message an das Topic gesendet wurde, benachrichtigt und können diese nun empfangen.

3.2.2 Verwendung von MQTT in unserem Projekt:

Unser Projekt besteht aus 2 MQTT-Clients und 1 MQTT-Broker. Die Sensor Box ist ein MQTT-Client, welcher die Messwerte an den MQTT-Broker sendet. Der MQTT-Broker ist im Projekt der Raspberry PI. Zum Empfangen der Daten liest das Backend, welches den zweiten MQTT-Client darstellt, die Daten vom Raspberry ein.

3.2.3 MQTT-Explorer:

MQTT-Explorer ist eine kostenlose Software, welches sich für das Testen einer Connection zwischen MQTT-Client und Broker bestens eignet. Der MQTT-Explorer ist ein weiterer MQTT-Client. Zur Benutzung und Verbindung mit dem Raspberry ist ein Login mit der Ip-Adresse und Port des Brokers sowie dem dazugehörigen Benutzernamen und Passwort notwendig.

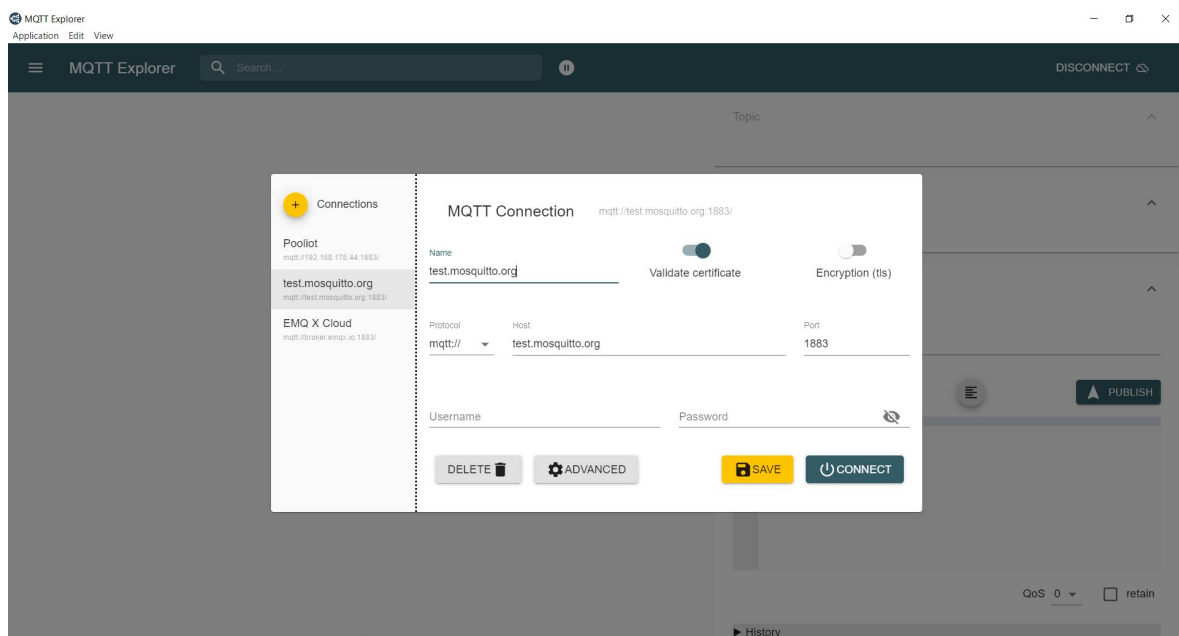


Abbildung 4: MQTT-Explorer Login

Sobald eine Verbindung zu einem MQTT-Broker möglich ist, wird ein Screen mit dem Namen des Brokers und den dazugehörigen Topics aufgelistet. In unten gezeigter Abbildung wurde eine Verbindung mit dem MQTT-Broker test.mosquitto.org hergestellt.

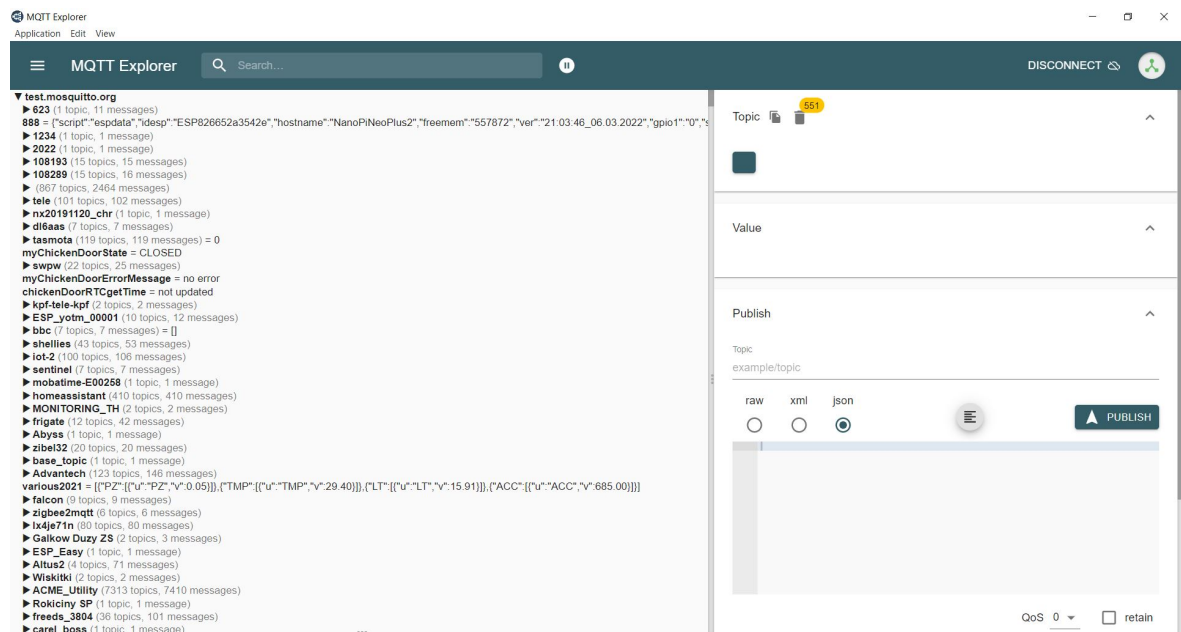


Abbildung 5: MQTT-Explorer nachdem Login

3.3 .Net

3.3.1 .Net Core und .Net Framework

In unserem Projekt verwenden wir .Net Core, eine alternative dazu wäre .NetFramework. Beides sind Frameworks, welche für die Erstellung serverseitiger Anwendungen verwendet werden. .Net Core und .Net Framework haben manche funktionale Komponenten gemeinsam und können Ihren Code auch über die beiden Plattformen hinweg gemeinsam nutzen.

Was wird unter .Net Framework und .Net Core verstanden

Bisher wurde .NetFramework genutzt um sowohl .NET Desktop-Anwendungen als auch serverbasierte Anwendungen zu erstellen. .NET Core wiederum dient der Erstellung von Serveranwendungen, die unter Windows, Linux und Mac laufen. Bei .NET Core handelt es sich in erster Linie um ein Open-Source-Framework mit Multiplattform-Unterstützung.

.Net Framework Vorteile

Während .NET Core die Zukunft der Anwendungsentwicklung ist und das .NET Framework in der Zukunft ablösen wird, wird die jahrelange Verwendung von .NET Framework

nicht so schnell vergehen! Spricht man über .NET Framework vs. .NET Core im heutigen Kontext, so hat .NET Framework immer noch einige praktische .NET-Vorteile.

.Net Core benötigt derzeit für unerfahrene Programmierer einen größeren Lernaufwand als bei .Net Framework. Ein weiterer Vorteil gegenüber .Net Core ist die Wartung bestehender Anwendungen. Nachdem .Net Framework sich seit Jahren im Einsatz befindet, wurden die meisten .Net Anwendungen mittels .Net Framework geschrieben. Ein weiterer großer Vorteil von .Net Framework ist die Stabilität der Plattform.

Microsoft hat verkündet, dass die aktuelle Version des Microsoft .NET Framework (Version 4.8) die letzte Version sein wird und es danach kein größeres .NET Framework-Update mehr geben wird. Das bedeutet die Entwicklung von .Net Framework schreitet nicht mehr voran, aber somit können auch keine durch Updates verursachten Bugs mehr entstehen.

.Net Core Vorteile

Wie bereits oben erwähnt ist .Net Core ein Open-Source-Framework, welches sich kontinuierlich durch offene Beiträge verbessert. Ein weiterer Vorteil ist die plattformübergreifende Kompatibilität, damit bei Verwendung eines Computers mit Linux oder macOS als Betriebssystem trotzdem Apps, welche unter Windows laufen, verwendet werden können. Mit .Net Core entwickelten Apps wird Benutzern die Möglichkeit geboten, diese Apps nicht nur unter Windows zu verwenden, sondern auch auf Linux oder macOS. Diese Flexibilität entfällt bei Entwicklungen mit .Net Framework.

Es gibt natürlich noch einige weitere Vorteile von .Net Core, die hier nicht erwähnt wurden.

3.3.2 Visual Studio Code

Visual Studio Code ist ein Code Editor mit vielen Funktionen. Wie bereits im Kapitel .Net angesprochen, gibt es sehr viele Extensions in diesem Editor, welche zum Beispiel eine Visualisierung von Inhalten einer Sqlite Datenbank ermöglicht oder als Latex Editor genutzt werden kann.

3.3.3 Visual Studio

Visual Studio ist ein Code Editor, welcher von Microsoft entwickelt wurde, um eine benutzerfreundliche Programmierumgebung anzubieten. Seit 2008 wird jedes zweite Jahr ein neues Visual Studio mit mehr Funktionen, welche dem Benutzer beim Programmieren das Leben vereinfachen, veröffentlicht.

Was ist Swagger und Verwendung von Swagger in unserem Projekt

Swagger ist eine Sammlung von Open-Source-Werkzeugen, um HTTP-Webservices (auch HTTP API oder REST-like API) zu entwerfen, zu erstellen, zu dokumentieren und zu nutzen. In diesem Projekt wurde Swagger verwendet, um die API zu beschreiben und zu testen. Swagger bietet nicht nur die Zusammenarbeit mit C# an sondern auch mit anderen Programmiersprachen wie zum Beispiel JAVA, JavaScript, Groovy und noch weitere Programmiersprachen, die hier nicht explizit erwähnt werden.

Wie wird ein Swagger in C# eingebunden

In C# kann man Swagger durch ein NuggetPackage Namens Swashbuckle.AspNetCore verwenden. Durch dieses NuggetPackage kann ein Swagger nun in eine WebApi, wie in nachfolgender Abbildung implementiert werden:



Abbildung 6: DotNet6ConApp

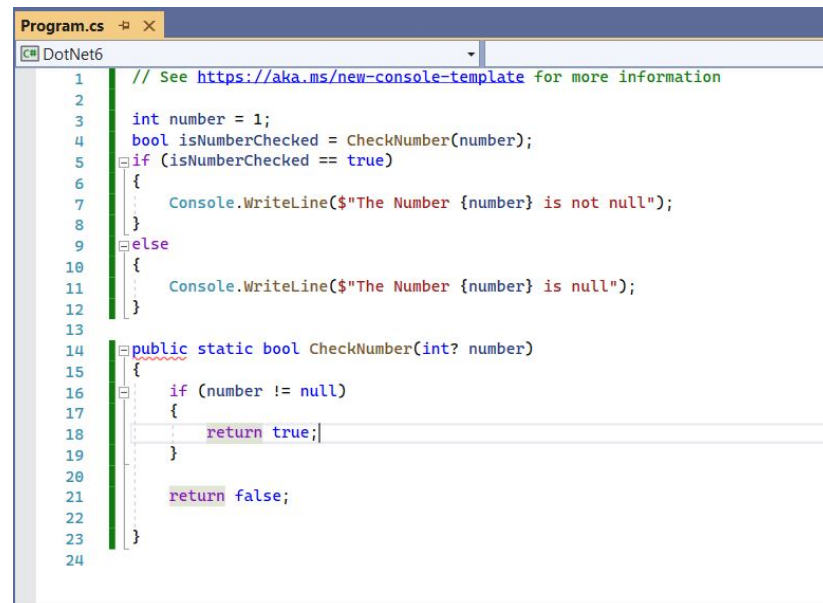
Die im Projekt verwendete Implementation wird im Unterkapitel Verwendung von Swagger im Projekt dargestellt.

3.3.4 DotNet 5 vs DotNet 6

Die Entwicklung der Diplomarbeit startete schon mit Sommerbeginn 2021 und .Net6 wurde erst im November 2021 released. Aufgrund der schon vorhandenen Kenntnisse in .Net5 und der im .Net6 bei Standardkonfiguration nicht vorhandenen Methoden / Usings Unterstützung (siehe nachfolgende Abbildungen) wurde beschlossen, wähen der Umsetzung nicht die Entwicklungsumgebung zu ändern.

Eine Migration von .Net5 auf .Net6 umfasst in der Regel dann das Umstellen aller Paketversionen auf die neueste .NetVersion. Hier sollten im Bedarfsfall dann nur wenige Änderungen notwendig werden und sollten dann auch aufgrund des längeren Supports von .Net6 und der Vorteile rund um Performance, die bei eventueller Ausführung in der Cloud sicherlich auch positive Auswirkungen auf die Bedarfskosten mit sich bringen kann, in Erwägung gezogen werden.

Persönlich empfundener Nachteil bei der Lesbarkeit des Codes:



```
1 // See https://aka.ms/new-console-template for more information
2
3 int number = 1;
4 bool isNumberChecked = CheckNumber(number);
5 if (isNumberChecked == true)
6 {
7     Console.WriteLine($"The Number {number} is not null");
8 }
9 else
10 {
11     Console.WriteLine($"The Number {number} is null");
12 }
13
14 public static bool CheckNumber(int? number)
15 {
16     if (number != null)
17     {
18         return true;
19     }
20
21     return false;
22 }
23
24
```

Abbildung 7: DotNet6ConApp

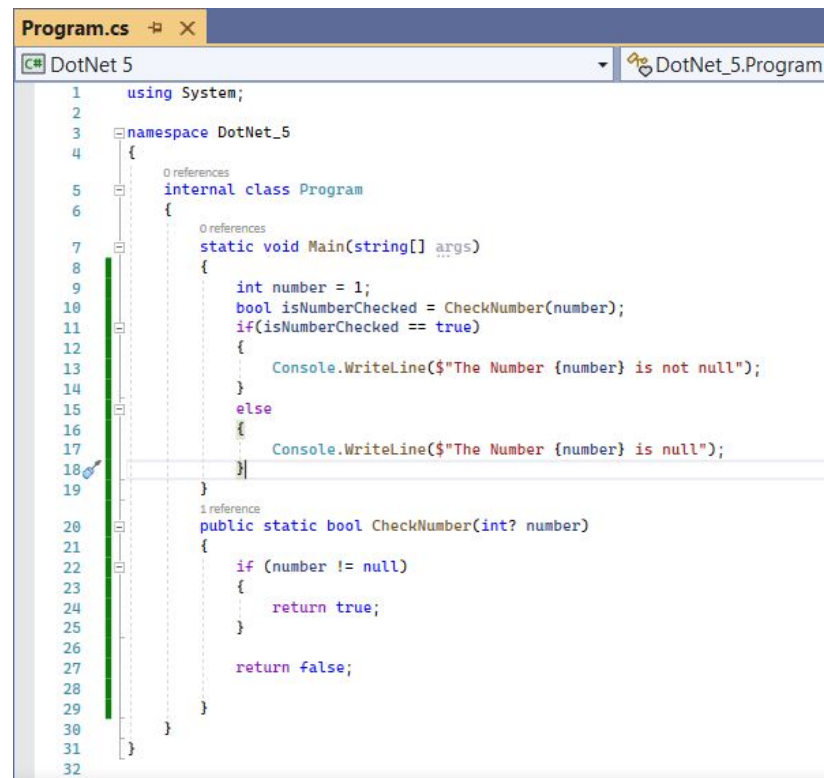


Abbildung 8: DotNet5ConApp

Im Vergleich zu .Net6 ist die Unterstützung bei der Verwendung von Usings und Methoden unter .Net5 klar zu erkennen.

Was ist eine Datenbank und wofür eignet sich eine Datenbank

Eine Datenbank ist eine Sammlung von strukturierten Informationen oder Daten, die typischerweise elektronisch in einem Computersystem gespeichert sind. Solch eine Datenbank wird meistens von einem Datenbankverwaltungssystem abgekürzt DBMS gesteuert und sie besteht aus Tabellen, in welchen die Daten gespeichert sind. Auf diese gespeicherten Daten kann mittels SQL-Abfragen zugegriffen werden.

Warum Sqlite und nicht SqlServer als Datenbank:

In diesem Projekt wird eine Sqlite-Datenbank verwendet, weil die Datenbank im Vergleich zu einem Microsoft SQL Server eine kleinere Version ist und sich sehr gut für Endgeräte eignet, da der Raspberry Pi nur über einen begrenzten Speicher verfügt. Weiters muss auch Rücksicht auf die Architektur vom ARM genommen werden, denn die Version muss für die CPU Architektur geeignet sein. Hinweis: Microsoft stellte erst mit SQL Server 2017 die erste Version auf Linux zur Verfügung, aber erst mit der Version 2019 sind die meisten Funktionen wie unter Windows verfügbar.

Überprüfung einer Sqlite Datenbank

Zum Überprüfen einer Sqlite Datenbank eignet sich in Visual Studio Code die Extension "SSQLite", welche das Innenleben einer SQLite Datenbank veranschaulicht.

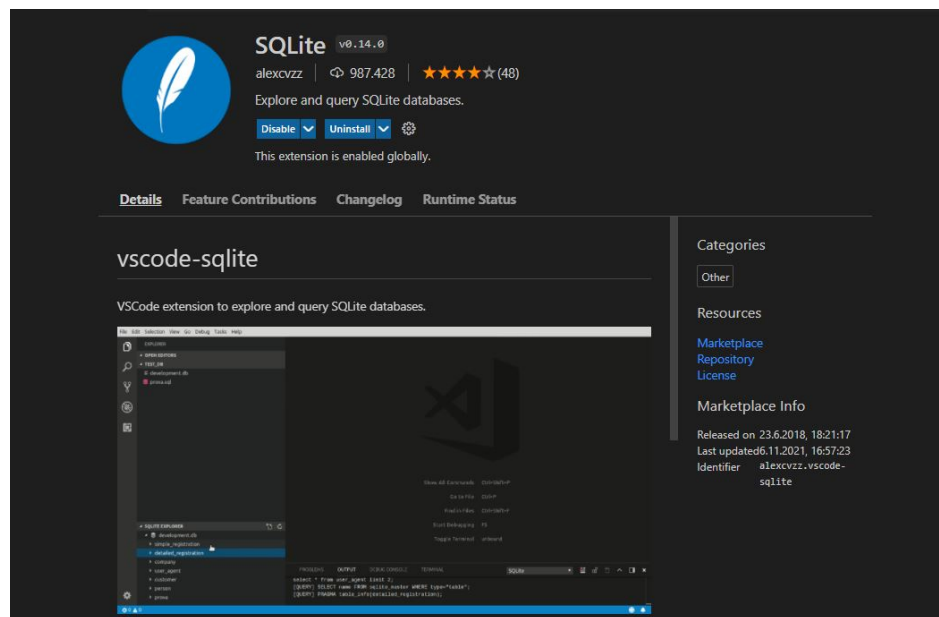


Abbildung 9: SQLite Extension für Visual Studio Code

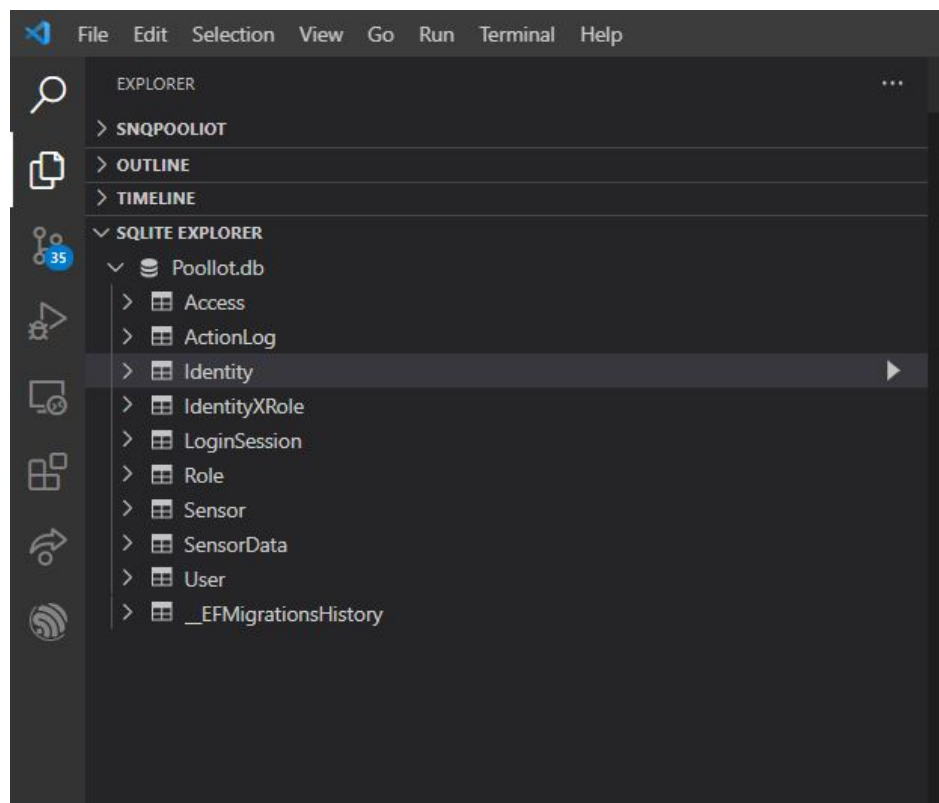


Abbildung 10: Innenleben der SQLite Datenbank

Asynchrone Programmierung und Implementierung in C#

Es sollte wenn möglich immer asynchron programmiert werden, weil es Zeit spart, denn Prozesse können dadurch parallel ausgeführt werden. Somit muss ein Prozess nicht mehr auf einen anderen Prozess warten. Ein kleines Beispiel hierfür wäre ein Frühstück, denn eine Person muss eben nicht warten bis das Toastbrot fertig ist, um sich danach erst den Kaffee zu machen, dieses Tun sollte parallel möglich sein.

In C# gibt es die Keywords `async` und `await`. Wenn eine Methode asynchron aufgerufen werden soll, muss die Methode im Methodenkopf als Kennung `async` aufweisen und einen `Task` als return Wert festlegen. Um diese Methode danach aufrufen zu können, muss `await` vor dem Methodennamen verwendet werden.

Keyword `partial` C#

In C# gibt es das sogenannte Keyword `partial`, wodurch die Implementierung von einer Methode in einer Klasse der selben Klasse jedoch in einem anderen File passieren kann. Ein Code Beispiel im Projekt wäre die `SnQPoolIot.ConApp`.

In den nachstehenden 2 Abbildungen wird beschrieben wie eine Implementierung von einer `partial` Method erfolgt.

Im ersten Foto ist zu erkennen, dass die Klasse `Program.cs` `partial` gesetzt wurde und die Methode `BeforeRun()` auch das Keyword `partial` beinhaltet.

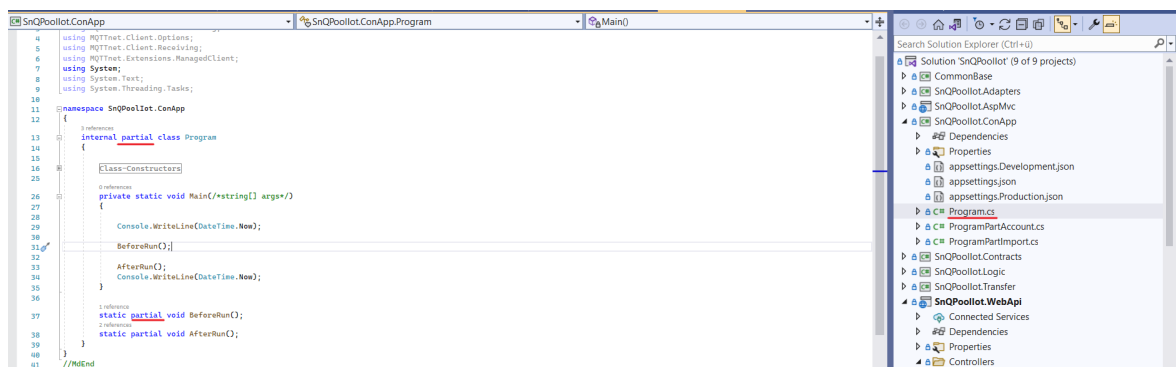


Abbildung 11: Implementierung `partial` Class und Method

Für die Implementierung in einem anderen File wird der Name des Files auf einen anderen Namen umbenannt und danach wird die Klasse wieder auf `Program.cs` umgeschrieben. Nun erkennt C# dass es sich um eine `partial` Class handelt und somit können nun die Methoden, welche in der Klasse `partial` sind, aber in einem anderen File liegen, umgeschrieben werden.

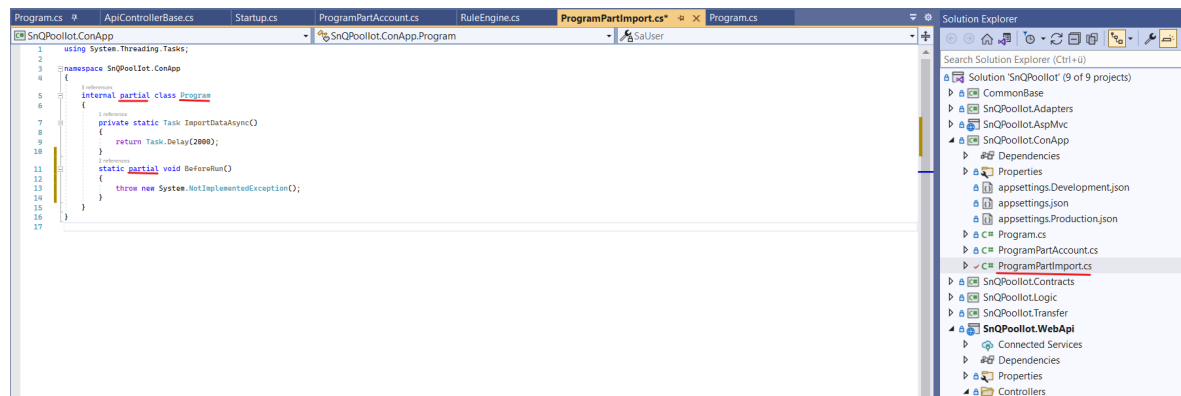


Abbildung 12: Implementierung partial Class und Method

3.4 Telegram

Telegram ist ein Messaging-Dienst wie WhatsApp, welcher auf Smartphones und Computer vorzufinden ist. In diesem Projekt wird Telegram auch zum Versenden und Empfangen von Messages verwendet, jedoch über einen sogenannten Bot. Telegram hat ein eigenes Botnetzwerk, wodurch Messages über das Internet gesendet werden können. Um einen Bot für solche Zwecke erzeugen zu können, muss bei dem sogenannten BotFather ein Bot erzeugt werden.

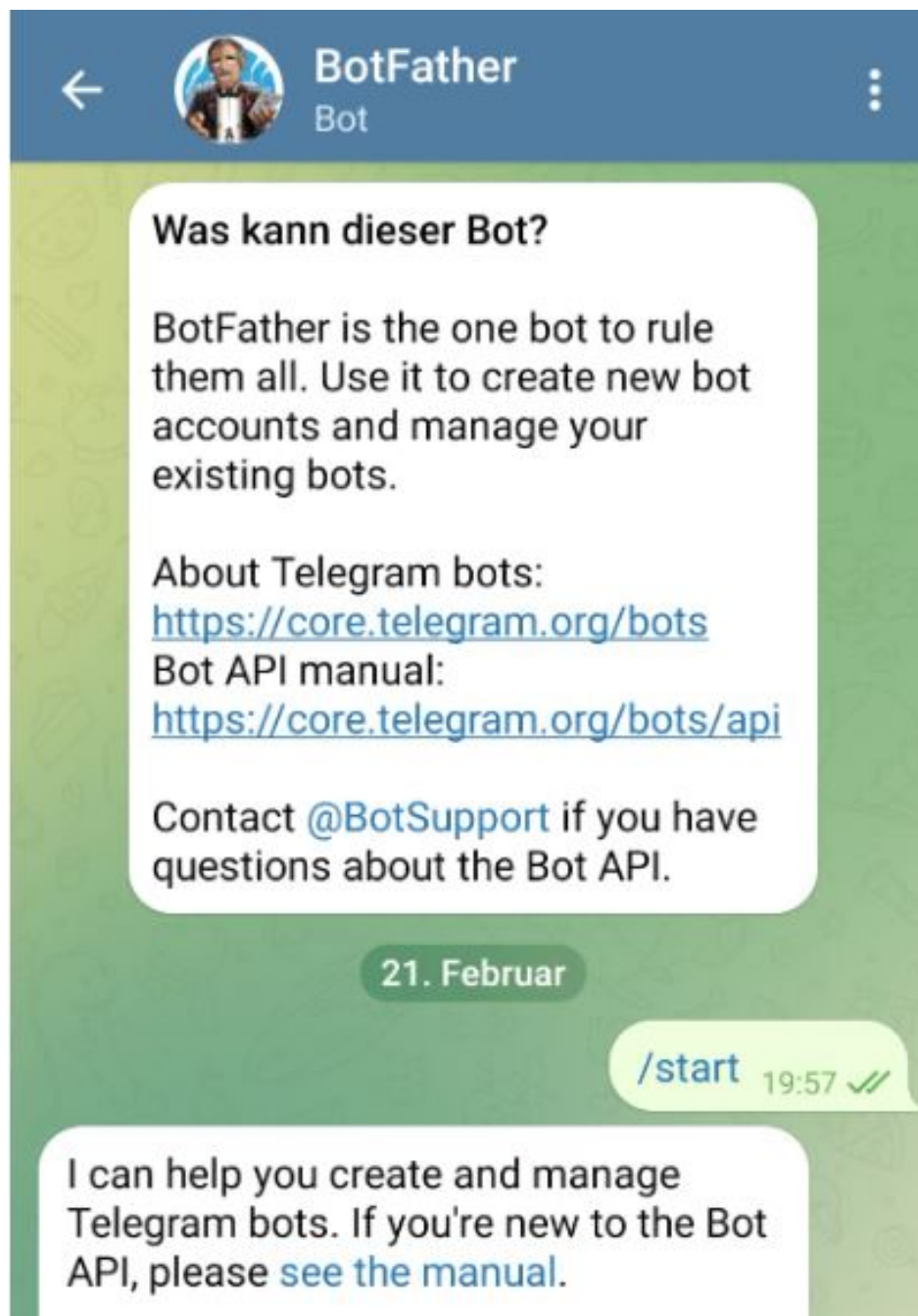


Abbildung 13: Erzeugen eines Bots über den BotFather

Sobald der BotFather gestartet wurde, kann über den Befehl `/newbot` ein neuer Bot erzeugt werden, indem eine Person dem Bot einen Namen und einen nicht existenten Usernamen, der mit `bot` endet, zuweist. Wenn dieser Vorgang durchgeführt wurde, kann auf den Bot über HTTP Requests zugegriffen werden.

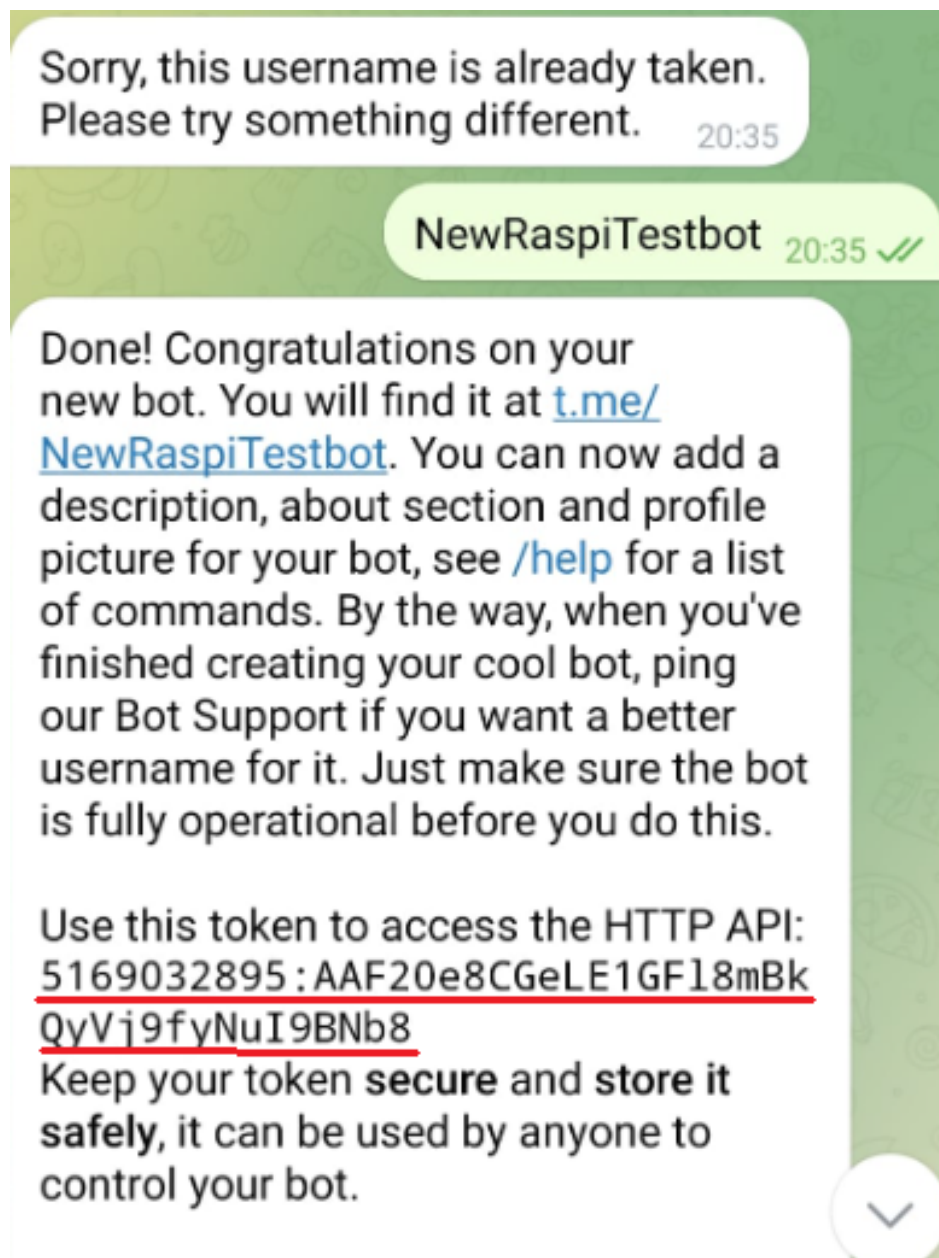


Abbildung 14: Beispiel eines Tokens der zum Zugriff auf die API benötigt wird.

Durch diesen Token kann über das Internet Messages gesendet werden oder auch die ChatId, welche eine große Rolle im Backend des Projektes spielt, herausfinden. Der Befehl um die ChatId zu erlangen lautet:

`https://api.telegram.org/bot(Token)/getMe`

danach kann eine Message über den Befehl

`https://api.telegram.org/bot(Token)/sendMessage?chat_id=(chatId)&text=Textdddddd`
gesendet werden.

3.5 Begriffserklärungen

3.5.1 C#

C# ist eine objektorientierte Programmiersprache, welche von Microsoft entwickelt wurde

3.5.2 ARM

ARM stand für Acorn RISC Machines, später für Advanced RISC Machines und ist einer der meistverbreiteten Mikroprozessoren.

4 Projektumsetzung

4.1 Projektmanagement

Das Projekt ist im Softwarebereich mittels Scrum und im Hardwarebereich mittels einer hybriden Projektentwicklung entwickelt worden.

4.1.1 Scrum

Scrum ist ein agiles Vorgehensmodell für Projektmanagement und kommt zum Einsatz wenn viele Entwicklungsprojekte zu komplex sind, um sie in einen vollumfassenden Plan zu fassen. Aus diesem Grund ist ein wesentlicher Teil der Anforderungen und Lösungen zu Beginn unklar. Diese werden schrittweise erarbeitet, indem Zwischenergebnisse geschaffen werden. Durch Scrum wird die Projektlaufzeit in Etappen, so genannte Sprints, eingeteilt und am Ende jedes Sprints soll jeweils ein funktionsfähiges Zwischenprodukt stehen. Die Einteilung wie lange ein Sprint dauert wird intern geregelt. In diesem Projekt war die Dauer eines Sprints einen Monat lang.

4.1.2 Hybride Projektentwicklung

Hybride Projektentwicklung wird meist im Hardwarebereich verwendet. Es ermöglicht eine schnellere Produktion von Hardwareteilen. In diesem Projekt ist es bei dem Zusammenbau für den Rasperray verwendet und für die Produktion einer Schwimmboje vorgesehen worden.

4.1.3 Zusätzliche Vereinbarungen der Diplomarbeit

Zusätzlich zu den Projektentwicklungskonzepten wurde jede zweite Woche ein Projektbericht veröffentlicht, um die Diplomarbeitsbetreuer auf dem Laufenden zu halten.












	20210507_Besprechungsprotokoll1.pdf	07.05.2021 08:54	Microsoft Edge PD...	75 KB
	20210519_Besprechungsprotokoll2.pdf	20.05.2021 00:57	Microsoft Edge PD...	74 KB
	20210527_Besprechungsprotokoll3.pdf	31.05.2021 20:21	Microsoft Edge PD...	108 KB
	20210616_Besprechungsprotokoll4.pdf	16.06.2021 14:07	Microsoft Edge PD...	115 KB
	Aufgabenstellung MQTT mit Raspi.pdf	05.01.2022 16:01	Microsoft Edge PD...	159 KB
	Bestätigung.pdf	29.06.2021 10:15	Microsoft Edge PD...	72 KB
	Meilensteinbericht 6.3.2022.pdf	06.03.2022 18:32	Microsoft Edge PD...	52 KB
	Projektbericht 28.10.2021.pdf	28.10.2021 15:01	Microsoft Edge PD...	66 KB
	Projektbericht_11.11.2021.pdf	05.12.2021 19:16	Microsoft Edge PD...	119 KB
	Projektbesprechung 29.10.2021.pdf	30.10.2021 10:34	Microsoft Edge PD...	46 KB
	Raspberry Configuring.pdf	28.06.2021 20:21	Microsoft Edge PD...	194 KB

Abbildung 15: Ausschnitt aus den Projektberichten

In den Projektberichten sind die fertigen Meilensteine und die Meilensteine an denen gearbeitet wurde, mittels einer Projektampel veranschaulicht. Eine Projektampel sagt aus, ob sich ein Projekt in Gefahr, in Verzug oder im grünen Bereich befindet.

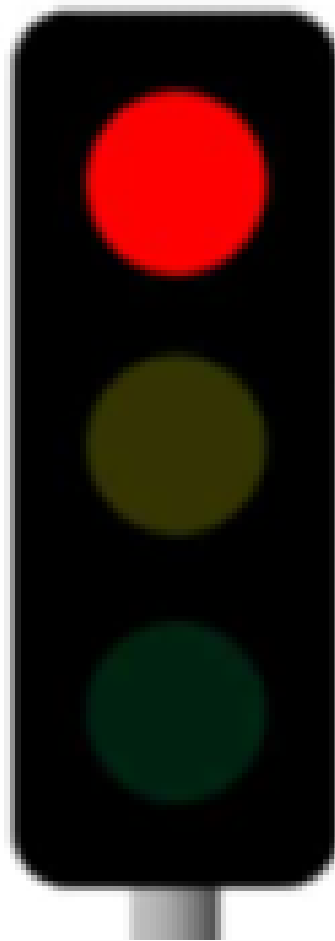


Abbildung 16: Visualisierung einer Projektampel

4.2 Backend Projekt-Überblick

Das Backend setzt sich aus 9 Projekten zusammen, welches mithilfe des von Herrn DI Professor Gerhard Gehrler zur Verfügung gestellten Frameworkes SmartNQuick entwickelt wurde. Das Backend wurde in der Sprache C# in .Net 5 programmiert und als Code Editor wurde Visual Studio 2019 und 2022 verwendet.

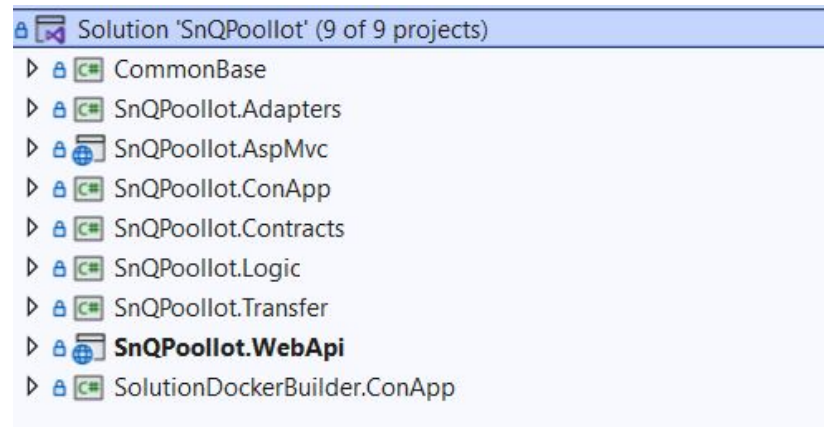


Abbildung 17: Projektmappe

- **CommonBase**

In CommonBase befinden sich Klassen und Methoden, die wiederverwendbar sind, um Codeverdoppelung zu vermeiden.

- **SnQPoolot.Adapters**

SnQPoolot.Adapters bietet einen direkten Zugriff auf die Logic. Der Zugriff auf die Logic kann dadurch entweder direkt erfolgen oder per Rest über die WebApi.

- **SnQPoolot.WebApi**

Der Zugriff auf die Messwerte wird durch Rest-Zugriffe in SnQPoolot.WebApi provided. Auf die Daten kann aber nur per Login mit einem gültigen Account zugegriffen werden. Genauer zu den einzelnen HTTP-Requests ist im Kapitel HTTP und Verwendung in unserem Backend zu finden.

- **SnQPoolot.Contracts**

SnQPoolot.Contracts beinhaltet alle notwendigen Schnittstellen und Enumerationen des Projektes. Hier werden die Entitäten als Interfaces angelegt.

- **SnQPoolot.Logic**

SnQPoolot.Logic ist das Kernstück des Projektes. Durch die Logic können alle Daten aus der Datenbank verwendet werden. Die Logic verbindet sich mit einer

Sqlite Datenbank. Der Zugriff und das Erzeugen der Datenbank wird mittels Entityframework.Sqlite durchgeführt.

- SnQPoolIot.Transfer SnQPoolIot.Transfer verwaltet die Transferobjekte für den Datenaustausch zwischen den Layern.
- SnQPoolIot.AspMvc SnQPoolIot.AspMvc ist ein Ersatz für das Frontend. Hier werden die Funktionen z.B.: das Einloggen eines Users oder Anzeigen von Messwerten dargestellt.
- SnQPoolIot.ConApp In SnQPoolIot.ConApp werden User mit verschiedenen Rechten angelegt, die für die Authentifizierung benötigt werden.

4.2.1 SnQPoolIot.Logic

Wie bereits im Backend Projekt-Überblick beschrieben, befindet sich in SnQPoolIot.Logic die Datenbank mit den Zugriffen. Die Datenbank wird mithilfe des Nugget-Package Microsoft.EntityFrameworkCore.Sqlite, den Befehlen: `dotnet ef migrations add InitDb` und `dotnet ef database update`, welche in der Developer-PowerShell im Visual Studio ausgeführt werden müssen um Migrations zu erzeugen und um die Datenbank mit den erzeugten Migrations upzudaten, und einem DbContext, welcher die Configuration der Datenbank mit sich bringt, automatisch erstellt.

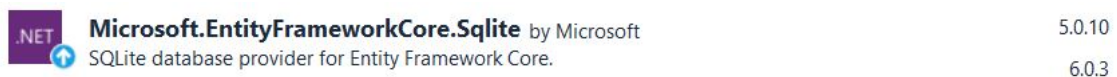


Abbildung 18: NuggetPackage für Entityframework mit Sqlite

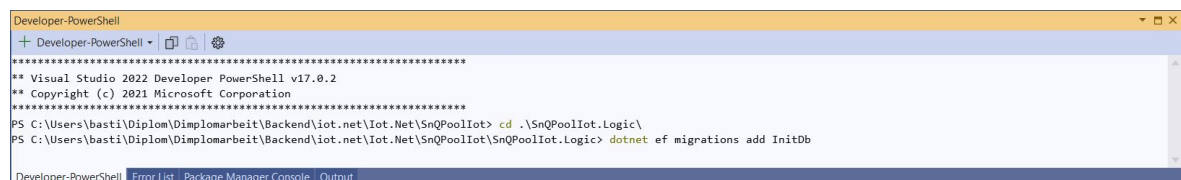


Abbildung 19: Befehl zum Erzeugen von Migrationen

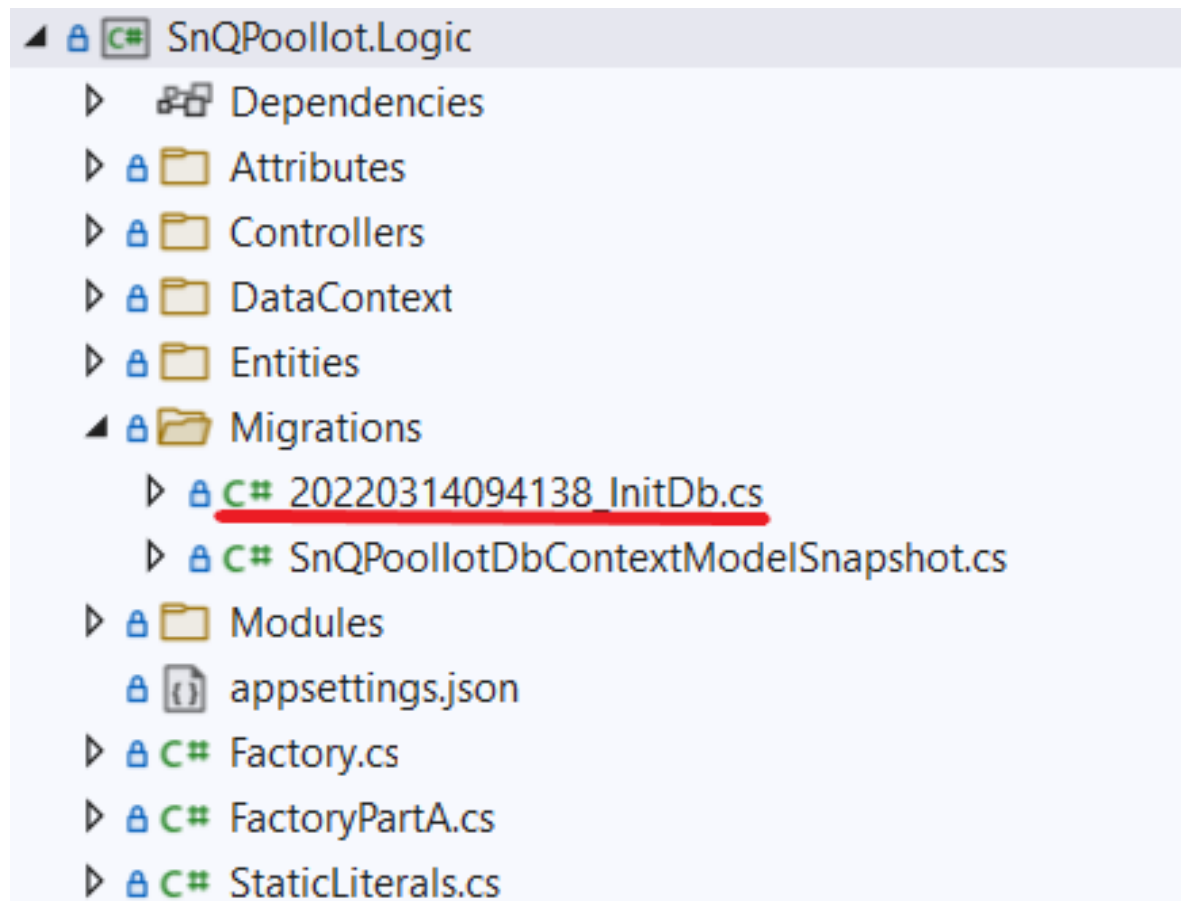


Abbildung 20: Erzeugte Migrationen in der Logik

```

using Microsoft.EntityFrameworkCore.Design;
using Microsoft.Extensions.Configuration;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SnQPoolIot.Logic.DataContext
{
    0 references
    class BloggingContextFactory : IDesignTimeDbContextFactory<SnQPoolIotDbContext>
    {
        0 references
        public SnQPoolIotDbContext CreateDbContext(string[] args = null)
        {
            var configuration = new ConfigurationBuilder().AddJsonFile("appsettings.json").Build();

            var optionsBuilder = new DbContextOptionsBuilder<SnQPoolIotDbContext>();
            optionsBuilder
                .UseSqlite(configuration["ConnectionStrings:DefaultConnection"]);

            return new SnQPoolIotDbContext();
        }
    }
}

```

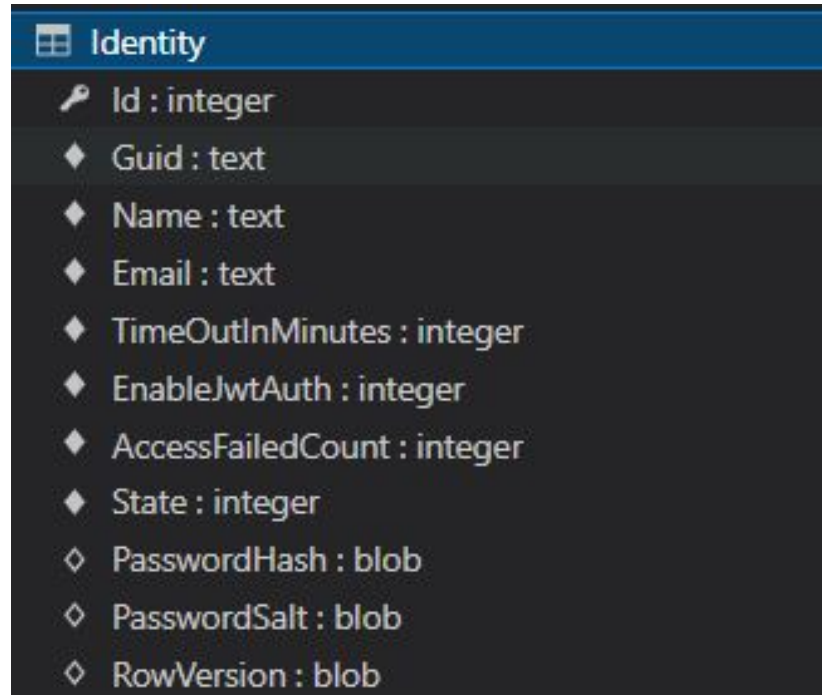
Abbildung 21: DbContext zum Erzeugen einer Datenbank

Nun zu den Datenbank Zugriffen. Um auf die Messwerte zugreifen zu können, muss sich ein User zuerst authentifizieren. Damit sich eine User authentifizieren kann benötigt

er einen Account mit E-Mail und Passwort, welche in der Datenbank gespeichert sind. Für die Authentifizierung werden die nachstehenden Tabellen benötigt:

- Identity

In dieser Tabelle befinden sich alle User-Accounts mit E-Mail gehastem Passwort.

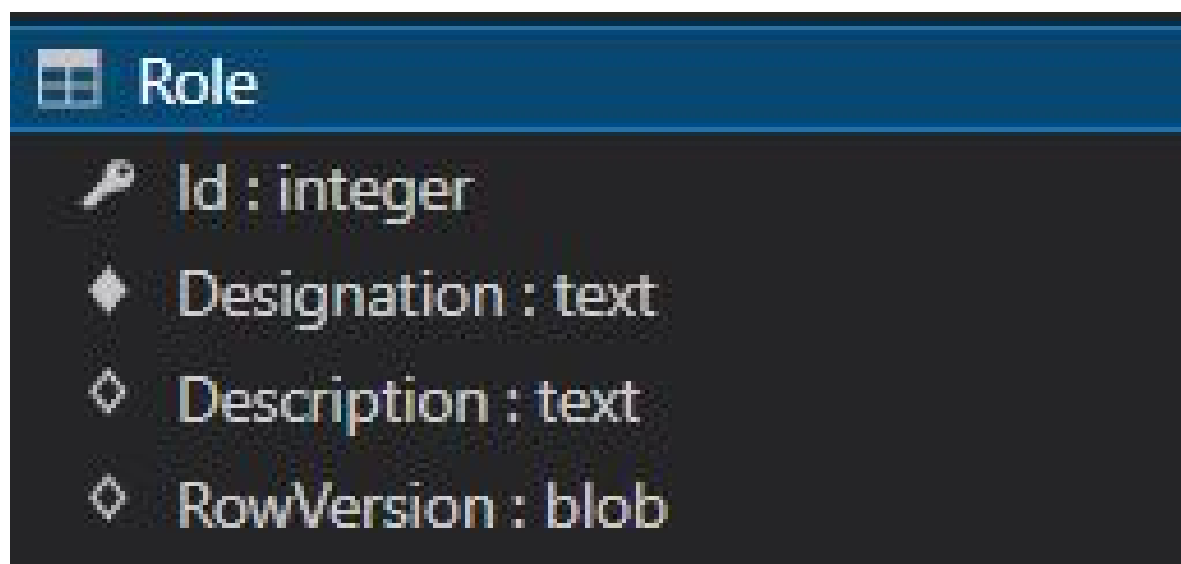


Identity	
Id : integer	
Guid : text	
Name : text	
Email : text	
TimeOutInMinutes : integer	
EnableJwtAuth : integer	
AccessFailedCount : integer	
State : integer	
PasswordHash : blob	
PasswordSalt : blob	
RowVersion : blob	

Abbildung 22: Identity Tabelle mit den dazugehörigen Tabellenspalten

- Role

In dieser Tabelle sind alle Rollen, die es gibt, vorzufinden.

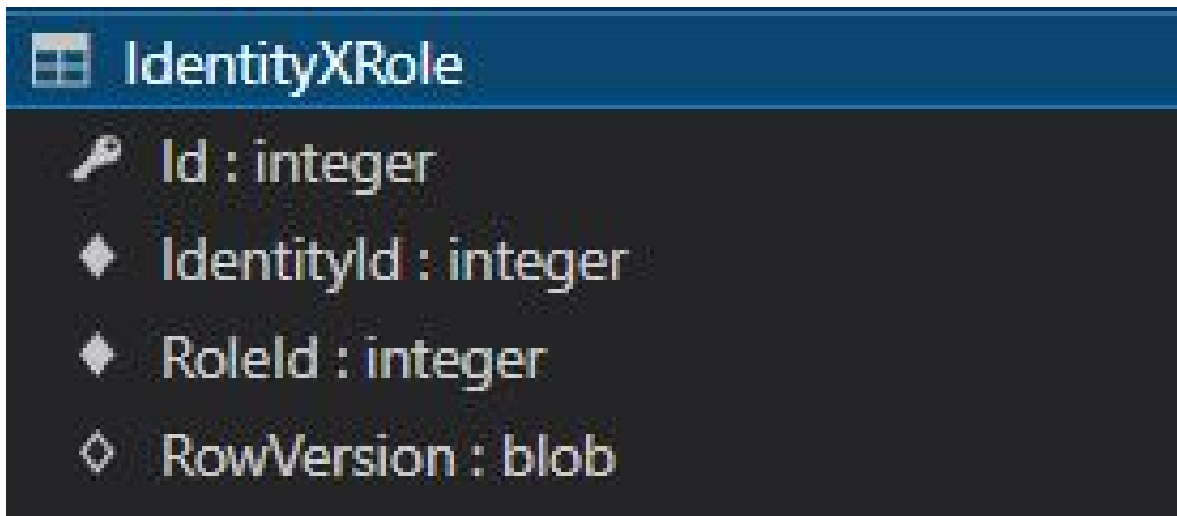


Role	
Id : integer	
Designation : text	
Description : text	
RowVersion : blob	

Abbildung 23: Role Tabelle mit den dazugehörigen Tabellenspalten

- IdentityXRole

Diese Tabelle weist einem Benutzer eine Rolle zu.



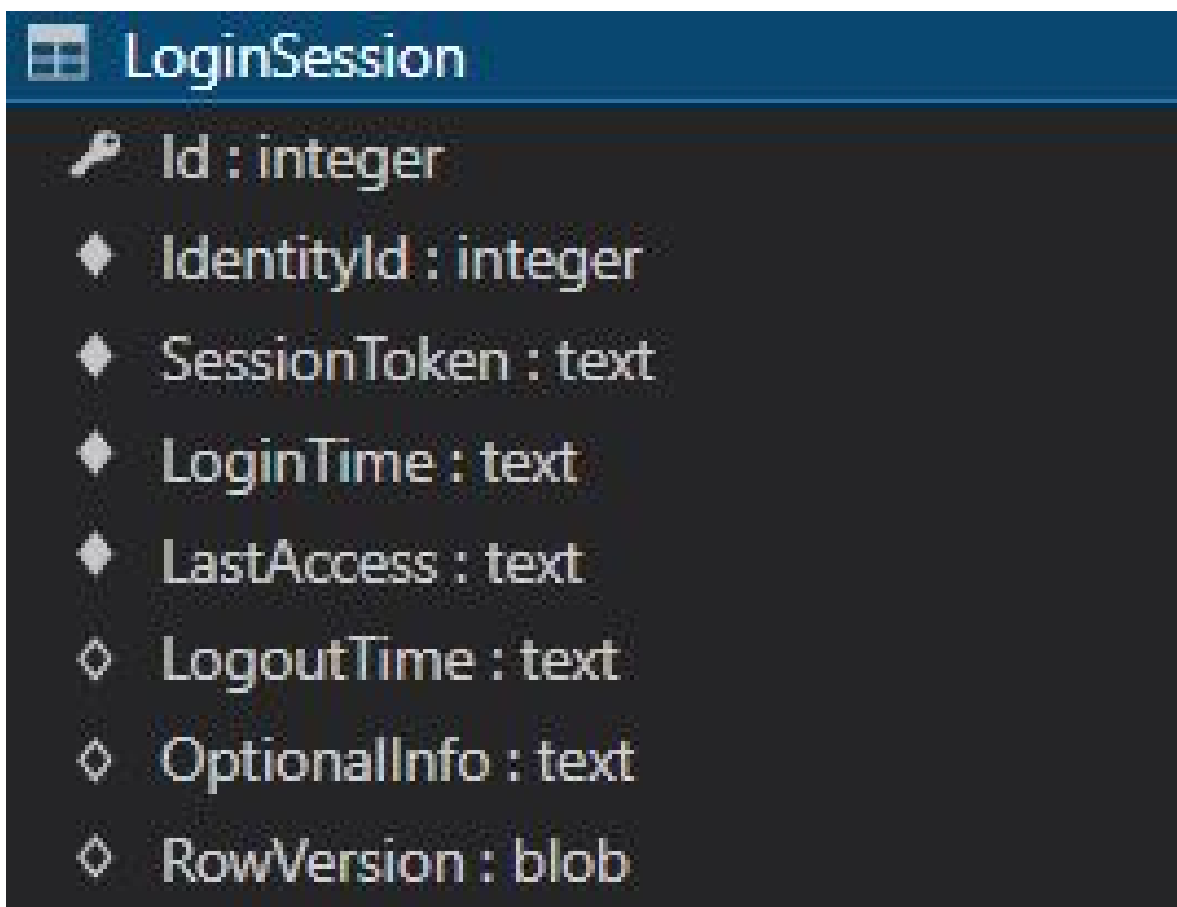
The diagram shows the structure of the IdentityXRole table. It has a dark blue header with a table icon and the text 'IdentityXRole'. Below the header, on a dark background, are four entries: 'Id : integer' with a key icon, 'IdentityId : integer' with a diamond icon, 'RoleId : integer' with a diamond icon, and 'RowVersion : blob' with a diamond icon.

IdentityXRole	
Id : integer	
IdentityId : integer	
RoleId : integer	
RowVersion : blob	

Abbildung 24: IdentityXRole Tabelle mit den dazugehörigen Tabellenspalten

- LoginSession

Diese Tabelle zeigt alle Logins mit dem dazugehörigen User und dem SessionToken, mit dem sich der User eingeloggt hat, an.



The diagram shows the structure of the LoginSession table. It has a dark blue header with a table icon and the text 'LoginSession'. Below the header, on a dark background, are eight entries: 'Id : integer' with a key icon, 'IdentityId : integer' with a diamond icon, 'SessionToken : text' with a diamond icon, 'LoginTime : text' with a diamond icon, 'LastAccess : text' with a diamond icon, 'LogoutTime : text' with a diamond icon, 'OptionalInfo : text' with a diamond icon, and 'RowVersion : blob' with a diamond icon.

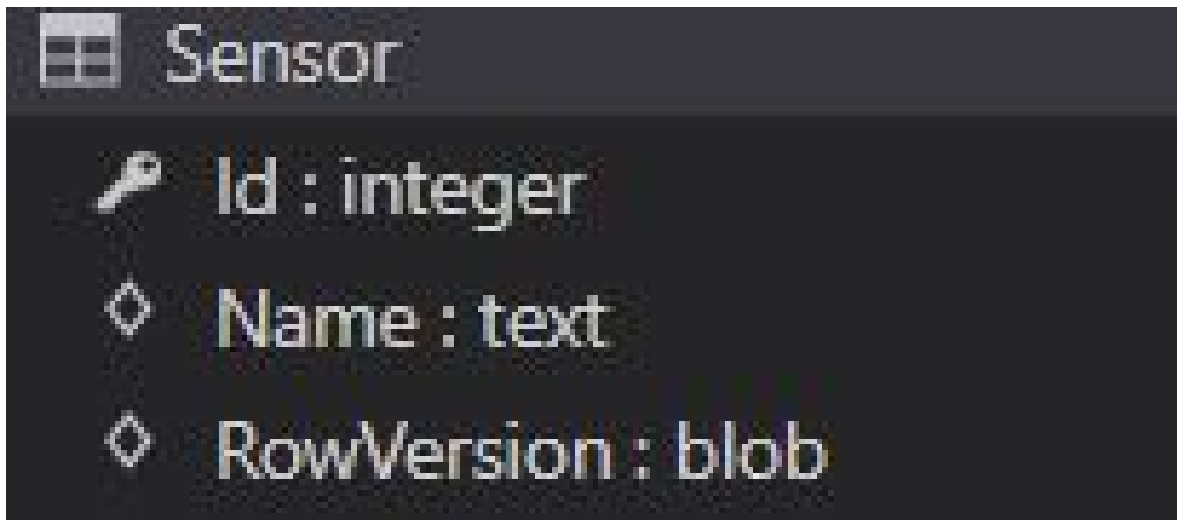
LoginSession	
Id : integer	
IdentityId : integer	
SessionToken : text	
LoginTime : text	
LastAccess : text	
LogoutTime : text	
OptionalInfo : text	
RowVersion : blob	

Abbildung 25: LoginSession Tabelle mit den dazugehörigen Tabellenspalten

Sobald sich ein User authentifiziert hat, wurde zugleich eine neue Session erstellt. Nur bei ausreichender Berechtigung des authentifizierten Benutzers kann dieser sich nun die Messwerte ansehen. Die nachstehenden Tabellen dienen zum Erfassen der Messwerte:

- Sensor

Diese Tabelle beinhaltet den Namen eines Sensors.

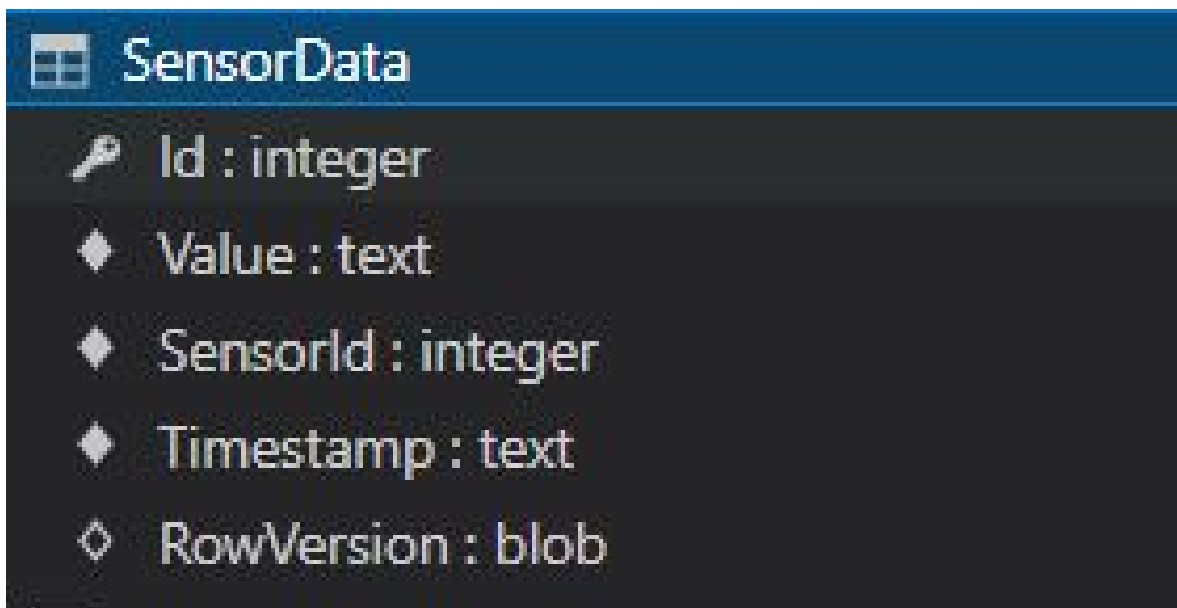


Sensor	
Id : integer	
Name : text	
RowVersion : blob	

Abbildung 26: Sensor Tabelle mit den dazugehörigen Tabellenspalten

- SensorData

Diese Tabelle beinhaltet die Messwerte aller Sensoren und jeder Messwert ist einem Sensor zugeordnet.



SensorData	
Id : integer	
Value : text	
SensorId : integer	
Timestamp : text	
RowVersion : blob	

Abbildung 27: SensorData Tabelle mit den dazugehörigen Tabellenspalten

Logging in unserem Projekt

Um Fehlermeldungen oder Warnungen im Projekt zu speichern, wurde eine Klasse mit dem Namen `LogWriter` programmiert. Diese Meldungen werden in einer `.txt` gespeichert und können jederzeit gelesen werden.

Im nachstehenden Code ist die `LogWriter` Klasse zu sehen:

```

1  using System;
2  using System.Collections.Generic;
3  using System.IO;
4  using System.Linq;
5  using System.Reflection;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace SnQPoolIot.Logic.Entities.Business.Logging
10 {
11     public class LogWriter
12     {
13         private string m_exePath = string.Empty;
14         private static LogWriter _instance = null;
15
16         public static LogWriter Instance
17         {
18             get
19             {
20                 if(_instance == null)
21                 {
22                     _instance = new LogWriter();
23                 }
24                 return _instance;
25             }
26         }
27
28         private LogWriter()
29         {
30         }
31
32         public void LogWrite(string logMessage)
33         {
34             m_exePath =
35                 Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
36             try
37             {
38                 using StreamWriter w = File.AppendText(m_exePath + "\\\" +
39                     "log.txt");
40                 Log(logMessage, w);
41             }
42             catch (Exception)
43             {
44             }
45         }
46         public static void Log(string logMessage, TextWriter txtWriter)
47         {
48             try
49             {
50                 txtWriter.Write("\r\nLog Entry : ");
51                 txtWriter.WriteLine("{0} {1}", DateTime.Now.ToLongTimeString(),
52                     DateTime.Now.ToLongDateString());
53                 txtWriter.WriteLine("   :{0}", logMessage);
54                 txtWriter.WriteLine("-----");
55             }
56             catch (Exception )
57             {
58             }
59         }
60     }
61 }
```

Der Methode `LogWrite` wird durch einen Übergabeparameter die auszugebende Fehlermeldungen übergeben. Nach Ermittlung der Datei wird durch Aufruf der Methode `Log`

die Fehlermeldung inklusive Zeitstempel in der Datei gespeichert.

4.2.2 SnQPoolIot.WebApi

HTTP und Verwendung im Backend

HTTP ausgeschrieben Hypertext Transfer Protocoll wird zum Laden von Webseiten im Projekt verwendet. Die verwendeten HTTP-Requests und das dazugehörige Routing ist im Projekt SnQPoolIot.WebApi implementiert.

Im Folgenden ist ein Ausschnitt des Aufbaus der WebApi dargestellt:

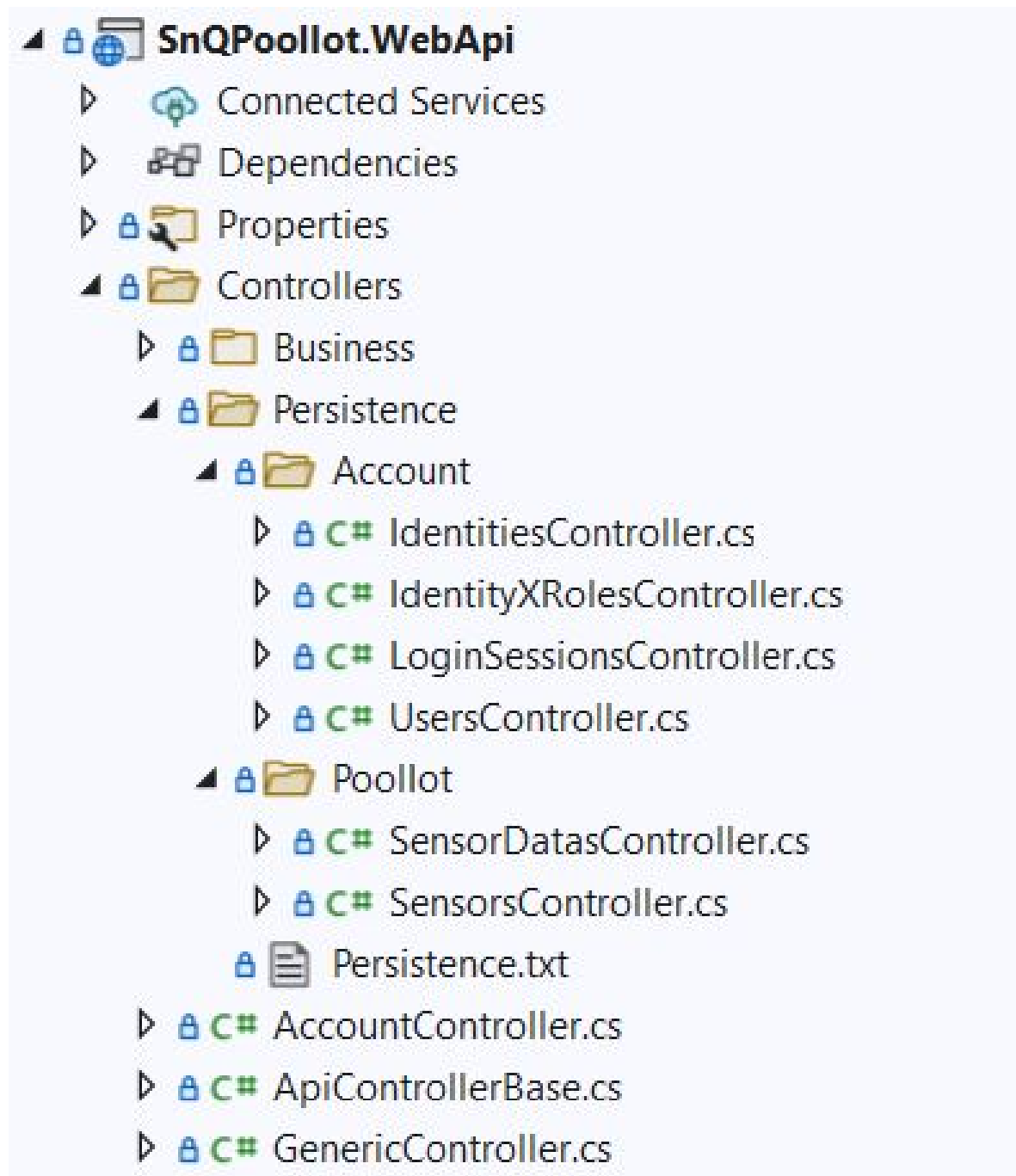


Abbildung 28: Aufbau der WebApi

Das Routing der Websites wird über die Controller gehandelt. Der meist genützte Controller in diesem Projekt ist der `GenericController`, denn er bietet allen abgeleiteten Controllern seine bereitgestellten Funktionen zur Verwendung an.


```

/// <summary>
/// Dieser Request übermittelt uns alle Werte der dazugehörigen Tabelle
/// </summary>
/// <returns></returns>
[HttpGet("/api/[controller]")]
public async Task<IEnumerable<M>> GetAllAsync()
{
    using var ctrl = await CreateControllerAsync().ConfigureAwait(false);
    var result = await ctrl.GetAllAsync().ConfigureAwait(false);

    return result.Select(e => ToModel(e));
}
/// <summary>
/// Dieser Request übermittelt uns die Tabellenspalten der dazugehörigen Tabelle, welche das Suchkriterium erfüllen
/// </summary>
/// <param name="predicate"></param>
/// <returns></returns>
[HttpGet("/api/[controller]/Query/{predicate}")]
public async Task<IEnumerable<M>> QueryAllBy(string predicate)
{
    using var ctrl = await CreateControllerAsync().ConfigureAwait(false);
    var result = await ctrl.QueryAllAsync(predicate).ConfigureAwait(false);

    return result.Select(e => ToModel(e));
}
/// <summary>
/// Dieser Request erzeugt einen neuen Datenbankeintrag in die Tabelle.
/// </summary>
/// <param name="model"></param>
/// <returns></returns>
[HttpPost("/api/[controller]")]
public async Task<M> PostAsync([FromBody] M model)
{
    using var ctrl = await CreateControllerAsync().ConfigureAwait(false);
    var result = await ctrl.InsertAsync(model).ConfigureAwait(false);

    await ctrl.SaveChangesAsync().ConfigureAwait(false);
    return ToModel(result);
}

```

Abbildung 29: Auszug GenericController der WebApi

Wie am Beispiel des Codeauszuges zu sehen, beinhaltet der Generic Controller einige Methoden.

Für die Ermittlung der Daten wird zum Beispiel eine Get-Methode "GetAllAsync" eingesetzt.

Die Codezeile `[HttpGet("/api/[controller]")]` drückt aus, dass es sich bei dieser Methode um eine Get-Methode handelt und veranschaulicht auch das Routing der WebApi.

All jene Methoden, die im GenericController definiert sind, stehen demnach allen von ihm abgeleiteten Controllern zur Verfügung und können direkt verwendet werden.

Im nachstehenden Beispiel wird eine Ableitung vom GenericController dargestellt.

```

namespace SnQPoolIot.WebApi.Controllers.Persistence.PoolIot
{
    using Microsoft.AspNetCore.Mvc;
    using TContract = Contracts.Persistence.PoolIot.ISensor;
    using TModel = Transfer.Models.Persistence.PoolIot.Sensor;
    [ApiController]
    [Route("Controller")]
    public partial class SensorsController : WebApi.Controllers.GenericController<TContract, TModel>
    {
    }
}

```

Abbildung 30: Anwendung des GenericControllers der WebApi

Verwendung von Swagger im Projekt

Zum Allgemeinen Überblick aller HTTP-Requests wird Swagger verwendet.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();
    services.AddSwaggerGen(options =>
    {
        options.SwaggerDoc("v1", new OpenApiInfo
        {
            Version = "v1",
            Title = "SnQPoolIot.WebApi",
            Description = "Api zum einlesen und auslesen von Sensoren und deren Messwerte",
        });
        options.AddSecurityDefinition("SessionToken", new OpenApiSecurityScheme
        {
            In = ParameterLocation.Header,
            Description = "Bitte geben Sie SessionToken und den Wert des gültigen SessionToken in das Feld",
            Name = "Authorization",
            Type = SecuritySchemeType.ApiKey
        });
        var xmlFilename = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
        options.IncludeXmlComments(Path.Combine(AppContext.BaseDirectory, xmlFilename));
        options.AddSecurityRequirement(new OpenApiSecurityRequirement {
            {
                new OpenApiSecurityScheme
                {
                    Reference = new OpenApiReference
                    {
                        Type = ReferenceType.SecurityScheme,
                        Id = "SessionToken"
                    }
                },
                Array.Empty<string>()
            }
        });
    });
}
```

Abbildung 31: Implementierung von Swagger in der WebApi

In den nachstehenden Zeilen findet sich eine kurze Ablaufbeschreibung des Codes: Sobald die Methode ConfigureServices aufgerufen wird, wird dem service ein neuer Controller angelegt und die benötigte Konfiguration des Swaggers mit übergeben.

Das options.SwaggerDoc gibt eine kurze Beschreibung über die WebApi an. Im darauf folgenden Schritt wird die Authentifizierung im Projekt mittels Swagger durchgeführt. Dadurch wird gewährleistet, dass nur eingeloggte Benutzer die Möglichkeit haben Zugriffe auf HTTP-Requests durchzuführen. Sobald die Methode fertig ausgeführt wurde startet im Browser eine Website mit allen HTTP-Requests die im Projekt implementiert sind.

Unterstützte HTTP-Requests des Projektes

Mit Hilfe von Swagger wurden die in den nachstehenden Grafiken ersichtlichen HTTP-Requests des Backends dokumentiert:

Account		
POST	/api/Account/Logon	Dieser Request dient zum Einloggen mittels Session Token. Ohne, dass sich der Benutzer vorher einloggt kann er keine Daten sehen.
POST	/api/Account/JsonWebLogon	Dieser Request dient zum Ermitteln von einer Session, wo man sehen kann welcher User wann und wo eingeloggt wurde.
GET	/api/Account/Logout/{sessionToken}	Dieser Request dient zum Ausloggen von einem User.
GET	/api/Account/ChangePassword/{sessionToken}/{oldPwd}/{newPwd}	Dieser Request dient zum Ändern von dem Passwort des gerade angemeldeten User's.
GET	/api/Account/ChangePasswordFor/{sessionToken}/{email}/{newPwd}	Dieser Request dient zum Ändern von einem Passwort eines User's.
GET	/api/Account/ResetFailedCountFor/{sessionToken}/{email}	Dieser Request zeigt an, wenn ein Reset von einem Passwort eines User's.
GET	/api/Account/HasRole/{sessionToken}/{role}	Dieser Request dient zum Kontrollieren von der Rolle eines User's.
GET	/api/Account/IsSessionAlive/{sessionToken}	Dieser Request dient zum Kontrollieren, ob die Session noch gültig ist.
GET	/api/Account/QueryRoles/{sessionToken}	Dieser Request gibt alle Rollen dieser Session zurück.
GET	/api/Account/QueryLogin/{sessionToken}	Dieser Request dient zum Ermitteln vom eingeloggten User.

Abbildung 32: HTTP-Requests des Projektes

AppAccesses		
GET	/api/AppAccesses/Count	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden.
GET	/api/AppAccesses/Count/{predicate}	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden, welche das Suchkriterium erfüllen.
GET	/api/AppAccesses/{id}	Dieser Request übermittelt den Wert welcher die dazugehörige Id hat
DELETE	/api/AppAccesses/{id}	Dieser Request löscht einen Datenbankeintrag mit der dazugehörigen Id in der Tabelle.
GET	/api/AppAccesses	Dieser Request übermittelt uns alle Werte der dazugehörigen Tabelle
POST	/api/AppAccesses	Dieser Request erzeugt einen neuen Datenbankeintrag in die Tabelle.
PUT	/api/AppAccesses	Dieser Request verändert einen Datenbankeintrag in der Tabelle.
GET	/api/AppAccesses/Query/{predicate}	Dieser Request übermittelt uns die Tabellenspalten der dazugehörigen Tabelle, welche das Suchkriterium erfüllen.
POST	/api/AppAccesses/Array	Dieser Request erzeugt mehrere neue Datenbankeinträge in die Tabelle.
PUT	/api/AppAccesses/Array	Dieser Request verändert mehrere Datenbankeinträge in der Tabelle.
GET	/api/AppAccesses/GetSessionTokenAsync	Dieser Request übermittelt den derzeitigen SessionToken von der aktiven Session.

Abbildung 33: HTTP-Requests des Projektes

Identities		
GET	/api/Identities/Count	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden.
GET	/api/Identities/Count/{predicate}	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden, wenn...
GET	/api/Identities/{id}	Dieser Request übermittelt den Wert welcher die dazugehörige Id hat
DELETE	/api/Identities/{id}	Dieser Request löscht einen Datenbankeintrag mit der dazugehörigen Id in der Tabelle.
GET	/api/Identities	Dieser Request übermittelt uns alle Werte der dazugehörigen Tabelle
POST	/api/Identities	Dieser Request erzeugt einen neuen Datenbankeintrag in die Tabelle.
PUT	/api/Identities	Dieser Request verändert einen Datenbankeintrag in der Tabelle.
GET	/api/Identities/Query/{predicate}	Dieser Request übermittelt uns die Tabellenspalten der dazugehörigen Tabelle, welche das Suchkriterium...
POST	/api/Identities/Array	Dieser Request erzeugt mehrer neue Datenbankeintrag in die Tabelle.
PUT	/api/Identities/Array	Dieser Request verändert mehrere Datenbankeintrag in der Tabelle.
GET	/api/Identities/GetSessionTokenAsync	Dieser Request übermittelt den derzeitigen SessionToken von der aktiven Session.

Abbildung 34: HTTP-Requests des Projektes

IdentityUsers		
GET	/api/IdentityUsers/Count	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden.
GET	/api/IdentityUsers/Count/{predicate}	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden, wenn...
GET	/api/IdentityUsers/{id}	Dieser Request übermittelt den Wert welcher die dazugehörige Id hat
DELETE	/api/IdentityUsers/{id}	Dieser Request löscht einen Datenbankeintrag mit der dazugehörigen Id in der Tabelle.
GET	/api/IdentityUsers	Dieser Request übermittelt uns alle Werte der dazugehörigen Tabelle
POST	/api/IdentityUsers	Dieser Request erzeugt einen neuen Datenbankeintrag in die Tabelle.
PUT	/api/IdentityUsers	Dieser Request verändert einen Datenbankeintrag in der Tabelle.
GET	/api/IdentityUsers/Query/{predicate}	Dieser Request übermittelt uns die Tabellenspalten der dazugehörigen Tabelle, welche das Suchkriterium...
POST	/api/IdentityUsers/Array	Dieser Request erzeugt mehrer neue Datenbankeintrag in die Tabelle.
PUT	/api/IdentityUsers/Array	Dieser Request verändert mehrere Datenbankeintrag in der Tabelle.
GET	/api/IdentityUsers/GetSessionTokenAsync	Dieser Request übermittelt den derzeitigen SessionToken von der aktiven Session.

Abbildung 35: HTTP-Requests des Projektes

IdentityXRoles		
GET	/api/IdentityXRoles/Count	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden.
GET	/api/IdentityXRoles/Count/{predicate}	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden.
GET	/api/IdentityXRoles/{id}	Dieser Request übermittelt den Wert welcher die dazugehörige Id hat
DELETE	/api/IdentityXRoles/{id}	Dieser Request löscht einen Datenbankeintrag mit der dazugehörigen Id in der Tabelle.
GET	/api/IdentityXRoles	Dieser Request übermittelt uns alle Werte der dazugehörigen Tabelle
POST	/api/IdentityXRoles	Dieser Request erzeugt einen neuen Datenbankeintrag in die Tabelle.
PUT	/api/IdentityXRoles	Dieser Request verändert einen Datenbankeintrag in der Tabelle.
GET	/api/IdentityXRoles/Query/{predicate}	Dieser Request übermittelt uns die Tabellenspalten der dazugehörigen Tabelle, welche das Suchkrit
POST	/api/IdentityXRoles/Array	Dieser Request erzeugt mehrer neue Datenbankeintrag in die Tabelle.
PUT	/api/IdentityXRoles/Array	Dieser Request verändert mehrere Datenbankeintrag in der Tabelle.
GET	/api/IdentityXRoles/GetSessionTokenAsync	Dieser Request übermittelt den derzeitigen SessionToken von der aktiven Session.

Abbildung 36: HTTP-Requests des Projektes

LoginSessions		
GET	/api/LoginSessions/Count	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden.
GET	/api/LoginSessions/Count/{predicate}	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden.
GET	/api/LoginSessions/{id}	Dieser Request übermittelt den Wert welcher die dazugehörige Id hat
DELETE	/api/LoginSessions/{id}	Dieser Request löscht einen Datenbankeintrag mit der dazugehörigen Id in der Tabelle.
GET	/api/LoginSessions	Dieser Request übermittelt uns alle Werte der dazugehörigen Tabelle
POST	/api/LoginSessions	Dieser Request erzeugt einen neuen Datenbankeintrag in die Tabelle.
PUT	/api/LoginSessions	Dieser Request verändert einen Datenbankeintrag in der Tabelle.
GET	/api/LoginSessions/Query/{predicate}	Dieser Request übermittelt uns die Tabellenspalten der dazugehörigen Tabelle, welche das Suchkriter
POST	/api/LoginSessions/Array	Dieser Request erzeugt mehrer neue Datenbankeintrag in die Tabelle.
PUT	/api/LoginSessions/Array	Dieser Request verändert mehrere Datenbankeintrag in der Tabelle.
GET	/api/LoginSessions/GetSessionTokenAsync	Dieser Request übermittelt den derzeitigen SessionToken von der aktiven Session.

Abbildung 37: HTTP-Requests des Projektes

SensorDatas		
GET	/api/SensorDatas/Count	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden.
GET	/api/SensorDatas/Count/{predicate}	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden, wenn wir...
GET	/api/SensorDatas/{id}	Dieser Request übermittelt den Wert welcher die dazugehörige Id hat
DELETE	/api/SensorDatas/{id}	Dieser Request löscht einen Datenbankeintrag mit der dazugehörigen Id in der Tabelle.
GET	/api/SensorDatas	Dieser Request übermittelt uns alle Werte der dazugehörigen Tabelle
POST	/api/SensorDatas	Dieser Request erzeugt einen neuen Datenbankeintrag in die Tabelle.
PUT	/api/SensorDatas	Dieser Request verändert einen Datenbankeintrag in der Tabelle.
GET	/api/SensorDatas/Query/{predicate}	Dieser Request übermittelt uns die Tabellenspalten der dazugehörigen Tabelle, welche das Suchkriterium erfüllen
POST	/api/SensorDatas/Array	Dieser Request erzeugt mehrer neue Datenbankeintrag in die Tabelle.
PUT	/api/SensorDatas/Array	Dieser Request verändert mehrere Datenbankeintrag in der Tabelle.
GET	/api/SensorDatas/GetSessionTokenAsync	Dieser Request übermittelt den derzeitigen SessionToken von der aktiven Session.

Abbildung 38: HTTP-Requests des Projektes

Sensors		
GET	/api/Sensors/Count	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden.
GET	/api/Sensors/Count/{predicate}	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden, wenn wir...
GET	/api/Sensors/{id}	Dieser Request übermittelt den Wert welcher die dazugehörige Id hat
DELETE	/api/Sensors/{id}	Dieser Request löscht einen Datenbankeintrag mit der dazugehörigen Id in der Tabelle.
GET	/api/Sensors	Dieser Request übermittelt uns alle Werte der dazugehörigen Tabelle
POST	/api/Sensors	Dieser Request erzeugt einen neuen Datenbankeintrag in die Tabelle.
PUT	/api/Sensors	Dieser Request verändert einen Datenbankeintrag in der Tabelle.
GET	/api/Sensors/Query/{predicate}	Dieser Request übermittelt uns die Tabellenspalten der dazugehörigen Tabelle, welche das Suchkriterium erfüllen
POST	/api/Sensors/Array	Dieser Request erzeugt mehrer neue Datenbankeintrag in die Tabelle.
PUT	/api/Sensors/Array	Dieser Request verändert mehrere Datenbankeintrag in der Tabelle.
GET	/api/Sensors/GetSessionTokenAsync	Dieser Request übermittelt den derzeitigen SessionToken von der aktiven Session.

Abbildung 39: HTTP-Requests des Projektes

Users		
GET	/api/Users/Count	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden.
GET	/api/Users/Count/{predicate}	Dieser Request ergibt uns eine Anzahl wie viele Einträge sich in der dazugehörigen Tabelle befinden, wenn wir na
GET	/api/Users/{id}	Dieser Request übermittelt den Wert welcher die dazugehörige Id hat
DELETE	/api/Users/{id}	Dieser Request löscht einen Datenbankeintrag mit der dazugehörigen Id in der Tabelle.
GET	/api/Users	Dieser Request übermittelt uns alle Werte der dazugehörigen Tabelle
POST	/api/Users	Dieser Request erzeugt einen neuen Datenbankeintrag in die Tabelle.
PUT	/api/Users	Dieser Request verändert einen Datenbankeintrag in der Tabelle.
GET	/api/Users/Query/{predicate}	Dieser Request übermittelt uns die Tabellenspalten der dazugehörigen Tabelle, welche das Suchkriterium erfüllen
POST	/api/Users/Array	Dieser Request erzeugt mehrer neue Datenbankeintrag in die Tabelle.
PUT	/api/Users/Array	Dieser Request verändert mehrere Datenbankeintrag in der Tabelle.
GET	/api/Users/GetSessionTokenAsync	Dieser Request übermittelt den derzeitigen SessionToken von der aktiven Session.

Abbildung 40: HTTP-Requests des Projektes

Implementierung von MQTT in unserem Projekt

Das Einlesen und Überprüfen von den Messwerten soll stattfinden, sobald die WebApi läuft. Aus diesem Grund wird die RuleEngine, welche den Aufruf für das Einlesen und Überprüfen der Messwerte beinhaltet, instanziiert, welche gleichzeitig wie die WebApi startet.

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace SnQPoolIot.WebApi
{
    0 references
    public class Program
    {
        0 references
        public static void Main(string[] args)
        {
            // Aufruf der RuleEngine
            = RuleEngine.Instance.
            CreateHostBuilder(args).Build().Run();
        }

        1 reference
        public static IHostBuilder CreateHostBuilder(string[] args) =>
        {
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
        }
    }
}

```

Abbildung 41: Aufruf der RuleEngine

```

private static RuleEngine _ruleEngine = null;

1 reference
public static RuleEngine Instance {
    get
    {
        if(_ruleEngine == null)
        {
            _ruleEngine = new RuleEngine();
        }
        return _ruleEngine;
    }
}

3 references
public MqttActions MqttActions { get; }

1 reference
private RuleEngine()
{
    MqttActions = new MqttActions();

    MqttActions.OnMqttMessageReceived += MqttActions_OnMqttMessageReceived;

    foreach (var sensorName in Enum.GetNames(typeof(SensorName)))
    {
        MqttMeasurementDto sensor = new MqttMeasurementDto();
        // Inserten der Sensoren, welche sich in der SensorBox befinden
        var hasInserted = InsertSensors(sensorName);
        Sensors.Add(sensorName.ToLower(), sensor);
        // Startet das Einlesen der Messwerte
        MqttActions.StartMqttClientAndRegisterObserverAsync($"{sensorName.ToLower()}/state").Wait();
    }
}

```

Abbildung 42: Instanzieren der RuleEngine und der Sensoren zum Einlesen der Messwerte

Die einzelnen Sensoren, welche in der obigen Abbildung in die Datenbank gespeichert werden, befinden sich in dem nachstehenden enum:

```

public enum SensorName
{
    NeoPixel,
    Noise,
    Temperature,
    Humidity,
    Pressure,
    Motion,
    Co2
}

```

Abbildung 43: Liste der Sensoren gespeichert in einem Enum

Auf diese Sensoren wird nun die Methode `StartMqttClientAndRegisterObserverAsync` angewendet, welche in den nachstehenden Abbildungen und Texten beschrieben wird:

Damit das Backend einen Wert von einem Topic bekommen kann, muss auf dieses Topic subscribed werden. In C# gibt es eigene `NuggetPackages` zum Subscriben.

Nachdem die `NuggetPackages` installiert worden sind, kann nun ein MQTT Client mittels einer `MQTTFactory` erzeugt werden. Damit der MQTT Client auf den richtigen Broker subscriben kann, muss er sogenannte `MQTTClientOptions`, mittels eines `MQTT-ClientOptionsBuilder`, auf sich zugewiesen bekommen.

Zu diesen `MQTTClientOptions` zählt die `ClientId`, die Adresse, der Username und das Passwort des Brokers, sowie der Port auf dem der Broker läuft.

```
public async Task<Task<int>> StartMqttClientAndRegisterObserverAsync(string specifiedTopic)
{
    var configuration = new ConfigurationBuilder()
        .AddJsonFile("appsettings.Development.json")
        .Build();

    Console.WriteLine("SnQPoolIot");
    Guid g = Guid.NewGuid();
    var mqttClientId = Convert.ToString(g); // Unique ClientId
    var mqttBrokerAddress = configuration.GetValue<string>("Mqtt:mqttBrokerAddress"); // hostname or IP address of y
    var mqttBrokerUsername = configuration.GetValue<string>("Mqtt:mqttBrokerUsername"); // Broker Auth username
    var mqttBrokerPassword = configuration.GetValue<string>("Mqtt:mqttPassword"); // Broker Auth password
    var topic = configuration.GetValue<string>("Mqtt:mqttTopic") + specifiedTopic; // topic to subscribe to

    var mqttClient = new MqttFactory().CreateManagedMqttClient();
    var mqttClientOptions = new ManagedMqttClientOptionsBuilder()
        .WithAutoReconnectDelay(TimeSpan.FromSeconds(2))
        .WithClientOptions(new MqttClientOptionsBuilder()
            .WithTcpServer(mqttBrokerAddress, 1883)
            .WithClientId(mqttClientId)
            .WithCredentials(mqttBrokerUsername, mqttBrokerPassword)
            .WithCleanSession()
            .Build()
        )
        .Build();
}
```

Abbildung 44: MQTT Client Konfiguration

Sobald die Konfiguration eingetragen wurde, wird dem MQTT Client nun mitgeteilt, was zu tun ist, wenn sich der MQTT Client connected, disconnected oder eine Nachricht bekommt.

```
mqttClient.ApplicationMessageReceivedHandler = new MqttApplicationMessageReceivedHandlerDelegate(e => MqttOnNewMessageReceived(e));
mqttClient.ConnectedHandler = new MqttClientConnectedHandlerDelegate(e => MqttOnConnected(e));
mqttClient.DisconnectedHandler = new MqttClientDisconnectedHandlerDelegate(e => MqttOnDisconnected(e));

await mqttClient.SubscribeAsync(new MqttTopicFilterBuilder().WithTopic(topic).WithExactlyOnceQoS().Build());
await mqttClient.StartAsync(mqttClientOptions);
```

Abbildung 45: MQTT Cient HandlerDelegate, Subscriben und Starten

Wenn sich ein MQTT Client connected oder disconnected, wird dies ,durch die selbstgeschriebenen Logging Klasse, mitgeloggt.

```

1 reference
private static void MqttOnConnected(MqttClientConnectedEventArgs e)
{
    LogWriter.Instance.LogWrite($"MQTT Client: Connected with result: {e.ConnectResult.ResultCode}");
}

1 reference
private static void MqttOnDisconnected(MqttClientDisconnectedEventArgs e)
{
    LogWriter.Instance.LogWrite($"MQTT Client: Broker connection lost with reason: {e.Reason}");
}

```

Abbildung 46: MQTT Cient HandlerDelegate, Subscriben und Starten

Auch beim Empfangen einer neuen Message wird dies durch die Logging Klasse mitgeloggt, jedoch ist dies nicht die Hauptfunktion der Methode MqttOneNewMessage, sondern das Speichern (Inserten) von Messwerten in die Datenbank.

```

private async Task MqttOnNewMessageAsync(MqttApplicationMessageReceivedEventArgs e)
{
    var mqttPayloadData = Encoding.UTF8.GetString(e.ApplicationMessage.Payload);
    string[] datavalue = mqttPayloadData.Split(new char[] { ':', ',', '{', '}' });

    using var ctrl = Factory.Create<SnqPoolIot.Contracts.Persistence.PoolIot.ISensorData>();
    var entity = await ctrl.CreateAsync();
    var measurment = new MqttMeasurementDto();

    entity.SensorId = GetSensorId(e);
    measurment.SensorId = entity.SensorId;
    measurment.SensorName = currentTopic;

    for (int i = 0; i < datavalue.Length; i++)
    {
        if (i == 2)
        {
            measurment.Timestamp = DateTime.Now;
        }
        else if (i == 4)
        {
            entity.Value = datavalue[i].Trim();
            measurment.Value = entity.Value;
        }
    }

    OnMqttMessageReceived?.Invoke(this, measurment);

    await ctrl.InsertAsync(entity);
    await ctrl.SaveChangesAsync();

    LogWriter.Instance.LogWrite($"MQTT Client: OnNewMessage Topic: {e.ApplicationMessage.Topic} / Message: {mqttPayloadData}");
}

```

Abbildung 47: MQTT Cient beim Empfangen eines neuen Messwertes

Sobald ein Messwert in der Datenbank hinzugefügt wurde, wird die RuleEngine alarmiert, welche wiederum eine Überprüfung des Messwertes durchführt. Ein gutes Beispiel hierfür ist die Methode CheckNoiceSensorData, welche überprüft, ob der gemessene Wert über 300 db hat.

```

public static int CheckNoiseSensorData(int? sensorValue)
{
    var result = 0;
    if (sensorValue == null)
    {
        LogWriter.Instance.LogWrite($"The SensorValue is null!");
        return -1;
    }
    else if (sensorValue > 300)
    {
        MessageNotification.SendMessageByTelegram("Achtung es wurden zu Laute Messwerte gemessen bitte schauen Sie noch warum es sich handelt");
    }

    return result;
}

```

Abbildung 48: Kontrollieren eines Messwertes von dem Noicesensor

Sobald die Methode `CheckNoiseSensorData` ausgeführt wurde und der gemessene Wert über 300 db liegt, wird der Benutzer über Telegram verständigt. Aus diesem Grund wurde die Klasse `MessageNotification`, welche die Methode `SendMessageByTelegram` beinhaltet, geschrieben. Wie der Name `SendMessageByTelegram` schon sagt, wird in dieser Methode eine Message mittels Telegram über einen Bot versendet. Für diesen Vorgang wird der Token, welcher für den Zugriff der HTTP API zuständig ist, und die `ChatId` benötigt. Diese sind in einem `appsettings.json` File gespeichert und werden über die Variable `configuration` eingelesen.

```

public class MessageNotification
{
    /// <summary>
    /// Die Notification wird an Telegram übertragen.
    /// </summary>
    /// <param name="notification"></param>
    /// <returns></returns>
    1 reference
    public static bool SendMessageByTelegram(string notification)
    {
        var configuration = new ConfigurationBuilder()
            .AddJsonFile("appsettings.Development.json")
            .Build();
        string apiToken = configuration.GetValue<string>("Telegram:apiToken");
        string chatId = configuration.GetValue<string>("Telegram:chatId");
        string urlString = "https://api.telegram.org/bot{0}/sendMessage?chat_id={1}&text={2}";
        string messageText = notification;
        urlString = String.Format(urlString, apiToken, chatId, messageText);
    }
}

```

Abbildung 49: Configuration Daten für das Senden der Benachrichtigung

```
try
{
    WebRequest request = WebRequest.Create(urlString);
    WebResponse response = request.GetResponse();
    string responseStatus = ((HttpWebResponse)response).StatusDescription;
    if (responseStatus == "OK")
    {
        using Stream dataStream = response.GetResponseStream();
        using StreamReader reader = new(dataStream);
        string responseFromServer = reader.ReadToEnd();
        if (responseFromServer.Contains("\"ok\":true"))
        {
            LogWriter.Instance.LogWrite("Message text was successfully sent to Telegram");
        }
        else
        {
            LogWriter.Instance.LogWrite("Failed to send the message text to Telegram");
        }
        LogWriter.Instance.LogWrite(responseFromServer);
    }
    else
    {
        LogWriter.Instance.LogWrite("Failed to send the message text to Telegram");
        LogWriter.Instance.LogWrite("Response status: " + responseStatus);
    }
    response.Close();
    return true;
}
catch (Exception ex)
{
    LogWriter.Instance.LogWrite("Failed to send the message text to Telegram");
    Console.WriteLine(Environment.NewLine);
    Console.WriteLine(ex.ToString());
    return false;
}
```

Abbildung 50: Verwendung von der Configuration



Abbildung 51: Veranschaulichung von gesendeten Benachrichtigungen

4.2.3 SnQPoollot.AspMvc

Für das Veranschaulichen von Sensoren und gemessenen SensorDaten wurde eine AspMvc-Anwendung programmiert.

Eine Ansicht über aktuelle Sensoren wird folgendermaßen angezeigt:

SnQPoolot Home Show PoolData Show Sensors Contact

Sensors

Sensors.Create new...






















Name	
neopixel	  
noise	  
temperature	  
humidity	  
pressure	  
motion	  
co2	  

Abbildung 52: Veranschaulichung von den Sensoren in der AspMvc Anwendung

5 Zusammenfassung

Das Projekt war für mich persönlich ein riesiger Aufwand, der sich aber sicherlich gelohnt hat. Es wurden Hardware- als auch Software-technisch viele Technologien gelernt und vertieft. Beispiele dafür sind die Architektur eines Raspberry Pi's, Verwendung von Docker im Projekt, Übermitteln von Daten mittels Telegram und noch etliche andere Technologien, welche bereits beschrieben worden sind.

Zusätzlich, wie bereits im Kapitel 4.1 Projektmanagement angeführt, wurden weitere Erfahrungen im Themenbereich Projektentwicklung und Projektplanung dazu gewonnen, wie zum Beispiel Kontakthaltung zwischen dem Projektleiter und dem Kunden, detailliertes Schreiben von Projektberichten, Verhaltensweise bei Misserfolgen und Kommunikation in einem Projektteam.

Literaturverzeichnis

Abbildungsverzeichnis

1	Grafische Darstellung des Projektes	I
2	Raspberry Pi des Projektes	3
3	Samba auf Laptop	4
4	MQTT-Explorer Login	6
5	MQTT-Explorer nachdem Login	7
6	DotNet6ConApp	9
7	DotNet6ConApp	10
8	DotNet5ConApp	11
9	SQLite Extension für Visual Studio Code	12
10	Innenleben der SQLite Datenbank	12
11	Implementierung partial Class und Method	13
12	Implementierung partial Class und Method	14
13	Erzeugen eines Bots über den BotFather	15
14	Beispiel eines Tokens der zum Zugriff auf die API benötigt wird.	16
15	Ausschnitt aus den Projektberichten	19
16	Visualisierung einer Projektampel	19
17	Projektmappe	20
18	NuggetPackage für Entityframework mit Sqlite	21
19	Befehl zum Erzeugen von Migationen	21
20	Erzeugte Migrationen in der Logik	22
21	DbContext zum Erzeugen einer Datenbank	22
22	Identity Tabelle mit den dazugehörigen Tabellenspalten	23
23	Role Tabelle mit den dazugehörigen Tabellenspalten	23
24	IdentityXRole Tabelle mit den dazugehörigen Tabellenspalten	24
25	LoginSession Tabelle mit den dazugehörigen Tabellenspalten	24
26	Sensor Tabelle mit den dazugehörigen Tabellenspalten	25
27	SensorData Tabelle mit den dazugehörigen Tabellenspalten	25
28	Aufbau der WebApi	28
29	Auszug GenericController der WebApi	29
30	Anwendung des GenericControllers der WebApi	29
31	Implementierung von Swagger in der WebApi	30
32	HTTP-Requests des Projektes	31
33	HTTP-Requests des Projektes	31
34	HTTP-Requests des Projektes	32
35	HTTP-Requests des Projektes	32
36	HTTP-Requests des Projektes	33
37	HTTP-Requests des Projektes	33
38	HTTP-Requests des Projektes	34
39	HTTP-Requests des Projektes	34
40	HTTP-Requests des Projektes	35
41	Aufruf der RuleEngine	35
42	Instanzieren der RuleEngine und der Sensoren zum Einlesen der Messwerte	36

43	Liste der Sensoren gespeichert in einem Enum	36
44	MQTT Client Konfiguration	37
45	MQTT Client HandlerDelegate, Subscriben und Starten	37
46	MQTT Client HandlerDelegate, Subscriben und Starten	38
47	MQTT Client beim Empfangen eines neuen Messwertes	38
48	Kontrollieren eines Messwertes von dem Noicesensor	39
49	Configuration Daten für das Senden der Benachrichtigung	39
50	Verwendung von der Configuration	40
51	Veranschaulichung von gesendeten Benachrichtigungen	41
52	Veranschaulichung von den Sensoren in der AspMvc Anwendung	42

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang