

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

Gesamtbild des Projektes wird hier noch eingefügt...



Inhaltsverzeichnis

1	Einleitung	1
2	Umfeldanalyse	2
3	Technologien	3
3.1	Raspberry Pi	3
3.2	MQTT	5
3.3	.Net	7
4	Projektumsetzung	14
4.1	Backend Projekt-Überblick	14
5	Zusammenfassung	29
	Literaturverzeichnis	IV
	Abbildungsverzeichnis	V
	Tabellenverzeichnis	VI
	Quellcodeverzeichnis	VII
	Anhang	VIII

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 Temperatursensors 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

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.

(Hier kommt noch ein Foto vom Raspi rein)

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.

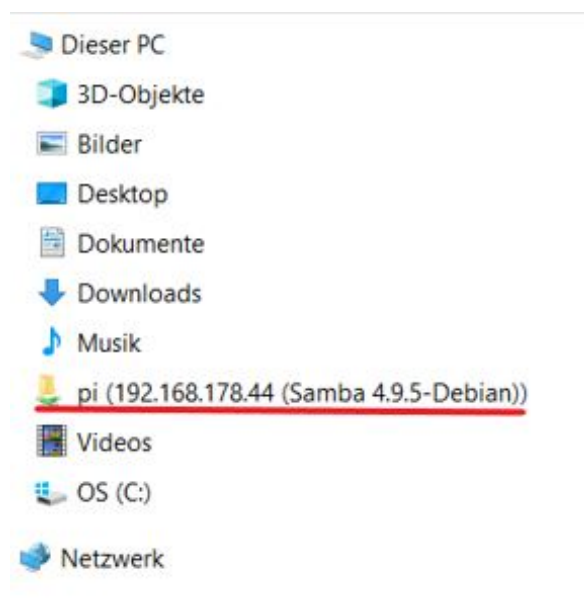


Abbildung 1: Samba auf Laptop

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. Genaueres 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, welcher 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 Funktionalität, dass ein Gerät immer eine bestimmte Ip-Adresse zugewiesen bekommt.

3.2 MQTT

3.2.1 Was ist MQTT:

MQTT ausgeschrieben 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 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.

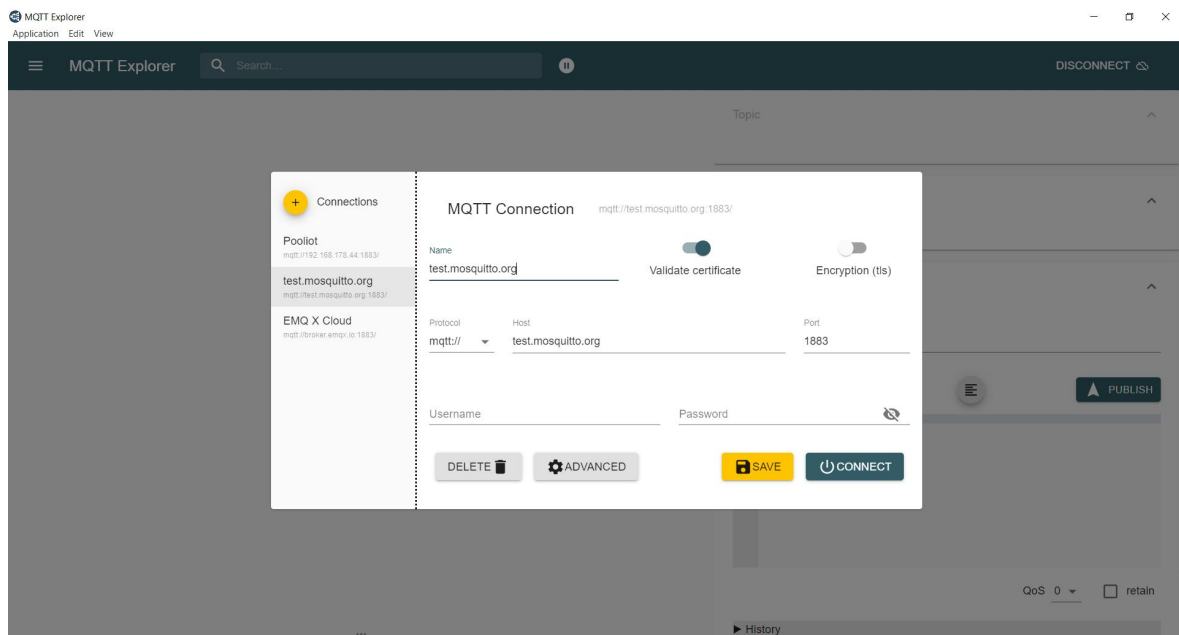


Abbildung 2: MQTT-Explorer Login

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.

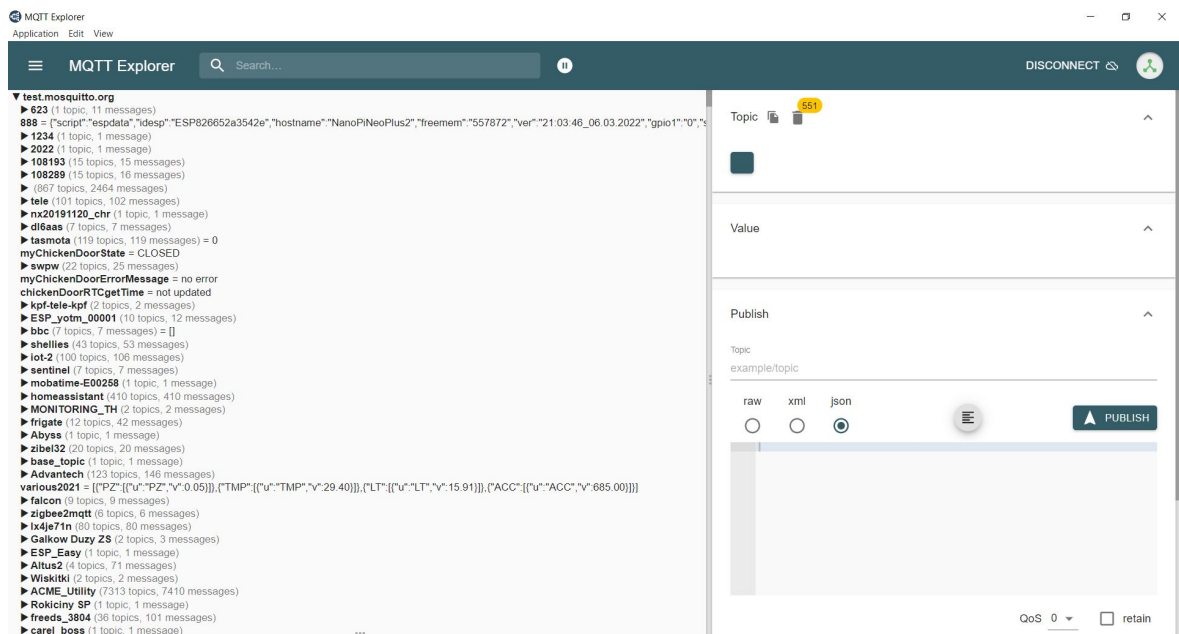


Abbildung 3: MQTT-Explorer nachdem Login

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 versteht man unter .Net Framework und .Net Core

Bisher wurde .NetFramework genutzt um .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 der Kater der jahrelangen traditionellen 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. Der letzte große 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 Vorteil von .Net Core, die hier nicht erwähnt wurden.

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 andere Programmiersprachen an.

Wie bindet man einen Swagger in C# ein

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



Abbildung 4: DotNet6ConApp

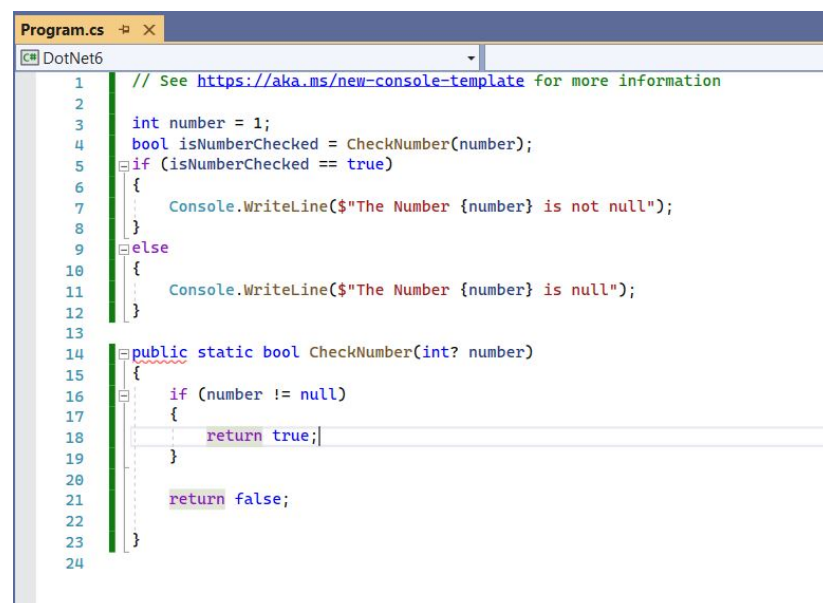
Die im Projekt verwendete Implementation wird im Unterkapitel Verwendung von Swagger im Projekt dargestellt.

3.3.2 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, während 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 5: DotNet6ConApp

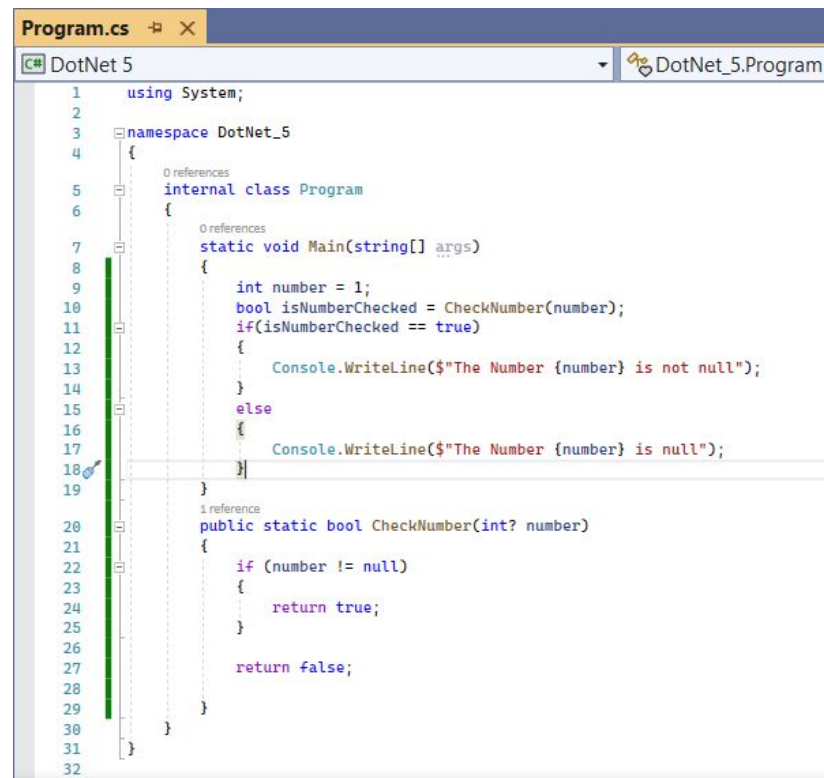


Abbildung 6: DotNet5ConApp

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

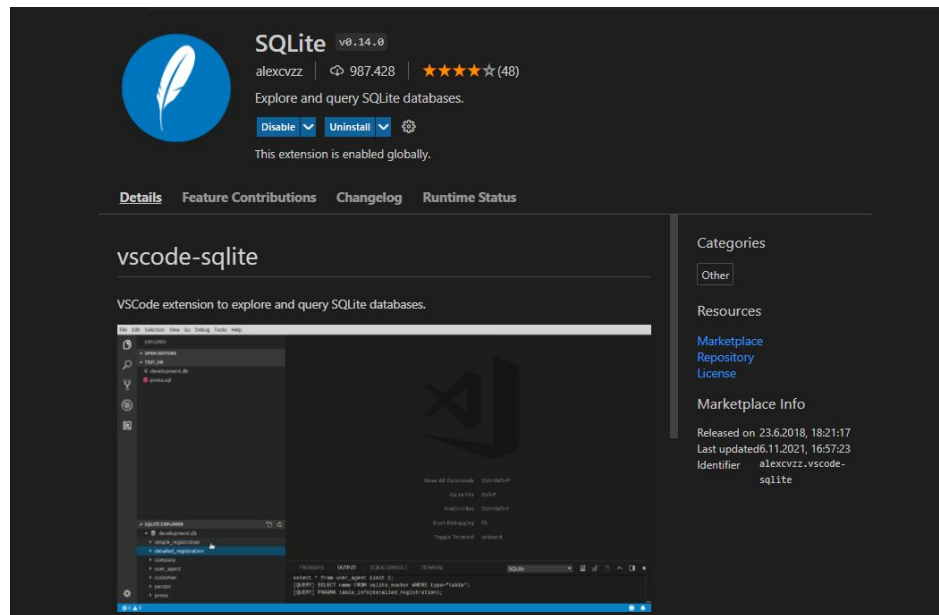


Abbildung 7: SQLite Extension für Visual Studio Code

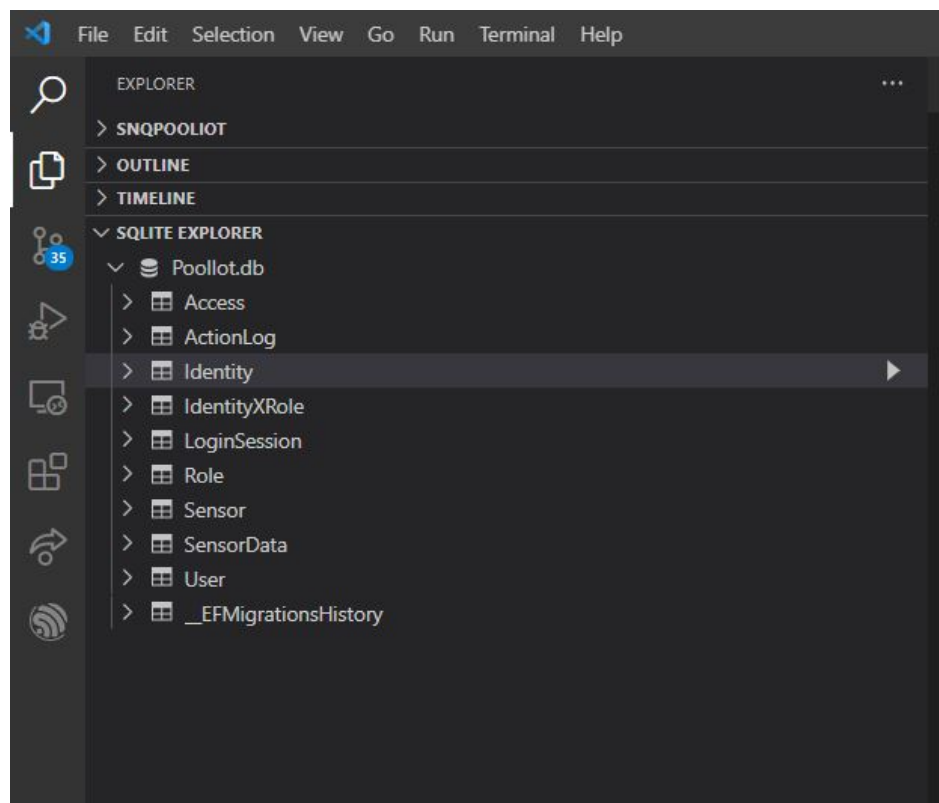


Abbildung 8: Innenleben unserer SQLite Datenbank

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 man wartet ja nicht bis zum Beispiel 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.

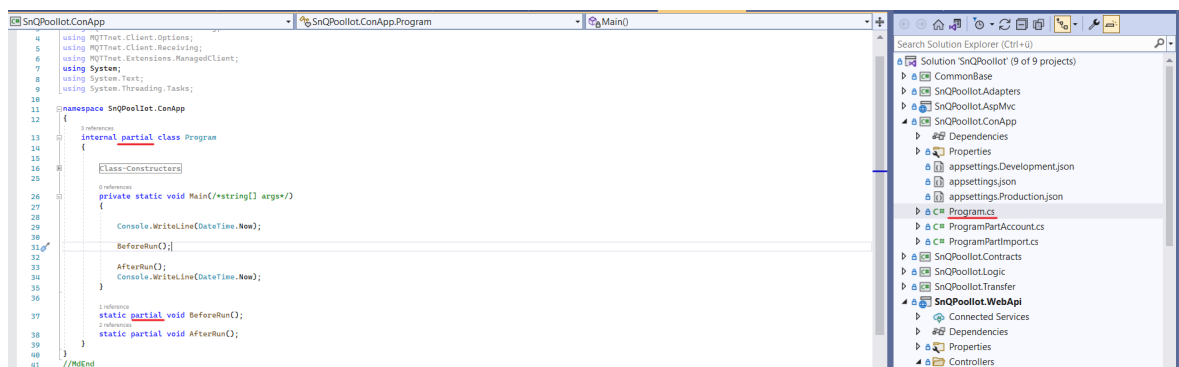


Abbildung 9: Implementierung `partial` Class und Method

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.

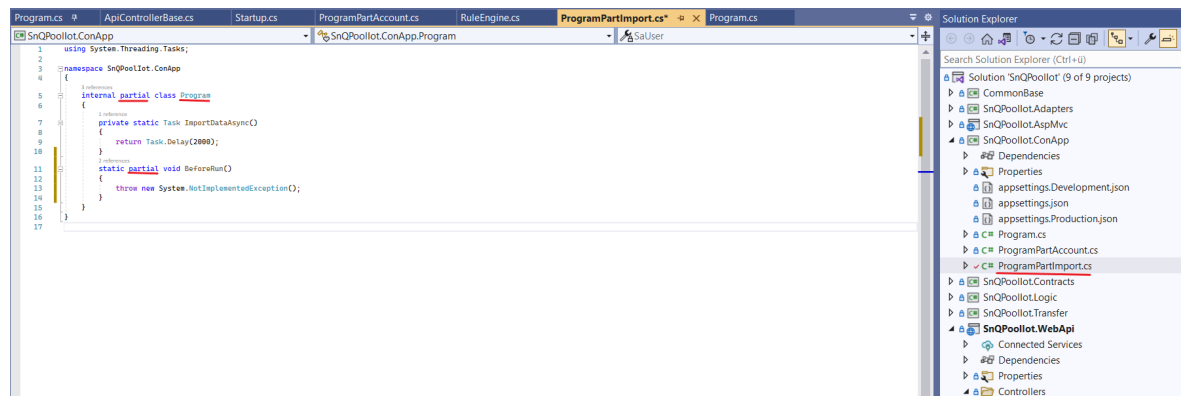


Abbildung 10: Implementierung partial Class und Method

Begriffserklärungen:

/subsectionC#

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

/subscribeARM ARM stand für Acorn RISC Machines, später für Advanced RISC Machines und ist einer der meistverbreitesten Mikroprozessoren.

Quellen: <https://chudovo.de/difference-between-net-core-and-net-framework/>

<https://www.oracle.com/at/database/what-is-database/>

4 Projektumsetzung

4.1 Backend Projekt-Überblick

Das Backend setzt sich aus 9 Projekten zusammen, welche in C# .Net 5 entwickelt wurden.

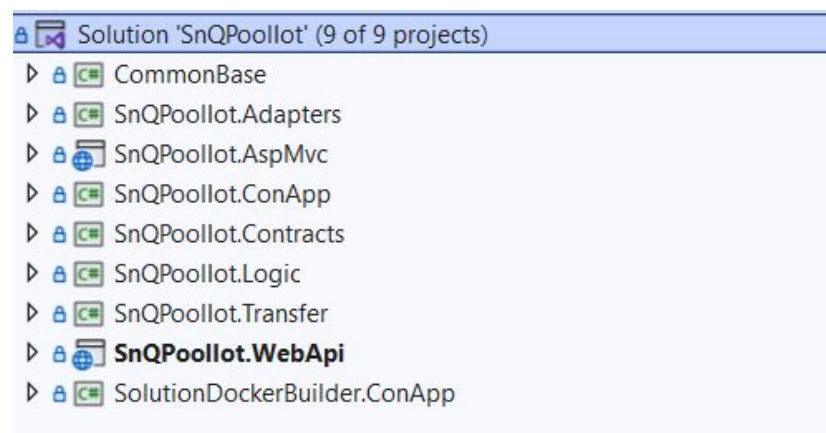


Abbildung 11: Projektmappe

- **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.

- **SnQPoolIot.Logic**
SnQPoolIot.Logic ist das Kernstück des Projektes. Durch die Logic können alle Daten aus der Datenbank verwendet werden. Die Logik 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.

4.1.1 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.

Hier ein Ausschnitt von dem Aufbau der WebApi:

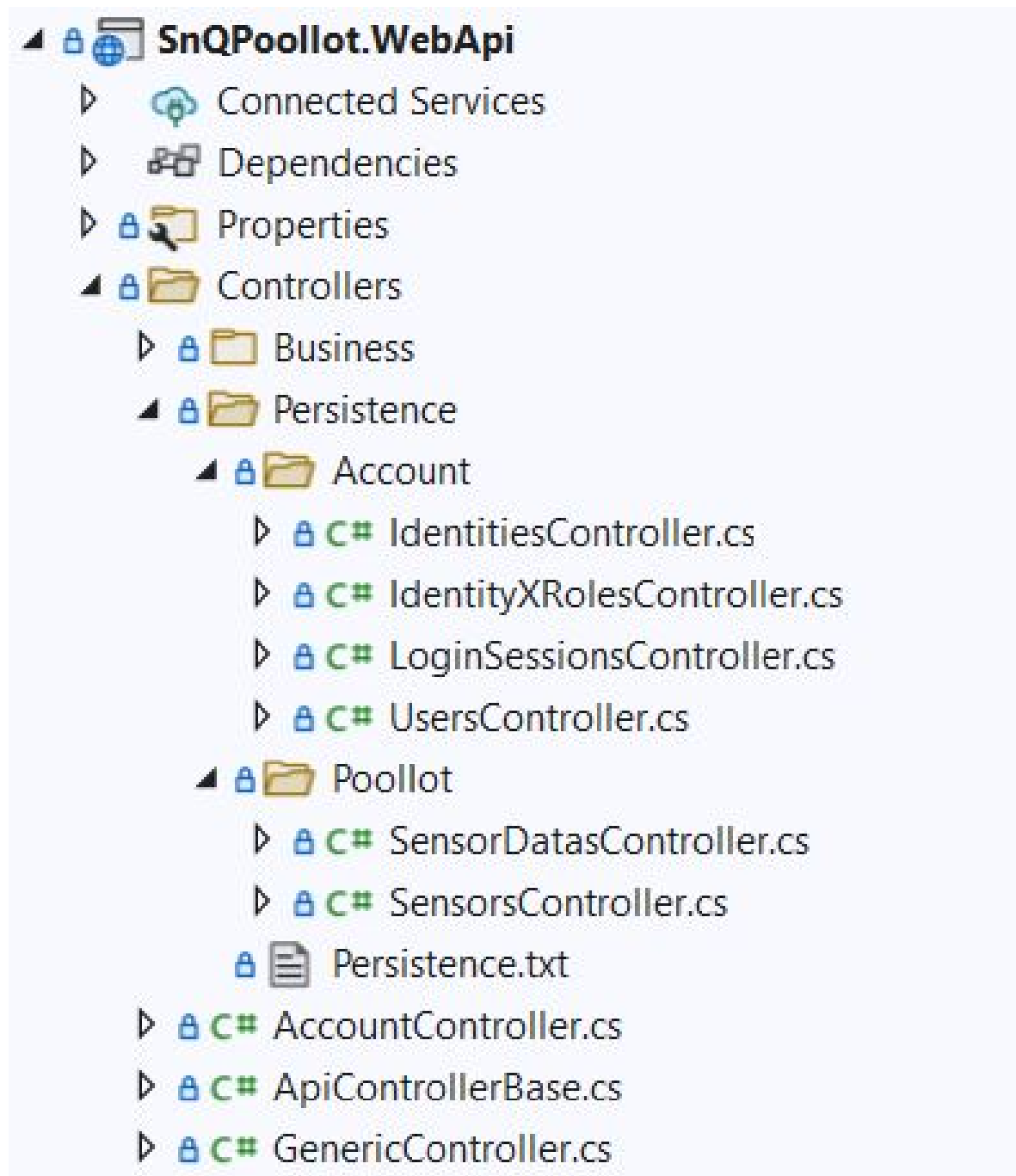


Abbildung 12: Aufbau der WebApi

Das Routing der Websites wird über die Controller gemanaged. 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]")]
D references
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}")]
D references
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]")]
D references
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 13: 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")]
    D references
    public partial class SensorsController : WebApi.Controllers.GenericController<TContract, TModel>
    {
    }
}

```

Abbildung 14: 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 15: 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 16: 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 17: 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 {predicate} angegeben ist.
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 {predicate} erfüllen.
POST	/api/Identities/Array	Dieser Request erzeugt mehrere neue Datenbankeinträge in die Tabelle.
PUT	/api/Identities/Array	Dieser Request verändert mehrere Datenbankeinträge in der Tabelle.
GET	/api/Identities/GetSessionTokenAsync	Dieser Request übermittelt den derzeitigen SessionToken von der aktiven Session.

Abbildung 18: 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 {predicate} angegeben ist.
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 {predicate} erfüllen.
POST	/api/IdentityUsers/Array	Dieser Request erzeugt mehrere neue Datenbankeinträge in die Tabelle.
PUT	/api/IdentityUsers/Array	Dieser Request verändert mehrere Datenbankeinträge in der Tabelle.
GET	/api/IdentityUsers/GetSessionTokenAsync	Dieser Request übermittelt den derzeitigen SessionToken von der aktiven Session.

Abbildung 19: 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 20: 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 Suchkrit
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 21: 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 22: 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 23: 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.

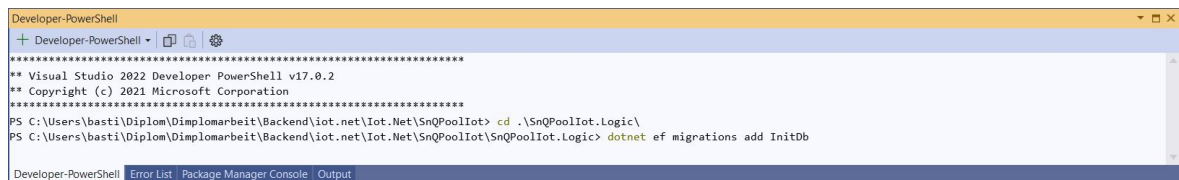
Abbildung 24: HTTP-Requests des Projektes

4.1.2 SnQPoolot.Logic

Wie bereits im Backend Projekt-Überblick beschrieben, befindet sich in SnQPoolot.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 25: NuggetPackage für Entityframework mit Sqlite



```
Developer-PowerShell
+ Developer-PowerShell
*****
** Visual Studio 2022 Developer PowerShell v17.0.2
** Copyright (c) 2021 Microsoft Corporation
*****
PS C:\Users\basti\Diplom\Diplomarbeit\Backend\iot.net\Iot.Net\SnQPoolIot> cd .\SnQPoolIot.Logic\
PS C:\Users\basti\Diplom\Diplomarbeit\Backend\iot.net\Iot.Net\SnQPoolIot.Logic> dotnet ef migrations add InitDb
```

Abbildung 26: Befehl zum Erzeugen von Migrationen

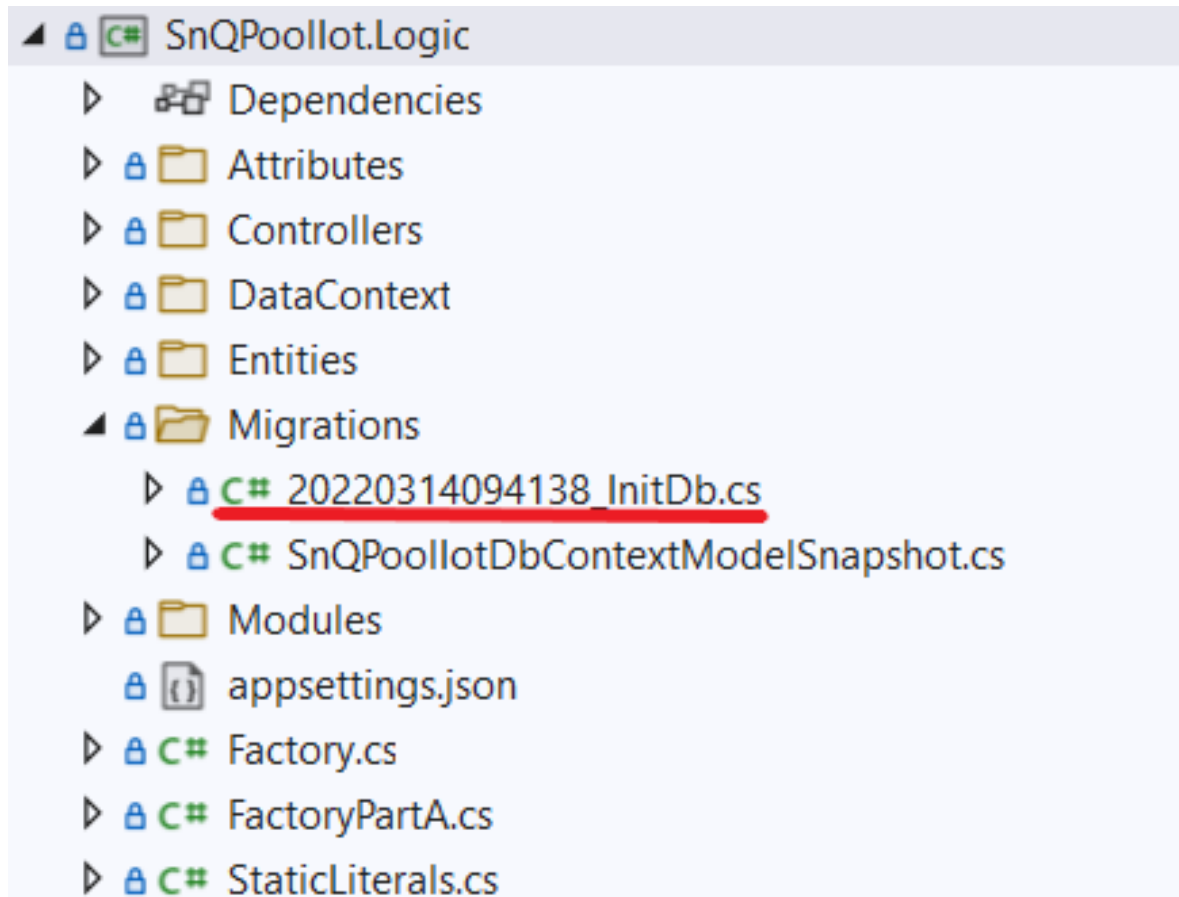


Abbildung 27: Erzeugte Migrationen in der Logik


```

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 28: DbContext zum Erzeugen einer Datenbank

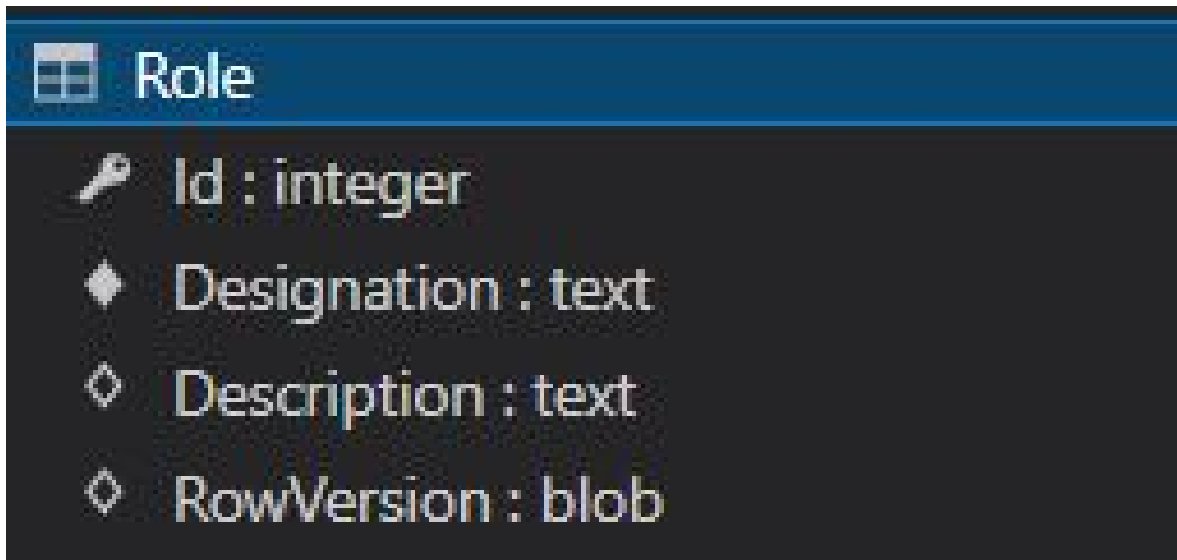
Nun zu den Datenbank Zugriffen. Damit auf die Messwerte zugegriffen werden kann muss sich ein User zuerst authentifizieren. Damit sich eine User authentifizieren kann benötigt er einen Account mit E-Mail und Passwort, welche in der Datenbank gespeichert sind. Für die Authentifizierung werden die nachstehenden 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 29: Identity Tabelle mit den dazugehörigen Tabellenspalten

- Role In dieser Tabelle sind alle Rollen die es gibt vorzufinden.

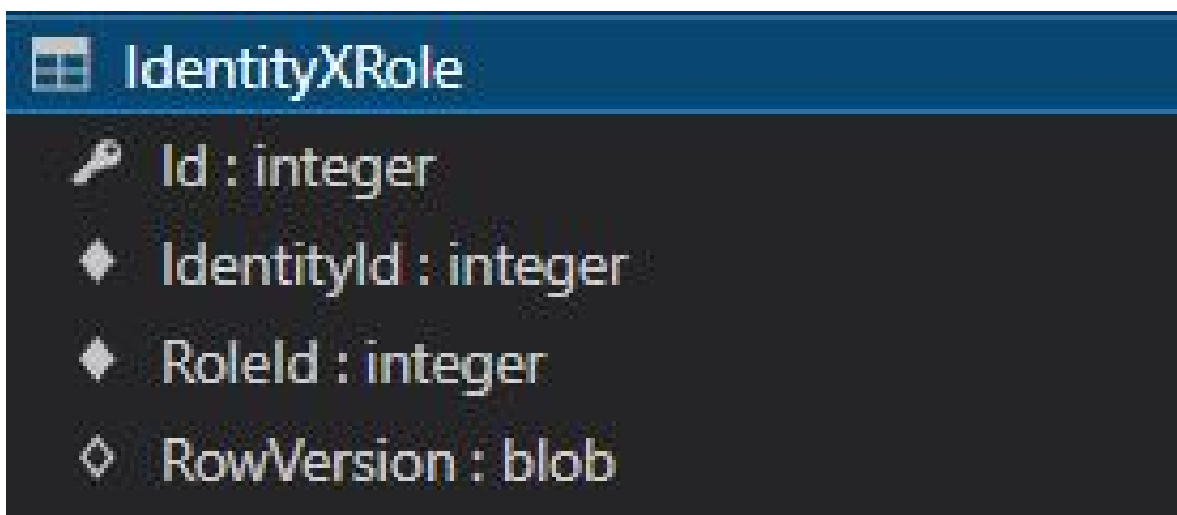


The diagram shows the structure of the 'Role' table. It has a dark blue header with a table icon and the title 'Role'. Below the header, on a dark background, are four entries: 'Id : integer' with a primary key icon (a key), 'Designation : text' with a diamond icon, 'Description : text' with a diamond icon, and 'RowVersion : blob' with a diamond icon.

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

Abbildung 30: Role Tabelle mit den dazugehörigen Tabellenspalten

- IdentityXRole Diese Tabelle weist einem Benutzer eine Rolle zu.

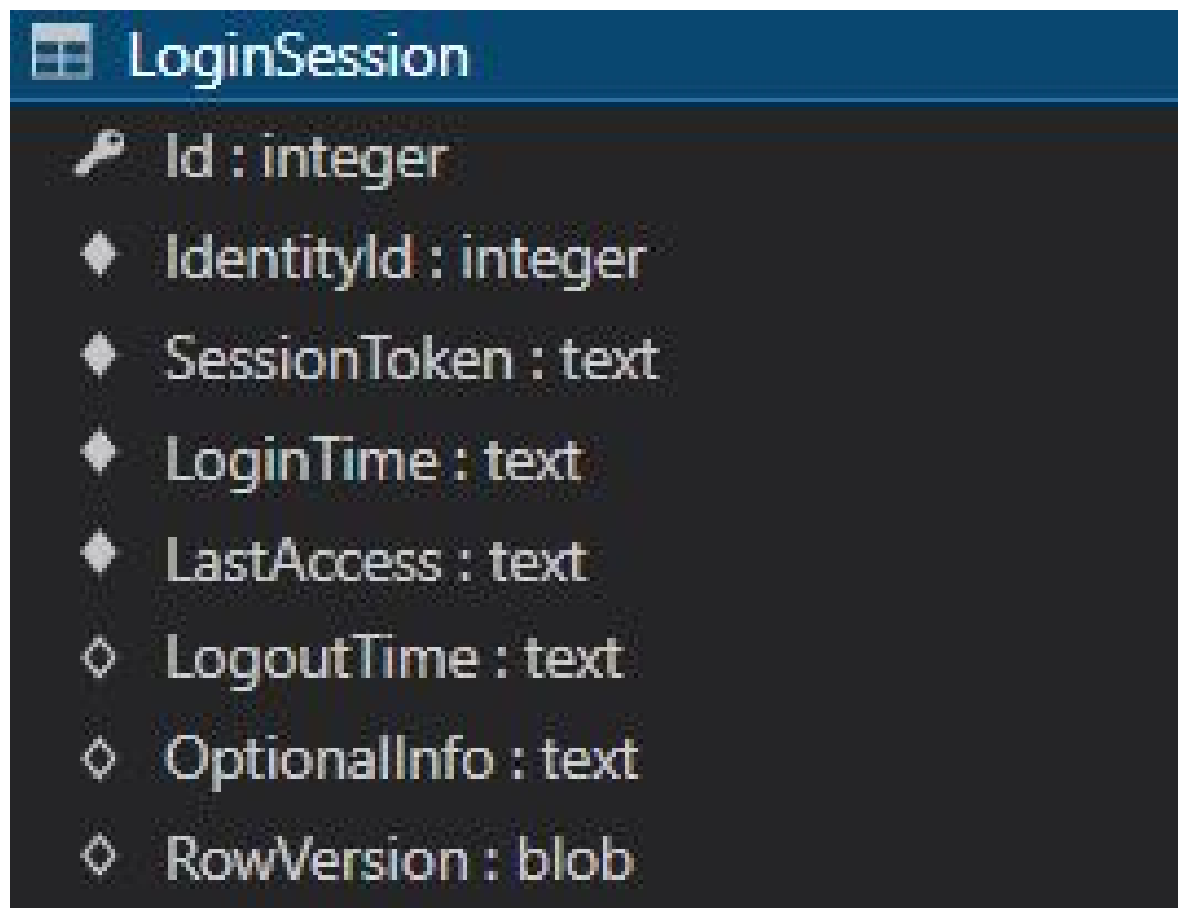


The diagram shows the structure of the 'IdentityXRole' table. It has a dark blue header with a table icon and the title 'IdentityXRole'. Below the header, on a dark background, are four entries: 'Id : integer' with a primary key icon (a key), 'IdentityId : integer' with a diamond icon, 'RoleId : integer'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

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

Abbildung 31: IdentityXRole Tabelle mit den dazugehörigen Tabellenspalten

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

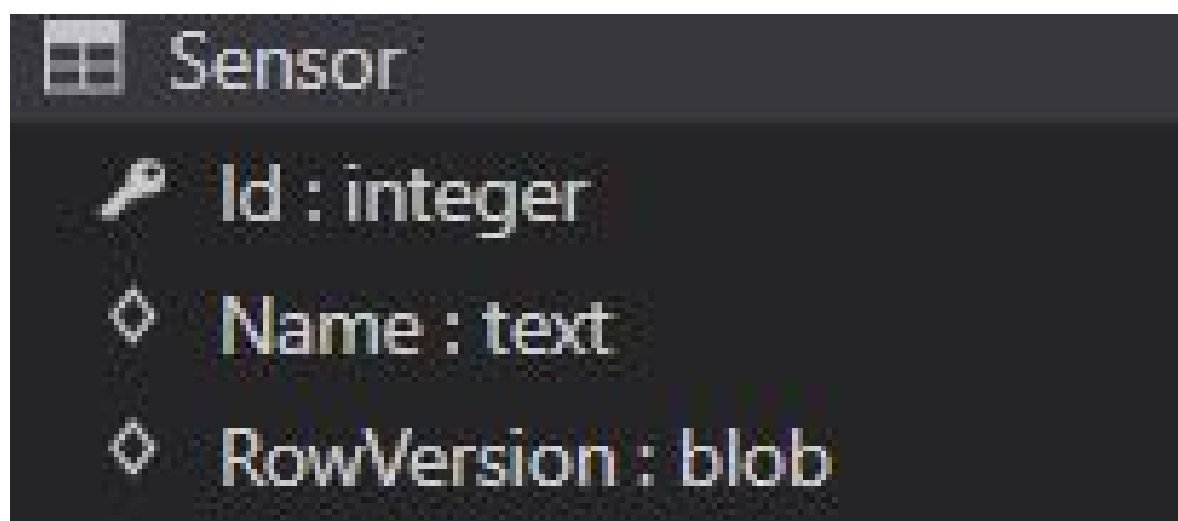


LoginSession	
Id : integer	
IdentityId : integer	
SessionToken : text	
LoginTime : text	
LastAccess : text	
LogoutTime : text	
OptionalInfo : text	
RowVersion : blob	

Abbildung 32: LoginSession Tabelle mit den dazugehörigen Tabellenspalten

Sobald sich ein User authentifiziert hat wurde zugleich eine neue Session erstellt und wenn die Rechte vom dem authentifizierten User hochgenug sind kann er sich nun die Messwerte ansehen. Die nachstehenden Tabellen dienen zum Erfassen von den Messwerten:

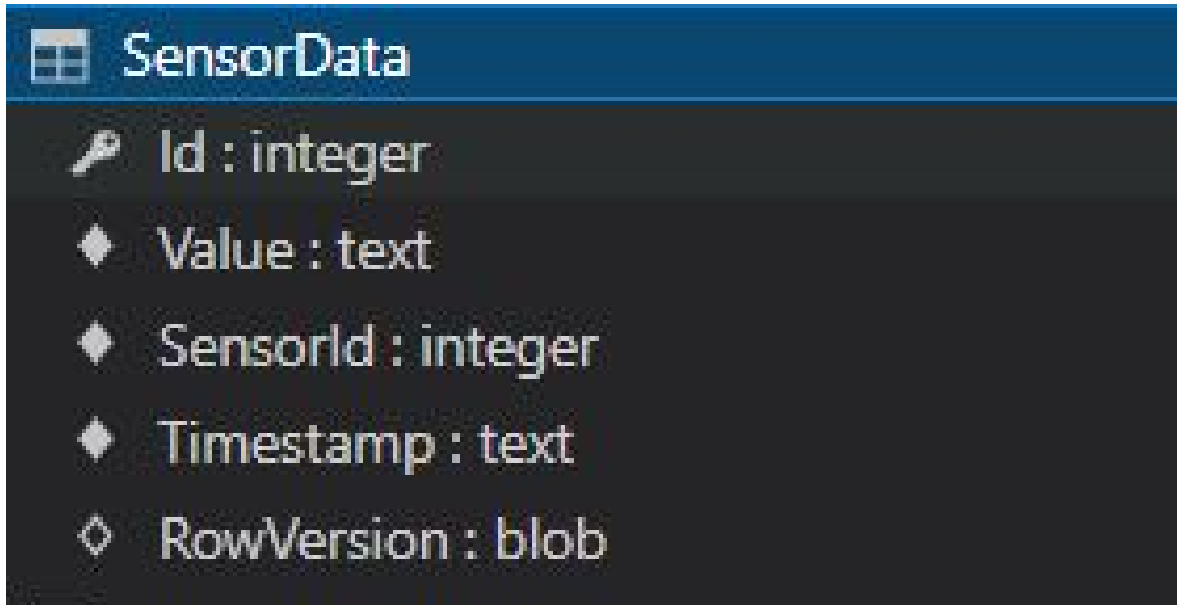
- Sensor Diese Tabelle beinhaltet den Namen eines Sensors.



Sensor	
Id : integer	
Name : text	
RowVersion : blob	

Abbildung 33: Sensor Tabelle mit den dazugehörigen Tabellenspalten

- Sensor Diese Tabelle beinhaltet die Messwerte aller Sensoren und beinhaltet welcher Messwert zu welchen Sensor gehört.



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

Abbildung 34: SensorData Tabelle mit den dazugehörigen Tabellenspalten

Logging in unserem Projekt

4.1.3 SnQPollot.AspMvc

5 Zusammenfassung

Literaturverzeichnis

Abbildungsverzeichnis

1	Samba auf Laptop	3
2	MQTT-Explorer Login	6
3	MQTT-Explorer nachdem Login	6
4	DotNet6ConApp	8
5	DotNet6ConApp	9
6	DotNet5ConApp	10
7	SQLite Extension für Visual Studio Code	11
8	Innenleben unserer SQLite Datenbank	11
9	Implementierung partial Class und Method	12
10	Implementierung partial Class und Method	13
11	Projektmappe	14
12	Aufbau der WebApi	16
13	Auszug GenericController der WebApi	17
14	Anwendung des GenericControllers der WebApi	17
15	Implementierung von Swagger in der WebApi	18
16	HTTP-Requests des Projektes	19
17	HTTP-Requests des Projektes	19
18	HTTP-Requests des Projektes	20
19	HTTP-Requests des Projektes	20
20	HTTP-Requests des Projektes	21
21	HTTP-Requests des Projektes	21
22	HTTP-Requests des Projektes	22
23	HTTP-Requests des Projektes	22
24	HTTP-Requests des Projektes	23
25	NuggetPackage für Entityframework mit Sqlite	23
26	Befehl zum Erzeugen von Migrationen	24
27	Erzeugte Migrationen in der Logik	24
28	DbContext zum Erzeugen einer Datenbank	25
29	Identity Tabelle mit den dazugehörigen Tabellenspalten	25
30	Role Tabelle mit den dazugehörigen Tabellenspalten	26
31	IdentityXRole Tabelle mit den dazugehörigen Tabellenspalten	26
32	LoginSession Tabelle mit den dazugehörigen Tabellenspalten	27
33	Sensor Tabelle mit den dazugehörigen Tabellenspalten	27
34	SensorData Tabelle mit den dazugehörigen Tabellenspalten	28

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang