

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

Zusammenfassung unserer genialen Arbeit. Auf Deutsch. Das ist das einzige Mal, dass eine Grafik in den Textfluss eingebunden wird. Die gewählte Grafik soll irgendwie eure Arbeit repräsentieren. Das ist ungewöhnlich für eine wissenschaftliche Arbeit aber eine Anforderung der Obrigkeit. *Bitte auf keinen Fall mit der Zusammenfassung verwechseln, die den Abschluss der Arbeit bildet!*



Inhaltsverzeichnis

1	Einleitung	1
2	Umfeldanalyse	2
3	Technologien	3
3.1	Raspberry Pi	3
3.2	MQTT	5
3.3	.Net	5
4	Zusammenfassung	10
	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 diese Funktionalitäten zuverlässig in einem Gerät kosteneffizient zusammenzuführen und über ein UI der SP-Anwendung wird ein 360 Grad Blick auf die Geschehnisse im Pool ermöglicht.

2 Umfeldanalyse

3 Technologien

3.1 Raspberry Pi

Das Projekt beinhaltet einen Raspberry Pi 4, welcher als MQTT Broker dient und auf diesem Raspberry Pi läuft auch unser Backend mit DotNet, Docker und Samba. Der Raspberry Pi hat 4 Gigabyte 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 0,5 und maximal 0,7 Watt. Samba Unser Raspberry dient als File-Server und zur leichteren Datenübertragung von Windows auf Linux kann man mit Samba von einem Windows Explorer direkt auf den Raspberry Pi zugreifen. Somit können Files oder Projekte direkt von einem Laptop oder Computer auf den Raspberry PI gelegt werden.

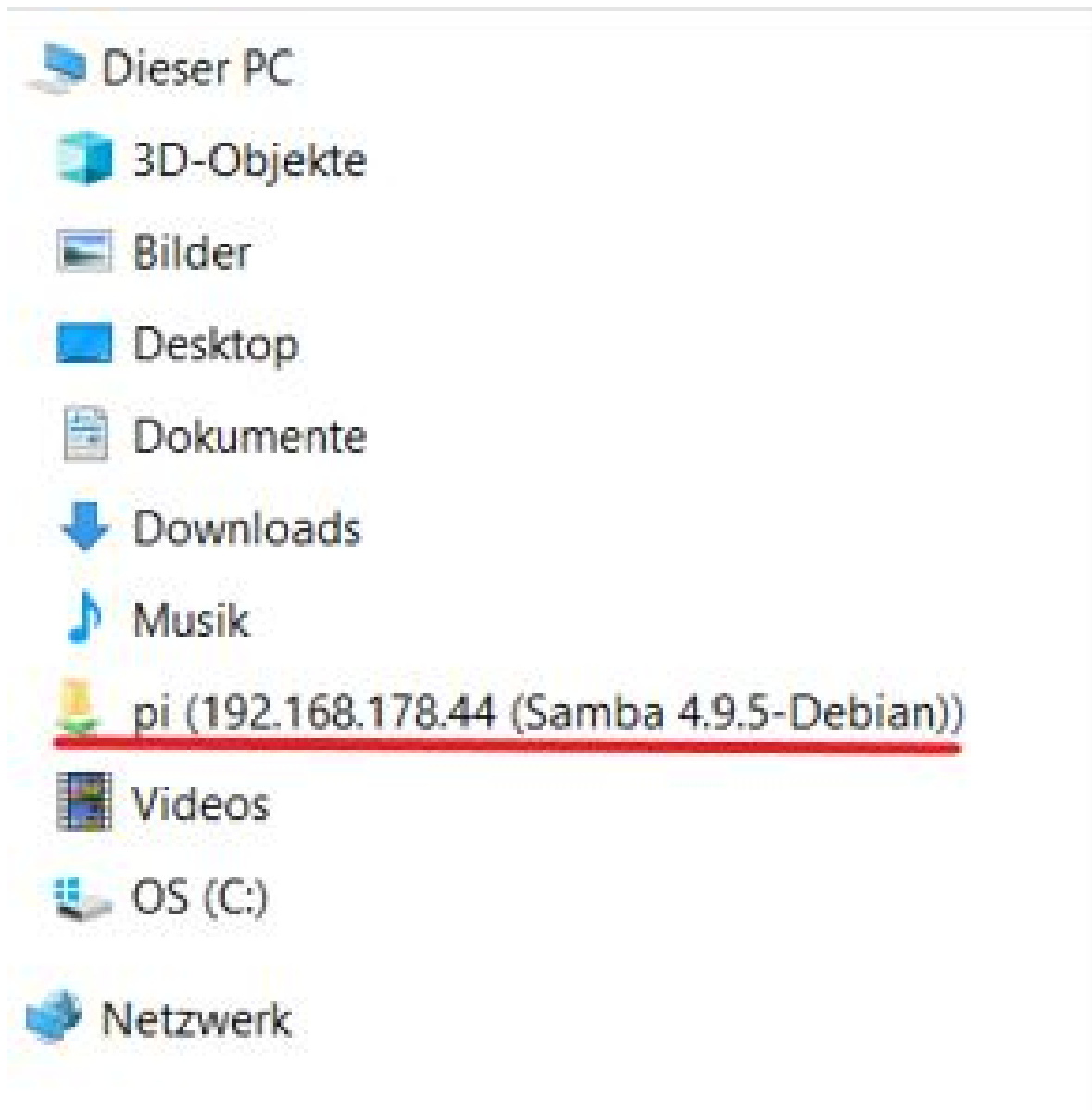


Abbildung 1: DotNet 6 Nullability von Objekten

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.

Docker auf dem Raspberry Pi installieren und die Verwendung von Docker auf dem Raspberry: Docker ist eine Software, welche das Management von Container 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. Weiter ist auch

ein Container mit dem Namen Portainer auf unserem Raspberry Pi installiert, welcher uns eine Liste aller Container und deren Informationen anzeigt.

Remote Access auf einen Raspberry PI: Wenn man auf einen Raspberry PI zugreifen möchte und man hat gerade keinen Monitor zu Verfügung, dann kann man mittels SSH von einem Laptop darauf zugreifen. Um mittels SSH auf einen Raspberry zugreifen zu können, muss man die Ip-Adresse in einem Netzwerk von dem Raspberry wissen. Falls man eine FRITZ!Box als WLAN-Router nutzt, kann man sich über die Ip-Adresse (192.168.178.1) im Browser auf seiner FRITZ!Box und eine Übersicht der verbundenen Geräte anzeigen lassen, wo auch unter anderem der Raspberry PI connected ist.

Was sind Ip-Adressen: Im oberen Kapitel Remote Access ist oft das Wort Ip-Adresse gefallen, deswegen wird in diesem Unterpunkt eine kleine Einführung was Ip-Adressen sind und wofür sie gebraucht werden. Eine Ip-Adresse ist eine Adresse in Computernetzen, welche von dem 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 Funktion, dass ein Gerät immer eine bestimmte Ip-Adresse zugewiesen bekommt.

3.2 MQTT

3.3 .Net

Projekte

Das Backend setzt sich aus 12 Projekten zusammen, welche in C# .Net 5 geschrieben sind.

- CommonBase

In CommonBase befinden Klassen und Methoden, die wiederverwendbar sind um Code verdoppelung zu vermeiden.

- CSharpCodeGenerator.Logic

CSharpCodeGenerator.Logic dient zum automatischen generieren von den Entitäten in SnQPoolIot.Logic, SnQPoolIot.WebApi und SnQPoolIot.AspMvc. In SnQPoolIot.Contracts werden Entitäten als Interfaces angegeben. Wenn danach die Solution

gebildet wird, werden für die Entitäten Klassen und die dazugehörigen Controller angelegt. Es werden auch Beziehung und Keys zwischen den Entitäten generiert.

- SnQPoolIot.Adapters

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

- SnQPoolIot.WebApi

Der Zugriff auf die Messwerte wird durch Rest-Zugriffe in SnQPoolIot.WebApi providet. Auf die Daten kann aber nur zugegriffen werden, wenn man sicher vorher mit einem Account einloggt.

- SnQPoolIot.Contracts

SnQPoolIot.Contracts beinhaltet alle notwendigen Schnittstellen und Enumerationen des Projektes. Hier werden die Entitäten als Interfaces angelegt, welche von CSharp-CodeGenerator.Logic als Klassen und Controller in den oben genannten Projekten generieren.

- SnQPoolIot.Logic

SnQPoolIot.Logic ist das Kernstück des Projektes. Durch die Logic können wir alle Daten von der Datenbank holen. Die Datenbank auf welche die Logik zugreift ist eine Sqlite Datenbank und der Zugriff und das Erzeugen der Datenbank wird mittels Entityframework.Sqlite durchgeführt.

- SnQPoolIot.Transfer

SnQPoolIot.Transfer verwaltet die Transferobjekte für den Datenaustausch zwischen den Layers.

- SnQPoolIot.AspMvc

SnQPoolIot.AspMvc ist ein Ersatz für das Frontend. Hier werden die Funktionen wie zum Beispiel 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.

.Net 5 vs .Net 6 Unser Backend wurde wie oben bereits erwähnt mit .Net 5 geschrieben, weil die Umstellung zu .Net 6 einiges mit sich bringt. Der Hauptgrund warum wir

uns für .Net 5 entschieden haben, ist die Unterscheidung der Nullability von Objekten und die dadurch ausgelösten Warnings. In .Net 5 werden durch Objekte die Null sein könnten keine Warnings angezeigt und man erspart sich etliche Zeit, wenn man sich nicht durch die Warnings käpfen muss.

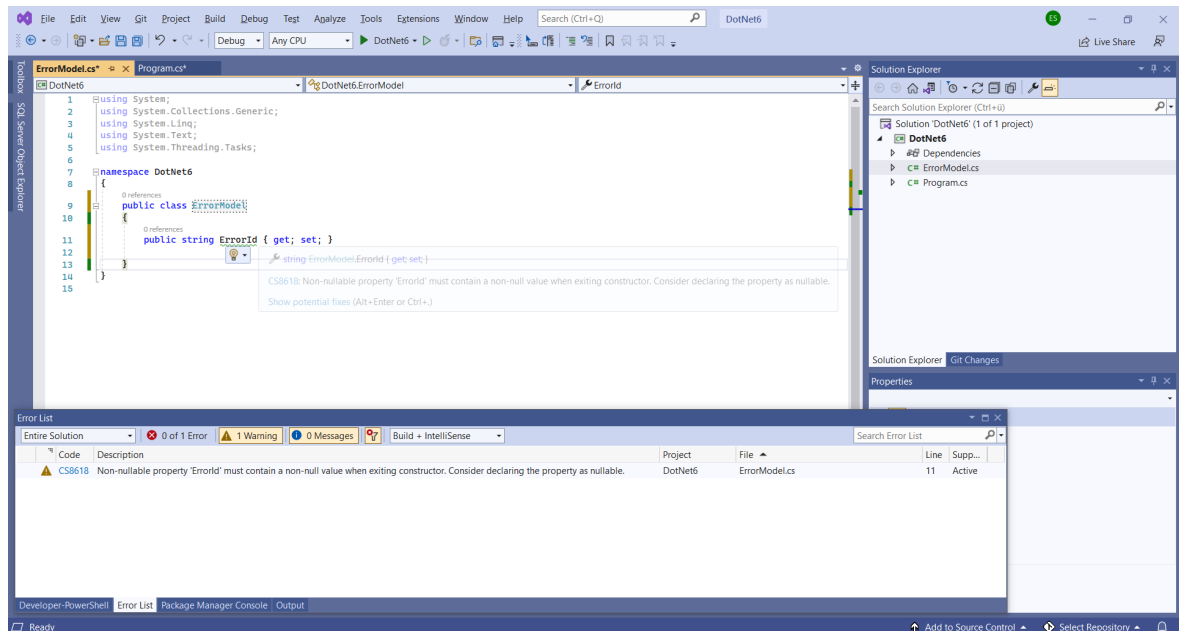


Abbildung 2: DotNet 6 Nullability von Objekten

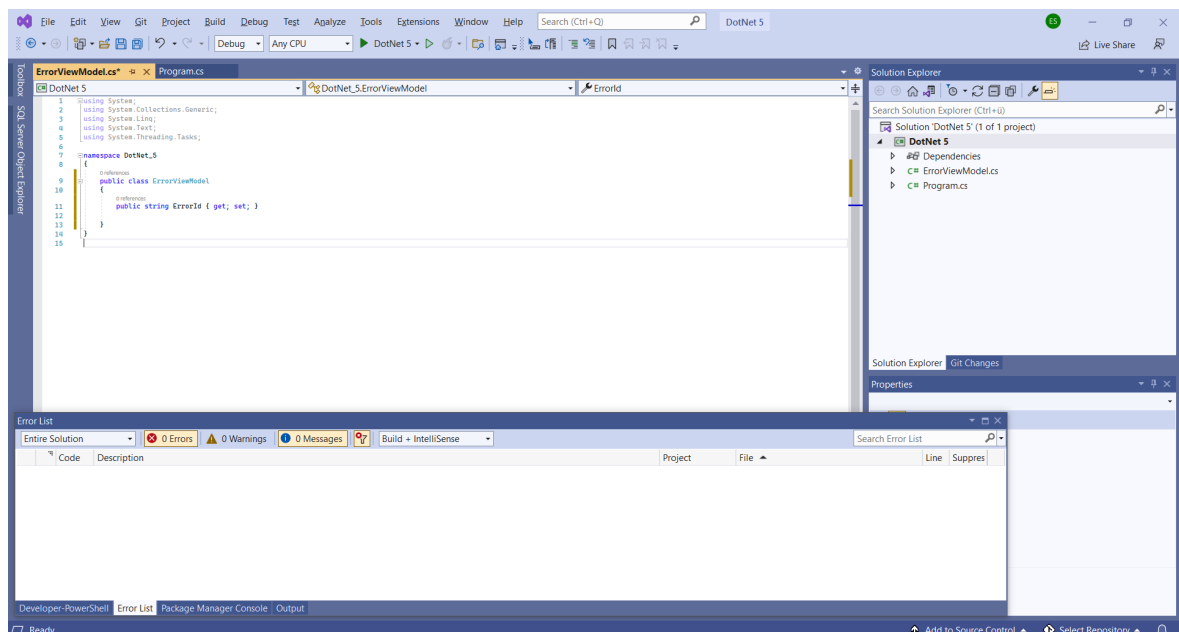
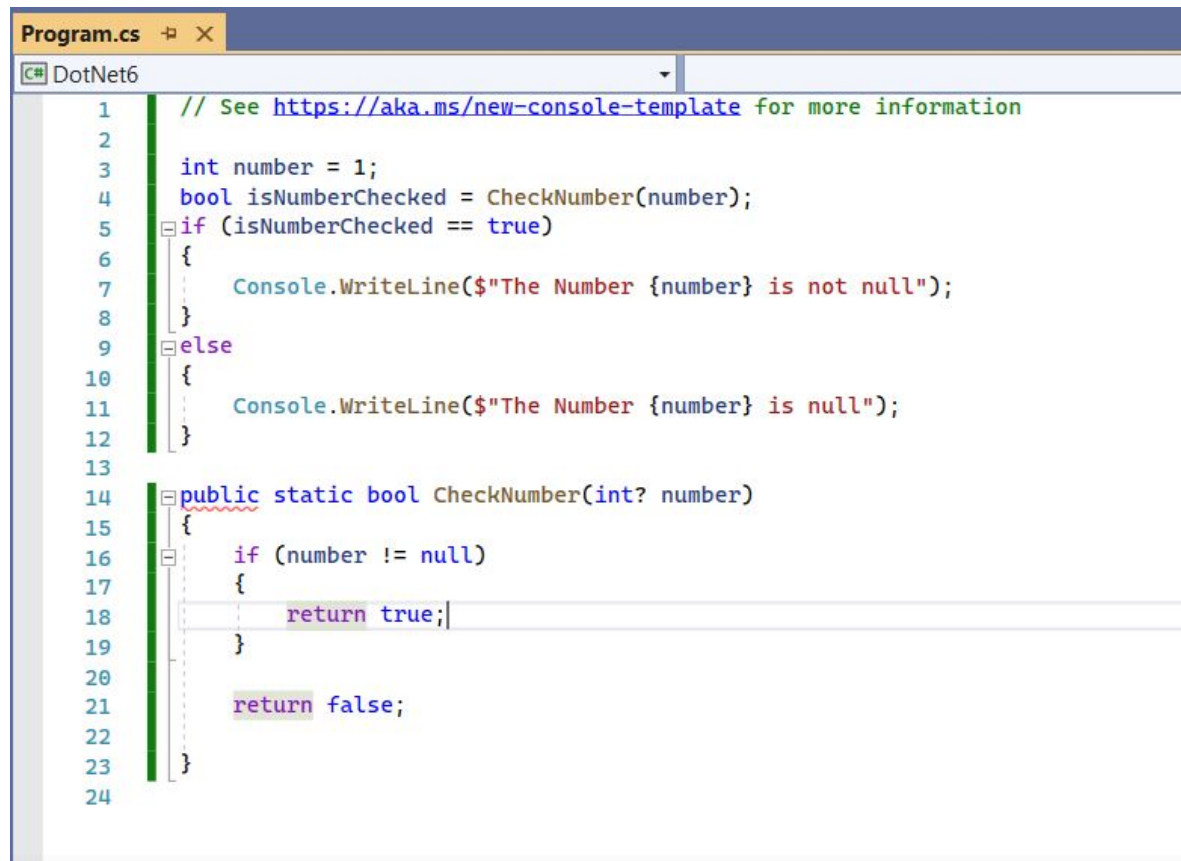


Abbildung 3: DotNet 5 keine Warnings durch Nullability

Jedoch im Vergleich zu .Net 5 hat .Net 6 eine bessere Leistung, denn bei einer Ausführung in der Cloud sinkt es die Computekosten, welches ich als riesigen Vorteil sehe. Einen weiteren Nachteil finde ich ist die Lesbarkeit des Codes.

Hier ein Beispiel zu diesem Thema:

Hier sieht man eine Consolen Application in DotNet 6 und man kann sehen, dass es keine Übersicht gibt wie man anfängt. Eine Person welche zum ersten Mal ein Tutorial, welches in DotNet 5 beschrieben wurde, nachprogrammiert, hat keine Ahnung wo man in DotNet 6 die Usings oder die Methoden implementiert.



```
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 4: DotNet6ConApp

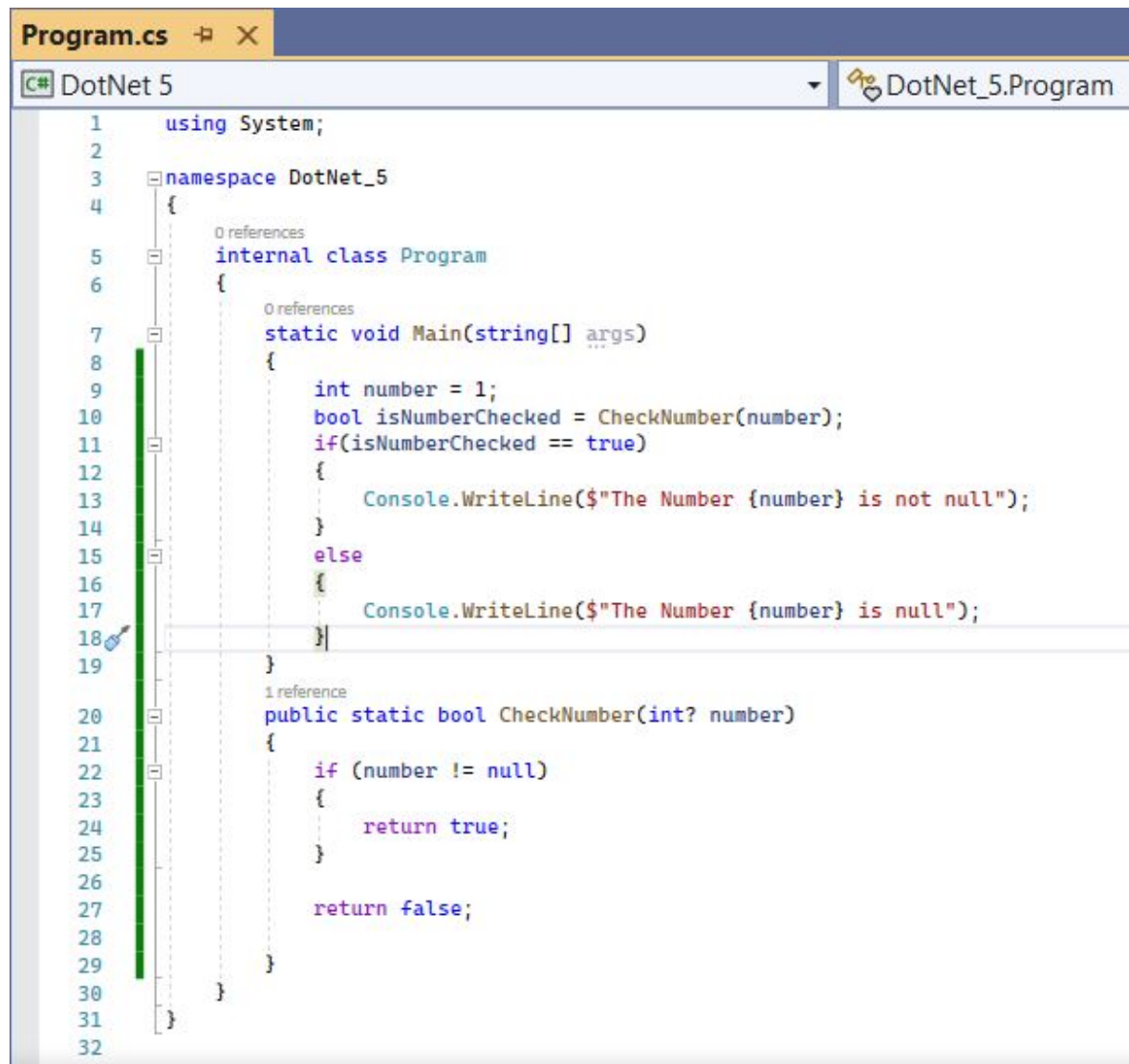


Abbildung 5: DotNet5ConApp

Im Vergleich zu DotNet 6 kann man hier klar erkennen, wo man die Usings und die Methoden hinschreibt.

Warum Sqlite und nicht SqlServer als Datenbank:

In unserem Projekt verwenden wir Sqlite als Datenbank, weil die Datenbank ansonsten nicht auf dem Raspberry Pi laufen würde. Unser Raspberry Pi verwendet Linux als Betriebssystem und Microsoft meint, dass ein SqlServer nicht für Linux geeignet sind und deshalb gibt es keinen SqlServer für Raspberry Pi's. Weiteres, wenn es einen SqlServer für den Raspberry Pi geben würde, wird wegen dem Speicherplatz noch immer eine Sqlite Datenbank einen großen Vorteil anbieten.

4 Zusammenfassung

Literaturverzeichnis

Abbildungsverzeichnis

1	DotNet 6 Nullability von Objekten	4
2	DotNet 6 Nullability von Objekten	7
3	DotNet 5 keine Warnungs durch Nullability	7
4	DotNet6ConApp	8
5	DotNet5ConApp	9

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang