

# **Embedded Frontend-Entwicklung mit .Net Core und Blazor**

William Mendat

## **BACHELORARBEIT**

zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

Studiengang Angewandte Informatik

Fakultät Elektrotechnik, Medizintechnik und Informatik  
Hochschule für Technik, Wirtschaft und Medien Offenburg

28.02.2022

Durchgeführt bei der Firma Junker Technologies

Betreuer

Prof. Dr.-Ing. Daniel Fischer, Hochschule Offenburg

M. Sc. Adrian Junker, Junker Technologies

**Mendat, William:**

Embedded Frontend-Entwicklung mit .Net Core und Blazor / William Mendat. –  
BACHELORARBEIT, Offenburg: Hochschule für Technik, Wirtschaft und Medien Offenburg, 2022. 8 Seiten.

**Mendat, William:**

Embedded Frontend-Development with .Net Core and Blazor / William Mendat. –  
BACHELOR THESIS, Offenburg: Offenburg University, 2022. 8 pages.

## Vorwort

—...—

## **Eidesstattliche Erklärung**

Hiermit versichere ich eidesstattlich, dass die vorliegende Bachelor-Thesis von mir selbstständig und ohne unerlaubte fremde Hilfe angefertigt worden ist, insbesondere, dass ich alle Stellen, die wörtlich oder annähernd wörtlich oder dem Gedanken nach aus Veröffentlichungen, unveröffentlichten Unterlagen und Gesprächen entnommen worden sind, als solche an den entsprechenden Stellen innerhalb der Arbeit durch Zitate kenntlich gemacht habe, wobei in den Zitaten jeweils der Umfang der entnommenen Originalzitate kenntlich gemacht wurde. Ich bin mir bewusst, dass eine falsche Versicherung rechtliche Folgen haben wird.

Ich bin damit einverstanden, dass meine Arbeit veröffentlicht wird, d. h. dass die Arbeit elektronisch gespeichert, in andere Formate konvertiert, auf den Servern der Hochschule Mannheim öffentlich zugänglich gemacht und über das Internet verbreitet werden darf.

Offenburg, 28.02.2022

William Mendat

# **Zusammenfassung**

## ***Embedded Frontend-Entwicklung mit .Net Core und Blazor***

In dieser Abschlussarbeit wird eine Net Core Blazor Anwendung implementiert, um somit einen Ersatz für die derzeit GUI-Entwicklung auf dem Raspberry Pi mit Qt zu kreieren. Dabei werden verschiedene Ansätze implementiert, sowohl auf dem Raspberry Pi als auch auf einem externen Server, um einen aussagekräftigen Vergleich zwischen den hier verschiedenen Technologien zu schaffen. Ein großer Teil dieser Ausarbeitung besteht darin, eine Laufzeitanalyse zu erstellen.

Die Arbeit gibt zunächst einen Einblick in den momentanen Stand der Technik, wie bislang mit einem Raspberry Pi gearbeitet wurde, um dann zu veranschaulichen, wie dass gleiche Verhalten mit Net Core und Blazor widergespiegelt werden kann. Am Ende dieser Arbeit wird ein aussagekräftiges Fazit darüber abgegeben, ob die Technologie Blazor für die Frontendentwicklung im Embedded Bereich zu gebrauchen ist.

## **Abstract**

### ***Embedded Frontend-Development with .Net Core and Blazor***

Englische Version von Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
1.1. Aufgabenstellung . . . . .	2
1.2. Verwendet Hardware . . . . .	2
1.3. Verwendete Software . . . . .	4
<b>2. Stand der Technik</b>	<b>5</b>
2.1. Embedded Systems . . . . .	5
2.1.1. Hardware . . . . .	6
2.1.2. Software . . . . .	7
2.2. Qt . . . . .	7
2.2.1. Was ist Qt? . . . . .	8
<b>Abkürzungsverzeichnis</b>	
<b>Tabellenverzeichnis</b>	<b>i</b>
<b>Abbildungsverzeichnis</b>	<b>ii</b>
<b>Quellcodeverzeichnis</b>	<b>iii</b>
<b>Literatur</b>	<b>iv</b>
<b>A. Ein Anhang</b>	<b>v</b>

# 1. Einleitung

Heutzutage ist die Menschheit darauf fokussiert, die komplette Welt zu digitalisieren. Dabei existiert ein Grundsatz, alles, was digitalisiert werden kann, soll digitalisiert werden. Um dies zu realisieren, ist es von Nöten, überall Hardware und Software zu verbinden. Sei es nun das Handy, mit welchem durch nur einem klick die Bankdaten angezeigt werden können, oder ein selbstfahrendes Auto, welches einen Anwender selbstständig zum Ziel fährt. Dies sind nur zwei Beispiele von einer unendlich langen Liste. Hinter diesen technischen Wundern stecken meist mehrere Tausend kleiner Mikrocomputern und Mikrocontrollern, die dann mittels Software zusammen interagieren. Die Kombination dieser zwei Komponenten werden durch den Oberbegriff „Embedded System“ oder auch zu Deutsch ein „Eingebettetes System“ definiert.

Embedded Systems können dabei grundsätzlich zwischen zwei Plattformen unterschieden werden:

- Deeply Embedded System
- Open Embedded System

Deeply Embedded Systems sind die wesentlichen Bausteine des Internet of Things [1]. Die Anwendung, die bei Deeply Embedded Systems implementiert wird, basiert auf speziell angepassten Echtzeitbetriebssystemen, den Programmiersprachen C oder C++ und ganz speziellen GUI<sup>1</sup>-Frameworks<sup>2</sup> wie zum Beispiel TouchGFX.

Anders als bei den Deeply Embedded Systems, die sehr auf speziellen Technologien aufbauen, bieten Open Embedded Systems eine höhere Flexibilität in Sachen Technologien an. Dem Programmierer ist also die Möglichkeit gegeben, unterschiedliche

---

<sup>1</sup>Graphical User Interface

<sup>2</sup>In der Softwareentwicklung ist ein Framework ein Entwicklungsrahmen, der dem Anwendungsprogrammierer zur Verfügung steht, um die grundlegende Architektur der Software zu bestimmen [2].



Technologien sowohl als auch unterschiedliche Programmiersprachen zu verwenden. Dort gilt bis dato die Kombination von C++ und Qt für GUI-lastige Systeme als „State of the Art“.

Die Konstellation zwischen C++ und Qt hat bislang auch funktioniert, jedoch kommt dieser Ansatz auch mit Problemen mit sich, denn die höheren Entwicklungszeiten für die Entwicklung von C++ Anwendungen sowie die geringe Verfügbarkeit von Experten auf dem Arbeitsmarkt sorgen für schlechtere Qualität und längere Produktionszeiten.

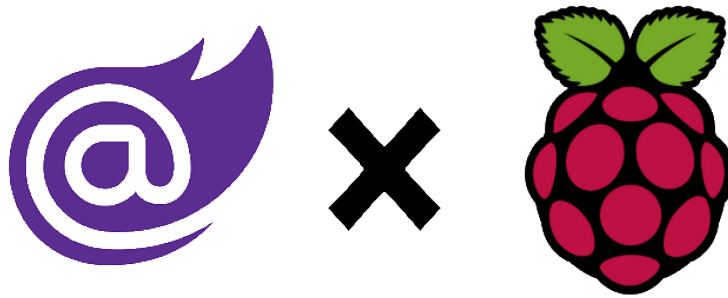
Um diesen Problemen zu entgegnen, soll in dieser Abschlussarbeit ein anderer Ansatz betrachtet werden. Und zwar könnten sowohl die Anwendungsschicht als auch die Persistenzschicht als .Net Core Anwendungen implementiert werden. Als GUI-Technologie soll dabei die neue Microsofttechnologie namens *Blazor* als Qt Ersatz zum Einsatz kommen. Somit kann erreicht werden, die komplette Codebasis mit .Net Core auszutauschen und eine Programmier-freundlichere Umgebung für Entwickler im Open Embedded Systems zu erschaffen.

### 1.1. Aufgabenstellung

Das Ziel dieser Arbeit ist die Entwicklung eines Blazor-basierten Frontend auf einem Raspberry Pi 4 um einen aussagekräftigen Vergleich zwischen den Technologien schaffen zu können und um eine mögliche Verdrängung mittels Blazor zu demonstrieren. Dazu soll zunächst begutachtet werden, wie der momentane Stand der Technik für Open Embedded Systems ist, um anschließend das gleiche Verhalten mittels Blazor zu reproduzieren. Insbesondere sollen dabei verschiedene Aspekte, wie zum Beispiel das Verhalten zur Laufzeit, dieses Ansatzes überprüft werden.

### 1.2. Verwendet Hardware

Als Zielplattform für diese Thesis dient ein Raspberry Pi 4 B. Dieses wird unter anderem deswegen verwendet, da es im Labor Embedded Systems 2 der Hochschule Offenburg verwendet wird, aber auch, da es sehr gut im Open Embedded Systems



**Abbildung 1.1:** Blazor mit Raspberry Pi

bereich eingesetzt werden kann. Der Raspberry Pi 4 B verfügt dabei, unter anderem, über die folgenden technischen Spezifikationen: [3]

- 1,5 GHz ARM Cortex-A72 Quad-Core-CPU
- 1 GB, 2 GB oder 4 GB LPDDR4 SDRAM
- Gigabit LAN RJ45 (bis zu 1000 Mbit)
- Bluetooth 5.0
- 2x USB 2.0 / 2x USB 3.0
- 2x microHDMI (1x 4k @60fps oder 2x 4k @30fps)
- 5V/3A @ USB Typ-C
- 40 GPIO Pins
- Mikro SD-Karten slot

Und wird mit dem *Raspbian Buster with desktop* auf der SD-Karte betrieben.

Um noch mehr Funktionalität aufbringen zu können, wurde auf dem Raspberry Pi ein *RPI SENSE HAT Shield* aufgesteckt. Mit hilfe des *RPI SENSE HAT Shield* können dann unter anderem Daten wie zum Beispiel die momentane Temperatur oder auch die momentane Luftfeuchtigkeit gewonnen werden. Zudem ist auf dem SENSE HAT noch eine 8x8 LED-Matrix enthalten und ein Joystick mit 5 knöpfen, die angesteuert werden können.

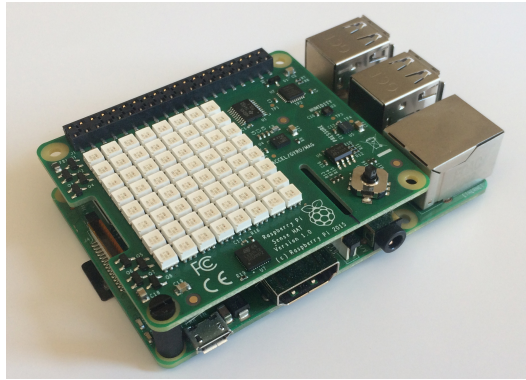


Abbildung 1.2: Verwendeter Raspberry Pi

### 1.3. Verwendete Software

Im Rahmen dieser Thesis werden die folgenden Softwaretools zur Entwicklung eingesetzt.

- Um eine Visualisierung des Images *Raspbian Buster with desktop* von dem Raspberry Pi zu erhalten, wird die Windows Desktop Anwendung *VNC Viewer* verwendet. Dadurch ist die Möglichkeit gegeben, bequem und einfach den Raspberry Pi über einen Bildschirm zu bedienen.
- Für die Demonstration des Kapitels *Stand der Technik* wird eine beispielhafte grafische Oberfläche mithilfe des Qt-Creators implementiert.
- Den Zugriff auf die Funktionalitäten des *RPI SENSE HAT Shield*, wird mittels der *RTIMULib* Bibliothek realisiert.
- Eine ausschlaggebende Technologie dieser Thesis wird durch das Framework *Blazor* von Microsoft abgebildet.
- Die Programmiersprachen dieser Thesis werden sich hauptsächlich aus C++ und C# beziehungsweise .Net zusammensetzen.
- Um den Zugriff auf die Funktionalitäten des *RPI SENSE HAT Shield* mittels .Net zu gewährleisten, wird die *IoT Bibliothek* von Microsoft verwendet.
- Die Implementierung der *Blazor* Anwendung wird dann letztendlich mittels der kostenlosen IDE *Visual Studio Code* realisiert.

Die oben vorgestellten Tools und Programmiersprachen wurden nicht explizit vorgegeben, sind dementsprechend auch selbst ausgesucht und sind zu Beginn dieser Thesis auch schon alle komplett eingerichtet.

## 2. Stand der Technik

Dieses Kapitel soll ein grundlegendes Verständnis wiedergeben, wie der momentane Stand der Technik aussieht. Dabei soll zunächst betrachtet werden, wie die momentane Entwicklung bei eingebetteten Systeme vonstattengeht, um anschließend eine beispielhafte Anwendung zu implementieren. Zudem soll auch das Framework dieser Thesis vorgestellt werden.

### 2.1. Embedded Systems

Ein Embedded System oder auf Deutsch ein eingebettetes System, wird als eine integrierte, mikroelektronische Steuerung angesehen, welches meist darauf ausgelegt ist eine spezifische Aufgabe zu erledigen. Dabei setzt sich ein eingebettetes System auch aus dem Zusammenspiel zwischen Hardware und Software zusammen. Solche eingebetteten Systeme haben meist kein ausgeprägtes Benutzerinterface und können weitergehend in die zwei unterklassen, Open und Deeply Embedded Systems unterteilt werden.

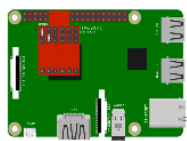
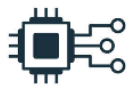
Open Embedded	Deeply Embedded
	
<b>32/64 Bit Multicore Prozessor</b>	<b>8/16 Bit Singlecore Prozessor</b>
<b>Systemsoftware Applikation</b>	<b>Monitorprogramm Applikation</b>
<b>Für komplexe Aufgaben gedacht</b>	<b>Für einfache Aufgaben gedacht</b>

Abbildung 2.1: Open vs Deeply Embedded Systeme [4][vgl.]

Neben der logischen Korrektheit die eingebettete Systeme an den Tag legen müssen, lassen sie sich durch eine Reihe unterschiedlicher Anforderungen und eigenschaften von den heutzutage üblichen Anwendungen abgrenzen. Unter anderem wird bei eingebettete Systeme ein sogenanntes *Instant on* gefordert, welches besagt, dass das Gerät unmittelbar nach dem Einschalten betriebsbereit sein muss [4][vgl.].

Weitere Anforderungen die bei Embedded Systemen zustande kommen, sind in der Folgenden Tabelle abgebildet:

Anforderung	Beschreibung
Funktionalität	Die Software muss schnell und korrekt sein
Preis	Die Hardware darf nicht zu kostenspielig sein
Robustheit	Muss auch in einem rauen Umfeld funktionieren
Fast poweroff	Muss in der Lage sein schnell das komplette System Abzuschalten
Räumliche Ausmaße	Muss klein sein, um sich in ein System einbinden zu können
Nonstop-Betrieb	Muss in der Lage sein, im Dauerbetrieb laufen zu können
Lange Lebensdauer	Muss in der Lage sein, mehr als 30 Jahre zu laufen ohne groß zu verschlechtern

**Tabelle 2.1.:** Anforderungen an eingebettete Systeme [4]

### 2.1.1. Hardware

Die einzelnen Komponenten, die in einem eingebetteten System verbaut worden sind, entscheiden über die vorhandene Leistung, den Stromverbrauch und die Robustheit, die dieses System ausmachen. Die kern Komponente eines eingebetteten Systems wird durch einen Prozessor repräsentiert und werden häufig als *System on Chip* eingebaut. Dabei ist die Tendenz zu den ARM-Core Modellen steigend. Neben dem Prozessor befinden sich typischerweise auf den eingebetteten Systemen noch weitere Komponenten, wie zum Beispiel dem Hauptspeicher, dem persistenten Speicher und diversen Schnittstellen, um weitere peripherie Geräte anzuschließen [4][vgl.].

Damit weitere peripherie Geräte angeschlossen werden können, müssen mittels einigen Leitungen, digitale Signale übertragen werden. Aufgrund dessen, dass Leitungen typischerweise nur über eine begrenzte Leistung verfügen, werden Treiber

eingesetzt, um periphere Geräte zu verbinden. Nicht selten kommt es vor, dass in einem eingebetteten System darüber hinaus noch eine galvanische Entkopplung<sup>1</sup> eingebaut wird, damit die Hardware vor Störungen von außen, wie Beispiel Motoren, geschützt ist [4][vgl.].

### 2.1.2. Software

Genauso, wie sich eingebettete Systeme in zwei Bereiche unterscheiden lassen können, kann die Software eines eingebetteten Systems in Systemsoftware und funktionsbestimmende Anwendungssoftware unterteilt werden. Da *Deeply Embedded Systems* meist nur für kleine und einfache Aufgaben ausgelegt sind, benötigen diese meist nur sehr schwache Software. Diese funktionsbestimmende Anwendungssoftware ist dann meist ein spezielles Echtzeitbetriebssystem, welche auf die Aufgabe und der Hardware angepasst ist.

Ganz anders sieht es im *Open Embedded Systems* Bereich aus, welche seine Software als Kennzeichen hat. Da diese für sehr komplexe Aufgaben zum Einsatz kommen, kommt es nicht selten vor, dass auch eine GUI für ein solches System vonnöten ist. Deswegen basiert ein *Open Embedded Systems* auf einer Systemsoftware, die dem Programmierer mehr Möglichkeiten beim Entwickeln gibt. Unter anderem ist somit auch die Möglichkeit gegeben, die Programmiersprache und das GUI-Framework, nach Belieben selbst auszusuchen und nicht auf spezielle Software angewiesen zu sein [4][vgl.].

## 2.2. Qt

In der vorherigen Sektion wurde ein allgemeines Bild dargestellt, was ein eingebettetes System ist und in welchen Varianten diese auftauchen. Weitergehend soll nun, in dieser Sektion, veranschaulicht werden, wie die Programmierung im *Open Embedded Systems* Bereich vonstattengeht.

Ein großer Anteil eines *Open Embedded Systems* wird heutzutage durch die GUI repräsentiert. Die GUI sollte intuitive und zuverlässig sein. Zudem ist es auch noch

---

<sup>1</sup>Unter der galvanischen Entkopplung versteht man das Vermeiden der elektrischen Leitung zwischen zwei Stromkreisen, zwischen denen Leistung oder Signale ausgetauscht werden sollen [5]

enorm wichtig, dass die GUI wenige Ressourcen verbraucht, schnell reagiert und einfach einzubinden ist. Um diese Anforderung zu erreichen, wurde bis dato *Qt* in Kombination mit der Programmiersprache *C++* verwendet [6][vgl.].

### 2.2.1. Was ist Qt?

Test

# Abkürzungsverzeichnis

<b>GUI</b>	Graphical User Interface . . . . .	1
------------	------------------------------------	---



# Tabellenverzeichnis

2.1. Anforderungen an eingebettete Systeme . . . . .	6
A.1. Tabellenunterschrift . . . . .	v

# Abbildungsverzeichnis

1.1. Blazor mit Raspberry Pi . . . . .	3
1.2. Raspberry Pi 4 B . . . . .	4
2.1. Open vs Deeply Embedded Systeme . . . . .	5
A.1. Beschreibung für Verzeichnis2 . . . . .	vi

# Listings

# Literatur

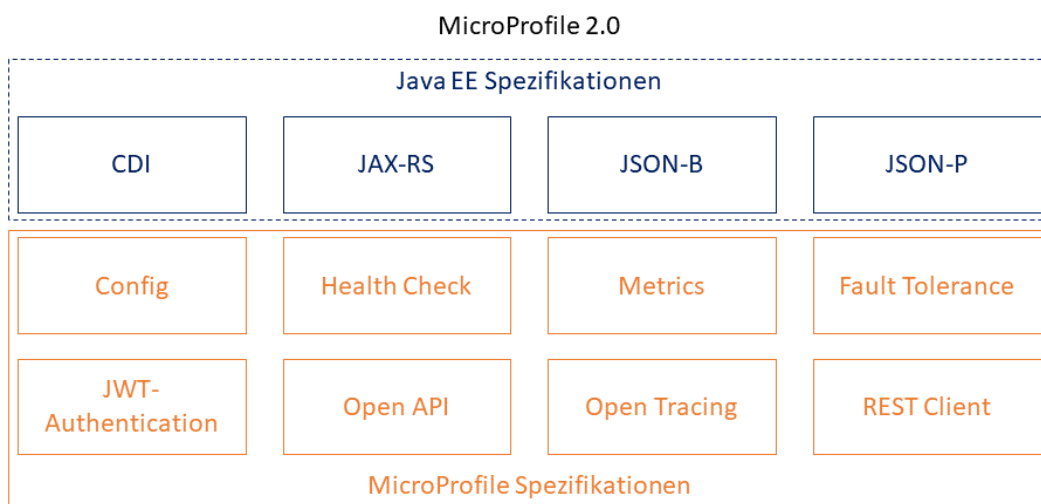
- [1] Hochschule niederrhein, *CAS Embedded Systems Professional - Hochschule Niederrhein*, 29.10.2021. Adresse:  
<https://www.hs-niederrhein.de/weiterbildung/sichere-software/cas-embedded-systems-professional/>.
- [2] Business Systemhaus AG, *Was ist ein Framework? Definition & Erklärung - BSH AG*, 16.11.2021. Adresse:  
<https://www.bsh-ag.de/it-wissensdatenbank/framework/>.
- [3] *Raspberry Pi 4 Model B specifications – Raspberry Pi*, 5.11.2021. Adresse:  
<https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>.
- [4] J. Quade, *Embedded Linux lernen mit dem Raspberry Pi: Linux-Systeme selber bauen und programmieren*, 1. Auflage. Heidelberg: dpunkt.verlag, 2014, ISBN: 9783864915093. Adresse:  
[http://ebooks.ciando.com/book/index.cfm/bok\\_id/1423957](http://ebooks.ciando.com/book/index.cfm/bok_id/1423957).
- [5] Breimer-Roth Transformatoren, *Galvanische Trennung: Wissenswertes zur galvanischen Trennung bei Breimer-Roth*, 10.05.2021. Adresse:  
<https://bre-trafo.de/wissen/galvanische-trennung/>.
- [6] Q. Jinhui, L. D. Hui und Y. Junchao, „The Application of Qt/Embedded on Embedded Linux“, in *2012 International Conference on Industrial Control and Electronics Engineering*, 2012, S. 1304–1307. DOI: 10.1109/ICICEE.2012.346.

## A. Ein Anhang

Referenz zu Tabelle A.1.

Bezeichnung	Typ	Beschreibung
load.load1	float	The load average over 1 minute.
load.load5	float	The load average over 5 minutes.
load.load15	float	The load average over 15 minutes.
cpu.user	int	The amount of CPU time spent in user space.
cpu.user_p	float	The percentage of CPU time spent in user space. On multi-core systems, you can have percentages that are greater than 100%. For example, if 3 cores are at 60% use, then the cpu.user_p will be 180%.
cpu.system	int	The amount of CPU time spent in kernel space.
cpu.system_p	float	The percentage of CPU time spent in kernel space.
mem.total	int	Total memory.
mem.used	int	Used memory.
mem.free	int	Available memory.
mem.used_p	float	The percentage of used memory.

**Tabelle A.1.:** Tabellenunterschrift



**Abbildung A.1:** Bildunterschrift2