Abschlussprüfung Winter 2021

Fachinformatiker für
Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Entwicklung einer allgemeinen Ressourcenverwaltung

für Diagnostikgeräte via gRPC

Prüfungsbewerber:

Lukas Klettke

Am Mühlenteich 17

23611 Bad Schwartau

11. November 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

RPC	Remote Procedure Call
gRPC	Open-Source „Remote Procedure Call“ System, entwickelt von Google
SoC	System-on-a-Chip
WDDM	Windows Display Driver Model
NuGet	Paketmanager basierend auf dem .NET Framework
Prism	Application Framework für WPF
WPF	Windows Presentation Foundation
IoC	Inversion of Control - Programmierprinzip
MVVM	Model View ViewModel
GitLab	Webanwendung zur Versionsverwaltung auf Git-Basis
HTTP2	größere Überarbeitung des HTTP-Netzwerkprotokolls

1 Einleitung

1.1 Vorstellung der eigenen Person

Mein Name ist Lukas Klettke. Ich bin am 14.01.2001 in Lübeck geboren und in Bad Schwartau aufgewachsen. Dort habe ich die Grundschule und das Leibniz Gymnasium besucht. Nach zwölf Jahren Schulzeit habe ich meinen Schulweg im Jahre 2019 mit dem Abitur abgeschlossen.

Direkt nach Abschluss der Schule habe ich im August 2019 eine Ausbildung zum Fachinformatiker für Anwendungsentwicklung begonnen und bin in der Softwareentwicklung für Diagnostikgeräte tätig.

In meiner Freizeit bin ich als ehrenamtlicher Schwimmtrainer tätig, segle und fahre Rennrad.

1.2 Vorstellung des Ausbildungsbetriebs

Mein Ausbildungsbetrieb ist die EUROIMMUN Medizinische Labordiagnostika AG mit Hauptsitz in 23560 Lübeck und Zweigstellen in Groß Grönau, Selmsdorf und Dassow im Norden und Rennersdorf, Pegnitz und Bernstadt im Süden Deutschlands. Durch den Verkauf der Firma im Dezember 2017 befindet sich EUROIMMUN in Besitz von PerkinElmer Inc., einem US-amerikanischen Technologieunternehmen im Bereich der Chemie- und Medizintechnik.

EUROIMMUN ist ein Hersteller für diverse medizinische Diagnostika von Autoimmun-, Infektionskrankheiten und Allergien, aber auch im Bereich der Automatisierung. Meine Ausbildung findet in Dassow in der Forschung und Entwicklung von Software zur Steuerung von Diagnostikautomaten statt.

Insgesamt hat EUROIMMUN mehr als 3.200 Mitarbeiter in 17 Ländern.

1.3 Projektauslöser

Neben der Herstellung von medizinischen Diagnostika zur manuellen Durchführung werden Geräte zur automatisierten Durchführung dessen hergestellt und vertrieben. Diese Diagnostikautomaten arbeiten mit diversen unterschiedlichen Betriebsmitteln (z.B. Reinigungsflüssigkeit zur Reinigung der Schläuche, Probenträger etc.), welche ebenfalls von EUROIMMUN an die Kunden verkauft werden.

Anhand der verbrauchten Betriebsmittel der Geräte wird die Menge, der in Zukunft benötigten, berechnet und die Preise dementsprechend auf den Kunden angepasst. Außerdem wird je nach Verbrauch der Labore die Produktions- und Lagermenge optimiert.

Derzeit werden jedoch keine Daten der Verbräuche von den Gerätesoftwares erhoben, was eine manuelle Berechnung derer zur Folge hat. Diese Berechnung wird durch Außendienstmitarbeiter durchgeführt, welche die Labore besuchen und die durchgeführten Testmengen als Maßstab nutzen. Je nach Art des Gerätebetriebs ist der Verbrauch jedoch unterschiedlich: Werden z.B. 500 Tests am Stück durchgeführt, ist das Verhalten des Geräts ein Anderes, als wenn über eine Zeit von zwei Wochen 500 Tests durchgeführt werden. Somit ist bei der Berechnung eine gewisse Ungenauigkeit vorhanden, die in Zusammenhang mit einer großen Anzahl an Kunden ebenfalls große Differenzen zwischen berechneten und reellen Verbräuchen verursacht.

Durch eine automatisierte und genauere Berechnung mithilfe von protokollierten Ressourcenverbräuchen könnten Punkte wie Preisgestaltung, Produktionsmenge oder Lagerhaltung weiter optimiert und somit Geld eingespart bzw. Gewinn maximiert werden.

1.4 Projektumfeld

Das Projektumfeld ist der EUROIMMUN Standort in Dassow. Dort befindet sich ein Teil der Entwicklung der Diagnostikgeräte und der zugehörigen Software.

Bei diesem Projekt handelt es sich um eine Software, die ausschließlich intern eingesetzt werden soll.

Der Entwicklungsstandard für die diversen Softwares zur Steuerung der Diagnostikgeräte ist die Programmiersprache C# in Verbindung mit dem .NET Framework.

1.5 Projektziel

Ziel des Projekts ist es eine Schnittstelle zu definieren, die unabhängig von Diagnostikgerät und der entsprechenden Software implementiert werden kann. Durch diese Schnittstelle wird definiert, in welcher Form die Verbrauchsdaten abgefragt werden.

Anhand dessen wird eine Software geschrieben, die die Ressourcenverbräuche verarbeitet, eine Gesamtberechnung durchführt und den Export einer „.xlsx“ ermöglicht, um die nachstehende Kalkulation mittels Microsoft Excel zu gewährleisten.

Die Planung eines solchen Projekts existiert bereits mehrere Jahre und wurde von der Geschäftsführung in Auftrag gegeben. Ziel ist es Daten über die Nutzung der EUROIMMUN Diagnostikgeräte zur erheben, welche zur Analyse von weiteren Optimierungmöglichkeiten dienen.

1.6 Projektschnittstellen

Das Projekt stellt eine gRPC Schnittstelle bereit. Diese wird mithilfe einer „.proto“ (s. Listing A.1) Datei definiert. Die Schnittstelle wird auf Seite des Clients implementiert und auf Seite des Servers zur Einbindung bereit gestellt, sodass die unterschiedlichen Softwares der Diagnostikgeräte diese implementieren können und die Freiheit haben, je nach Architektur und Speicherung, die Daten bereitzustellen.

Des Weiteren wird der Export einer „.xlsx“ Datei angeboten.

2 Projektplanung

2.1 Projektphasen

Tabelle 1 zeigt die vorgesehenen Phase des Projektes.

Projektphase	Geplante Zeit
Projektplanung	7 h
User-Interface Grundfunktionen	5 h
Implementierung der Anwendungslogik	40 h
Testen und Nacharbeiten	10 h
Dokumentation	8 h
Gesamt	**70 h**

Tabelle 1: Zeitplanung

2.2 Ist-Analyse

Zur Zeit besuchen Außendienstmitarbeiter Labore, die Diagnostikgeräte von EUROIMMUN verwenden, um die Ressourcenverbräuche zu ermitteln. Um weiterhin genügend Betriebsmittel vorzuhalten werden retrospektiv die Verbräuche entsprechend zur Zeit berechnet und die weitere Versorgung sicher gestellt. Diese Berechnung geschieht anhand der Anzahl durchschnittlich durchgeführter Tests und Aussagen der Labormitarbeiter, falls absehbar ist, dass zukünftig vom Durchschnitt abgewichen wird.

Diese Arbeitsweise erfordert gut geschulte Mitarbeiter.

2.3 Soll-Konzept

Durch die neu entwickelte Software soll es den Außendienstmitarbeitern möglich sein, die Ressourcenverbräuche genauer zu berechnen, den zukünftigen Bedarf präziser zu planen, somit die Preisgestaltung anzupassen und die Produktion und Lagerhaltung zu optimieren. Der Mitarbeiter kann sich mit einem Windows PC in das lokale Netzwerk des Labors einwählen und über die gegebene Schnittstelle eine Verbindung mit den Computern der Diagnostikautomaten aufbauen. Mithilfe dieser Verbindung werden die Verbräuche der Betriebsmittel abgefragt.

Das Programm soll für die Möglichkeit auf internationale Anwendung in Englisch geschrieben werden und auf einem Windows 7/8/10/11 PC laufen. Die Software wird generisch entwickelt, sodass der zukünftige Ausbau mithilfe anderer Technologien möglich ist. Sowohl die Seite des Diagnostikgeräts, als auch die des Außendienstmitarbeiters werden in C# entwickelt.

2.4 „Make or Buy“

Da der Anwendungsfall spezifisch für von EUROIMMUN entwickelte Diagnostikgeräte gilt, gibt es keine käufliche Software, die den Ansprüchen gerecht wird, weshalb sich zwangsläufig die Entwicklung einer eigenen Lösung ergibt.

2.5 Kosten- und Ablaufplanung

2.5.1 Wirtschaftlichkeitsprüfung

Die Wirtschaftlichkeit wird in den beiden folgenden Punkten genau beschrieben.

2.5.2 Projektkosten

Die Projektkosten werden mit einigen variablen Parametern betrachtet, da echte Daten seitens der Geschäftsleitung nicht herausgegeben werden. Aus diesem Grund wird mit fiktiven Stundensätzen gearbeitet, welche mit meinem Ausbilder abgestimmt wurden. Der Stundensatz eines Auszubildenden wird mit 75 EUR und der eines Mitarbeiters mit 100 EUR angesetzt. In diesen Stundensätzen sind neben Gehaltszahlungen Kostenbeiträge wie Lohnnebenkosten und Sozialbeiträge enthalten. Die im Unternehmen typischen Gemeinkosten, wie Miete, Reinigungskosten der Räumlichkeiten oder Abschreibungen auf das technische Equipment werden zusätzlich mit 15 EUR/Stunde angesetzt. Das Projekt wurde mit 70 Stunden angesetzt, woraus sich ein Gesamtbudget von 6.300 EUR ergibt (s. Tabelle 2).

Vorgang	Zeit	Kosten pro Stunde	Kosten
Entwicklungskosten	70 h	75 €	5250 €
Gemeinkosten	70 h	15 €	1050 €
Gesamt			**6300 €**

Tabelle 2: Kostenaufstellung

Kosten für die genutzte Hardware werden nicht angesetzt, da davon ausgegangen wird, dass diese schon vorhanden ist.

2.5.3 Amortisationsdauer

Die Amortisation des Projekts ist nicht das primäre Ziel. Wie oben genannt dient es zur Analyse von Daten in Bezug auf das Nutzungsverhalten der Kunden und zukünftige Optimierungen. Aus diesem Grund ist keine Amortisationsdauer zu berechnen.

Durch die hohe Anzahl an genutzten Diagnostikgeräten von EUROIMMUN in Laboren auf der ganzen Welt, werden die Projektkosten fiktiv auf die installierten Geräte umgelegt. Es werden 5.000 verkaufte Geräte weltweit veranschlagt, woraus sich ca. 1,25 EUR Kosten pro Gerät ergeben. Vor diesem Hintergrund erscheint es plausibel, dass sich dieses Projekt durch effizientere Außendienstbesuche innerhalb kurzer Zeit amortisiert.

2.6 Qualitätsanforderungen

Da die Anwendung unter anderem die Kosten für Mitarbeiterschulungen senken soll, muss sie einfach und intuitiv zu bedienen sein. Bei geringfügigen Fehlern soll die Software ausfallsicher sein und durch die Abfrage von unbestimmten Größen an Datenmengen soll die Kommunikation performant und schlank sein.

Die Bereitstellung einer frei implementierbaren Schnittstelle verlangt eine gute Dokumentation und Kommentierung des Quellcodes, sodass Entwickler diese Schnittstelle einfach nutzen können. Da die Softwares zur Steuerung der Diagnostikgeräte für Windows entwickelt werden, ist es nötig die Schnittstelle ebenfalls für Windows zu entwickeln und daraus resultierend auch die Software zur Abfrage der Betriebsmittelverbräuche.

3 Entwurfsphase

3.1 Zielplattform

Für die 64 Bit Version von Windows 10 wird ein Prozessor mit mindestens 1 GHz Arbeitsleistung oder ein SoC benötigt. Die Mindestkapazität des RAM liegt bei 2 GB, der Festplattenspeicher muss mindestens 32 GB groß sein. Die Grafikkarte muss über DirectX 9 oder höher mit einem WDDM 1.0 Treiber verfügen.

3.2 Qualitätssicherung

Zur Sicherung der Qualität wird die Zuverlässigkeit der Software getestet. Beide Seiten der Schnittstelle müssen zu jeder Zeit erreichbar sein und auf Anfragen reagieren bzw. Anfragen senden können. Gleichzeitig muss in Fehlerfällen reagiert werden und eine weitere Bereitstellung der Dienste gesichert sein. Zur Gewährleistung dessen wird die Software im fertigen Zustand Stresstests mit großen Datenmengen und absichtlich verursachten Fehlerfällen ausgesetzt.

Zur Gewährleistung der Einsatzbereitschaft ist es erforderlich, die einzubindenden Bibliotheken eindeutig zu dokumentieren um die Menge der möglichen Implementierungsfehler so klein wie möglich zu halten.

4 Realisierung

4.1 Eingesetzte Technologien

Für die Entwicklung der Schnittstelle, mit der die Verbräuche der Betriebsmittel der Diagnostikgeräte abgefragt werden, wird gRPC verwendet. gRPC ist ein Open-Source RPC System, welches von Google entwickelt wird.

RPC ist eine Technologie, die es möglich macht, Prozeduren auf anderen Geräte auszuführen, als auf dem, wo es aufgerufen wird (meistens innerhalb eines Netzwerks). Durch diese Auslagerung kann Datenverarbeitung auf andere Geräte ausgelagert werden, ohne dass ein Unterschied in der Entwicklung entsteht. Daraus ergibt sich eine Form der Client-Server-Architektur, bei der der aufrufende Part den Client und der ausführende Part den Server darstellt. Die Kommunikation basiert auf dem „request-response“ Protokoll, welche synchron via http abläuft. Schickt der Client eine Abfrage, ist er während der Bearbeitung durch den Server blockiert[1].

Die Wahl der Technologie fiel auf gRPC, da dieses bereits in der Firma genutzt wurde und somit eine Vorgabe darstellte.

Die Anwendung wird mit C# entwickelt. Dafür wird die Version 4.8 des .NET Frameworks verwendet.

Zum exportieren von „.xlsx“ Dateien wird das NuGet-Paket „Microsoft.Office.Interop.Excel“ verwendet. Die Dokumentation befindet sich auf der Microsoft Docs Webseite[2] des Frameworks.

Als Hilfe zur Implementierung wird Prism eingesetzt. Durch die Nutzung ergibt sich eine einfach Einbindung von IoC. Mithilfe von IoC können Konstruktoren automatisch initialisiert werden, woraus sich eine deutlich erhöhte Übersichtlichkeit des Programmcodes ergibt.

4.2 Entwicklungsumgebung

Zur Entwicklung wird Visual Studio Professional 2019 verwendet. Die IDE wurde durch die ReSharper Extension von Jetbrains erweitert. ReSharper fügt Tools für Refactoring und das Erkennen von Code Smells[3] hinzu. So wird gewährleistet, dass sauberer und qualitativ hochwertiger Quellcode produziert wird.

1 Wikipedia: https://en.wikipedia.org/wiki/Remote_procedure_call (Stand 08.11.2021 10:15 Uhr)
2 Microsoft.Office.Interop.Excel: https://docs.microsoft.com/en-us/dotnet/api/microsoft.office.interop.excel?view=excel-pia
3 Wikipedia: https://de.wikipedia.org/wiki/Code-Smell (Stand 10.11.2021 12:15 Uhr)

4.3 Erstellung einer Benutzeroberfläche

Die Oberfläche wurde mit WPF entworfen. Als Architekturmodell wird ein Hybrid aus zwei MVVM Architekturen angewendet. Die einzelnen Bestandteile (im Folgenden „Views“ genannt) verwenden je eine MVVM Architektur, womit ist die Benutzeroberfläche von der Logik abgekoppelt ist. Die Werte werden mittels Data Binding in die Screens integriert. Um die einzelnen Views in der Oberfläche zu verwenden, wird über ein einzelnes ViewModel kontrolliert, welche View gerade zu sehen ist.

Für die Erstellung eines einheitlichen Designs wird die EUROIMMUN interne Design Bibliothek genutzt, die nach der Material Design Sprache von Google geschrieben ist. Implementiert ist diese ebenfalls in WPF und lässt sich durch Auslagerung in sogenannte „Resource Dictionaries“ einbinden.

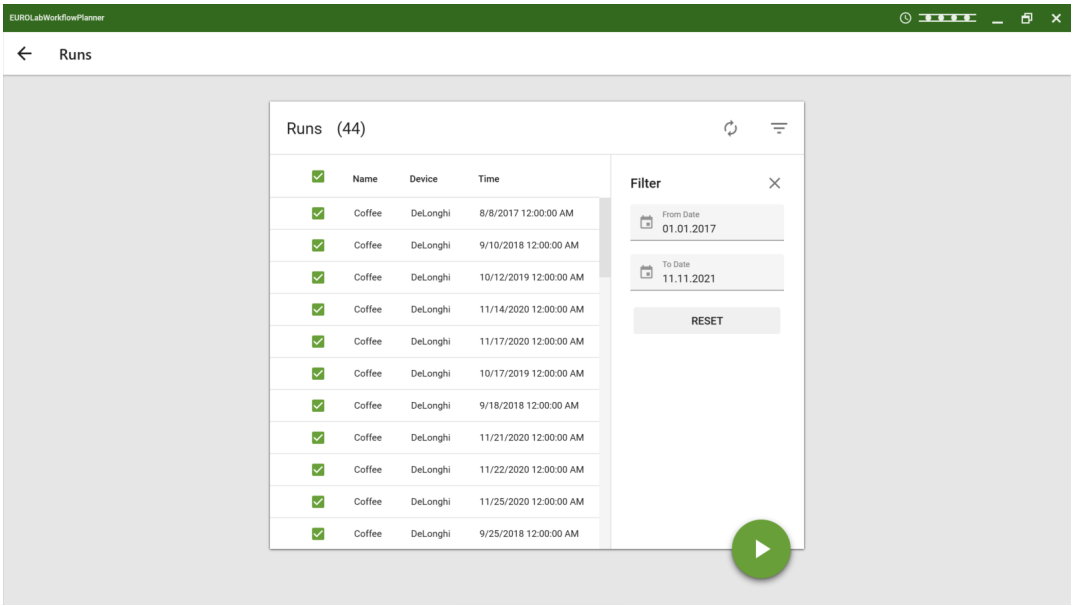

Abbildung 1: Hauptansicht der abgefragten Daten

4.4 Implementierung der Geschäftslogik

Die Implementierung der Schnittstelle via gRPC wurde vom Betrieb vorgegeben, da die Technologie an anderen Stelle bereits verwendet wurde und somit ein gewisses Grundwissen vorhanden war.

Zunächst war das Ziel eine Schnittstelle zu entwickeln, die einfach von anderen Gerätesoftware-Projekten implementiert werden kann. Diese Schnittstelle wird entwickelt und in ein NuGet-Paket ausgelagert. Durch die Nutzung eines Firmeninternen GitLab-Servers wird dieses Paket für die firmeninterne Nutzung freigegeben.

Zur Implementierung dieser Schnittstelle muss die Klasse „RPCServer“ (s. Listing A.2) initialisiert werden. In den Konstruktor wird dazu das Interface „IEUROLabWorkflowPlannerService“ (s. Listing A.4) rein gereicht. Über eine eigene Implementierung des Interfaces können die Daten entsprechend der Software des Diagnostikgeräts erbracht werden.

Bei Abfrage der Daten werden Verbräuche mit Verbrauchsnamen gekennzeichnet. Anhand dieser Namen werden gleichnamige zusammengerechnet und zur besseren Übersicht gegliedert (s. Abbildung 2). Eine qualitative Bewertung der Daten findet nicht statt, die Daten werden unabhängig von Einheiten verarbeitet.

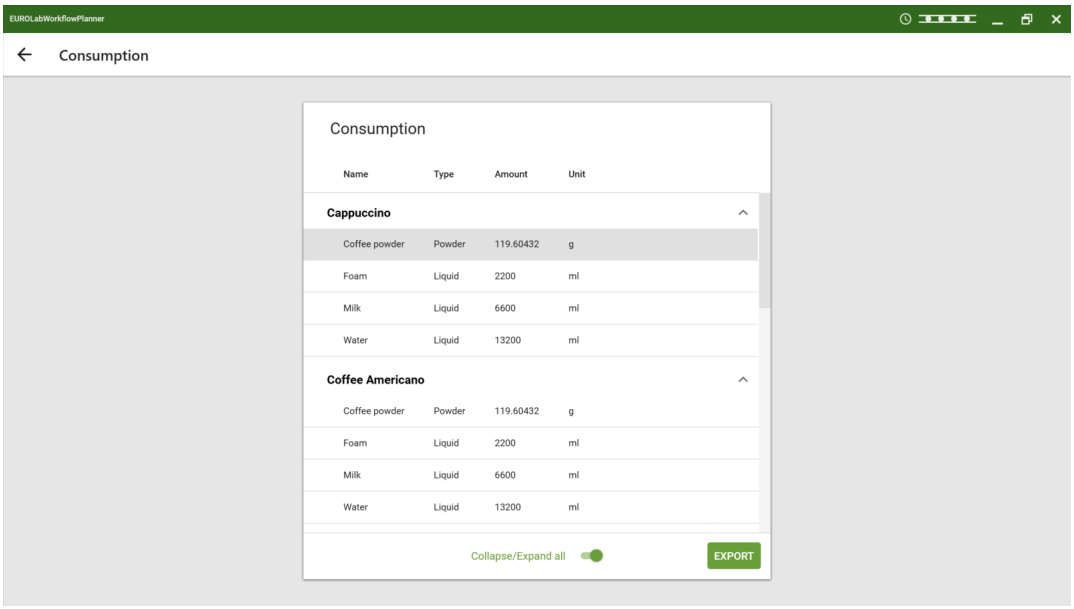

Abbildung 2: Ansicht der zusammengefassten Betriebsmittelverbräuche

5 Abnahmephase

5.1 Testphase

Zur Testung der Anwendung wurden diverse Testfälle manuell durchgeführt. Da die Logik zum Großteil auf der Serverseite vorhanden ist, wurde auf Unit Tests verzichtet.

Anhand von Systemtests wurde sicher gestellt, dass alle Anforderung, die an die Software gestellt wurden umgesetzt sind. Außerdem wurde mithilfe von Performance Tests getestet, wie standhaft die Software in Punkten wie Zuverlässigkeit, Stabilität und Verfügbarkeit ist. Auch beim Senden von größeren Datenmengen von mehreren Diagnostikgeräten gleichzeitig, wies die Anwendung eine gute Performance auf und stürzte nicht ab. Ebenfalls die Serverseite funktionierte zuverlässig und war mit unterschiedlichen Implementierungen immer erreichbar und lieferte auf Abfrage Daten.

Die Testung möglicher auftretender Fehler verlief ebenfalls ohne Mängel. Auf Fälle wie nicht erreichbare Server, Abbruch der Kommunikation oder zu große Datenmengen reagierte die Software und lief zuverlässig weiter.

5.2 Abnahme

Die Abnahme der Schnittstelle und der Software erfolgt durch den Auftraggeber des Projekt, dem Anforderungsmanager zur Prüfung, ob alle Anforderungen erfüllt sind und einem Softwaretester, der die Software ebenfalls auf Zuverlässigkeit testet.

6 Einführungsphase

6.1 Einrichtungsvorraussetzungen

6.1.1 Hardware

Zum Betrieb der Software ist folgende Hardware erforderlich:

- 2 Computer (inkl. Bildschirme etc.)
- Netzwerk, in das beide Computer eingewählt sind

6.1.2 Software

Es wird zum Betrieb der Software eine 64 Bit Version von Windows 7/8/10/11 gefordert, auf der das .NET Framework installiert ist. Des weiteren muss auf der Serverseite der Port zur Abfrage im Netzwerk frei gegeben werden, sodass die Clientseite die Verbrauchsdaten abfragen kann.

6.2 Installation

Zur Verwendung der Software auf einem Gerät, muss vorher das .NET Framework in der Version 4.8 installiert werden, welches von der offiziellen Microsoft Seite heruntergeladen werden kann[4]. Die Anwendung wurde inklusive aller benötigten Dateien auf einem lokalen GitLab-Server hochgeladen. Zur Nutzung müssen somit nur die Daten heruntergeladen werden und die Software kann genutzt werden.

4 .NET Framework: https://dotnet.microsoft.com/download/dotnet-framework/net48

7 Dokumentation

7.1 Dokumentation der Software

Zur Weiterentwicklung durch andere Entwickler wurde der Quellcode der Software mit Kommentaren gekennzeichnet. Neben einer kurzen Beschreibung der Klassen und Funktionen wurde darauf geachtet, dass Variablen möglichst selbsterklärend benannt sind.

Des weiteren wurde eine Anleitung zur Nutzung der Software geschrieben, die einen kurzen Überblick für den Anwender gibt.

7.2 Dokumentation der Schnittstelle

Zur Einbindung der Schnittstelle in andere Softwares zur Steuerung von Diagnostikgeräten, wurde ein NuGet-Paket erstellt, welches implementiert werden muss. Hierbei wurde ebenfalls darauf geachtet, dass Variablennamen selbsterklärend sind. Außerdem wurden die Klassen und Funktionen kommentiert und beschreiben Inhalt und Rückgabeparameter. Des weiteren wurde zum NuGet-Paket eine kurze Beschreibung hinzugefügt.

8 Abschluss

8.1 Soll-/Ist-Vergleich

Ziel des Projektes war die Erstellung und Implementierung einer Schnittstelle, die unabhängig von Diagnostikgerät Daten über Betriebsmittelverbräuche abfragen, zusammenrechnen und exportieren kann. Es wurden zwei Implementierungen der Serverseite erstellt und getestet. Es ist möglich beide Implementierungen zu nutzen. Somit wurden die Anforderungen nach einer unabhängigen Schnittstelle und einer Software, die darauf zugreift, umgesetzt und die erwarteten Funktionen bereit gestellt.

8.2 Erweiterungsmöglichkeiten

Durch die generische Entwicklung ist es einfach möglich anstatt einer gRPC Schnittstelle eine andere Technologie einzubinden und somit sich z.B. direkt mit einer Datenbank zu verbinden oder mithilfe eines anderen Protokolls zu kommunizieren.

Anhand der frei implementierbaren Schnittstelle können weitere Projekte zur Entwicklung einer Software eines Diagnostikgeräts diese einbinden und unabhängig vom Verhalten des Geräts Daten in die vorgegebene Form bringen und zur Abfrage bereit stellen. In Zukunft könnten auf diese Weise die Abfragen der Betriebsmittelverbräuche standardisiert werden.

8.3 Zukunft und Grenzen der Technologie

Im Verlauf der Bearbeitung sind immer wieder Schwierigkeiten aufgetreten, die Defizite an dieser Technologie aufgewiesen haben. Beim Senden von sehr großen Datenmengen z.B. wurde die Performance der Kommunikation meist schlecht und teilweise wurden Antworten des Servers gesendet, jedoch nicht vom Client empfangen.

Gleichzeitig ist gRPC im Gegensatz zu anderen ähnlichen Möglichkeiten der Netzwerk-Kommunikation nicht sehr weit verbreitet, was es schwieriger macht Entwickler zu finden, die ein gewisses Knowhow in diesem Bereich besitzen. Des weiteren ist die Kommunikation nicht standardisiert, was ebenfalls die Erweiterung und Implementierung schwieriger gestaltet. Durch die komplexe Nutzung von HTTP2 ist es nicht möglich gRPC in den Browser zu integrieren.

Dennoch ist gRPC eine gute Möglichkeit innerhalb eines lokalen Netzwerks zu kommunizieren und Daten geringer Größe zu verschicken.

8.4 Probleme

Das schwerwiegendste Problem, das während der Implementierung aufgetreten ist, war dass die automatische Generation der gRPC Klassen in Verbindung mit dem .NET Framework nicht funktioniert hat. Da zur Implementierung von gRPC mit dem .NET Framework unter

C# kaum etwas an Dokumentation zu finden war, hat die Lösung etwas Zeit in Anspruch genommen. Die Generation in Verbindung mit C# und .NET Core hat jedoch ohne Probleme funktioniert und somit musste nur an wenigen Stellen der Quellcode angepasst werden, sodass die Funktion auch mit dem .NET Framework gewährleistet war.

8.5 Fazit

Bei gRPC handelt es sich um eine spannende Technologie, die in gewissen Einsatzgebieten sehr gut Anwendung finden kann.

Speziell in diesem Fall wäre meiner Meinung nach der Einsatz einer alternativen Technologie von Vorteil gewesen, die mit größeren Datenmengen performanter arbeitet, weiter verbreitet und standardisiert ist, um den Fortbestand sicher gewährleisten zu können.

Durch die erfolgreiche Implementierung und Testung der Anforderungen wird das Projekt als Erfolg eingestuft.

Eidesstattliche Erklärung

Ich, Lukas Klettke, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

> *Entwicklung einer allgemeinen Ressourcenverwaltung für Diagnostikgeräte via gRPC*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Lübeck, den 01.12.2021

Lukas Klettke

A Anhang

A.1 Definition der gRPC Schnittstelle

```
syntax = "proto3";

option csharp_namespace = "Euroimmun.EUROLabWorkflowPlanner.RPCServer.gRPCGenerated";

message Processing {
  int32 Id = 1;
  string DeviceName = 2;
  string ProcessingName = 3;
  Timestamp StartTime = 4;

  repeated Consumption Consumptions = 5;
}

message Consumption {
  int32 Id = 1;
  int32 Count = 2;
  string Name = 3;

  repeated Resource Resources = 4;
}

message Resource {
  int32 Id = 1;
  string Name = 2;
  string CodeId = 3;
  string Type = 4;
  double Amount = 5;
  string Unit = 6;
}

message GetProcessingsResponse {
  repeated Processing Processings = 1;
}

message Timestamp {
  int32 Year = 1;
```

```
37    int32 Month = 2;
38    int32 Days = 3;
39    int32 Hours = 4;
40    int32 Minutes = 5;
41  }
42
43  message GetProcessingsRequest {
44    Timestamp FromDate = 1;
45    Timestamp ToDate = 2;
46  }
47
48  service EUROLabWorkflowPlannerService {
49    rpc GetProcessings(GetProcessingsRequest) returns (GetProcessingsResponse) {}
50  }
```

Listing 1: Definition der gRPC Schnittstelle

A.2 Serverseitige Klasse

```
1   public class RPCServer
2   {
3     #region Fields
4
5     private readonly Server _server;
6
7     #endregion
8
9     #region Constructors
10
11    /// <summary>
12    /// Creates the RPC-Server with the fewest necessary values
13    /// </summary>
14    /// <param name="host"></param>
15    /// <param name="port"></param>
16    /// <param name="EUROLabWorkflowPlannerService"></param>
17    public RPCServer(string host, int port,
18    IEUROLabWorkflowPlannerService euroLabWorkflowPlannerService)
19    {
20      if (host == null)
21      throw new NullReferenceException("Host");
```

```
22      if (port == 0)
23      throw new NullReferenceException("Port");
24
25      _server = new Server
26      {
27        Services =
28        {
29          EUROLabWorkflowPlannerService.BindService(
30          new EUROLabWorkflowPlannerServiceServerImpl(euroLabWorkflowPlannerService))
31        },
32        Ports =
33        {
34          new ServerPort(host, port, ServerCredentials.Insecure)
35        }
36      };
37    }
38
39    #endregion
40
41    #region Public Methods
42
43    /// <summary>
44    /// tries to start the RPC-Server. Returns true if success
45    /// </summary>
46    /// <returns></returns>
47    public bool StartServer()
48    {
49      try
50      {
51        _server.Start();
52        return true;
53      }
54      catch (Exception exception)
55      {
56        return false;
57      }
58    }
59
60    /// <summary>
61    /// Shuts down the server
62    /// </summary>
63    public void StopServer()
64    {
```

```
65      _server.ShutdownAsync();
66    }
67  }
```

Listing 2: Serverseitige Klasse

A.3 Clientseitige Klasse

```
1   public class RPCClient
2   {
3     #region Fields
4
5     private readonly Channel _channel;
6
7     private readonly EUROLabWorkflowPlannerService.↩
          EUROLabWorkflowPlannerServiceClient _client;
8
9     #endregion
10
11    #region Constructors
12
13    /// <summary>
14    /// Creates the RPC-Client with the fewest necessary values
15    /// </summary>
16    /// <param name="host"></param>
17    /// <param name="port"></param>
18    public RPCClient(string host, int port)
19    {
20      if (host == null)
21      throw new NullReferenceException("Host");
22      if (port == 0)
23      throw new NullReferenceException("Port");
24
25      Host = host;
26      Port = port;
27
28      _channel = new Channel(host, port, ChannelCredentials.Insecure);
29
30      _client = new EUROLabWorkflowPlannerService.EUROLabWorkflowPlannerServiceClient↩
            (_channel);
```

```
31      }
32
33      #endregion
34
35      #region Properties
36
37      public string Host { get; }
38
39      public int Port { get; }
40
41      #endregion
42
43      #region Public Methods
44
45      /// <summary>
46      /// Requests the runs from the server and filters for the selected Dates. Returns
            the interface if success
47      /// </summary>
48      /// <param name="FromDate"></param>
49      /// <param name="ToDate"></param>
50      /// <returns></returns>
51      public IGetProcessingResponse Request(IGetProcessingRequest getProcessingRequest)
52      {
53        try
54        {
55          var request = new GetProcessingsRequest()
56          {
57            FromDate = DateTimeToClientTimestampConverter.ConvertDateTimeToTimestamp(
                  getProcessingRequest.FromDate),
58            ToDate = DateTimeToClientTimestampConverter.ConvertDateTimeToTimestamp(
                  getProcessingRequest.ToDate)
59          };
60
61          var response = _client.GetProcessings(request);
62
63          return new GetProcessingResponse(InterfaceToClientGrpcConverter.
                ResponseToServiceConverter(response));
64        }
65        catch
66        {
67          return null;
68        }
69      }
```

```
70
71    /// <summary>
72    /// Stops the Client
73    /// </summary>
74    public void StopClient()
75    {
76      _channel.ShutdownAsync();
77    }
78  }
```

Listing 3: Clientseitige Klasse

A.4 Schnittstelle

```
1  public interface IEUROLabWorkflowPlannerService
2  {
3    IGetProcessingResponse GetProcessings(IGetProcessingRequest request);
4  }
```

Listing 4: Schnittstelle

A.5 Export Datei

	A	B	C	D
1	Coffee Americano			
2	Water	Liquid	13200	ml
3	Milk	Liquid	6600	ml
4	Foam	Liquid	2200	ml
5	Coffee powder	Powder	119,60432	g
6	Espresso Macchiato			
7	Water	Liquid	13200	ml
8	Milk	Liquid	6600	ml
9	Foam	Liquid	2200	ml
10	Coffee powder	Powder	119,60432	g
11	Cappuccino			
12	Water	Liquid	13200	ml
13	Milk	Liquid	6600	ml
14	Foam	Liquid	2200	ml
15	Coffee powder	Powder	119,60432	g
16	Flat White			
17	Water	Liquid	13200	ml
18	Milk	Liquid	6600	ml
19	Foam	Liquid	2200	ml
20	Coffee powder	Powder	119,60432	g
21	Coffee Latte			
22	Water	Liquid	13200	ml
23	Milk	Liquid	6600	ml
24	Foam	Liquid	2200	ml
25	Coffee powder	Powder	119,60432	g
26	Latte Macchiato			
27	Water	Liquid	13200	ml
28	Milk	Liquid	6600	ml
29	Foam	Liquid	2200	ml
30	Coffee powder	Powder	119,60432	g

Abbildung 3: Exportierte „.xlsx“ Datei