

Pflichtenheft und technische Spezifikation im Programmierprojekt

IIoT Simulator

TeamMitglieder: *Aleksandr Terekhov,*
 Mjellma Salihi,
 Housseem Hfasa,
 Paul Schult

Auftraggeber: Prof. Dr. E.Rodner (HTW Berlin)

Datum: 11.05.2022

Inhaltsverzeichnis

Visionen und Ziele	1
Anforderungsanalyse	2
Use-Cases	2
Risiken	3
GUI	4
Realisierung	10
Allgemeines	10
Interne Schnittstellen	14
Externe Schnittstellen	20
Entwicklungs- und Teststrategie	21
Lizenz	22

1 Visionen und Ziele

Industrieprojekte z.B. in der Fertigung neuer Produkte werden zunehmend komplexer. Eine Vielzahl von Sensoren werden in der sogenannten Industrie 4.0. zur besseren Steuerung und präziseren Fertigung angewendet. Um neue Projekte schon frühzeitig auf Herz und Nieren testen zu können, bietet unser IIOT Simulator die einfache Möglichkeit bereits frühzeitig die Machbarkeit eines Industrie Projektes zu validieren.

Ziel des Projektes ist die Erstellung einer grafischen Desktop-Anwendung, die es dem Nutzer erlaubt Sensordaten verschiedener Sensortypen zu simulieren und mittels MQTT Protokoll zu übertragen. Dabei kann der Nutzer Sensorgruppen erstellen, die verschiedene Sensoren enthalten. Die Sensordaten können mit verschiedensten Nutzereingaben nach Wunsch erzeugt werden. Es besteht die Möglichkeit mit verschiedenen Methoden Messdaten mit Fehlerwerten zu versehen. Die Daten können über einen frei wählbaren externen MQTT Broker versendet werden.

2.2 Risiken

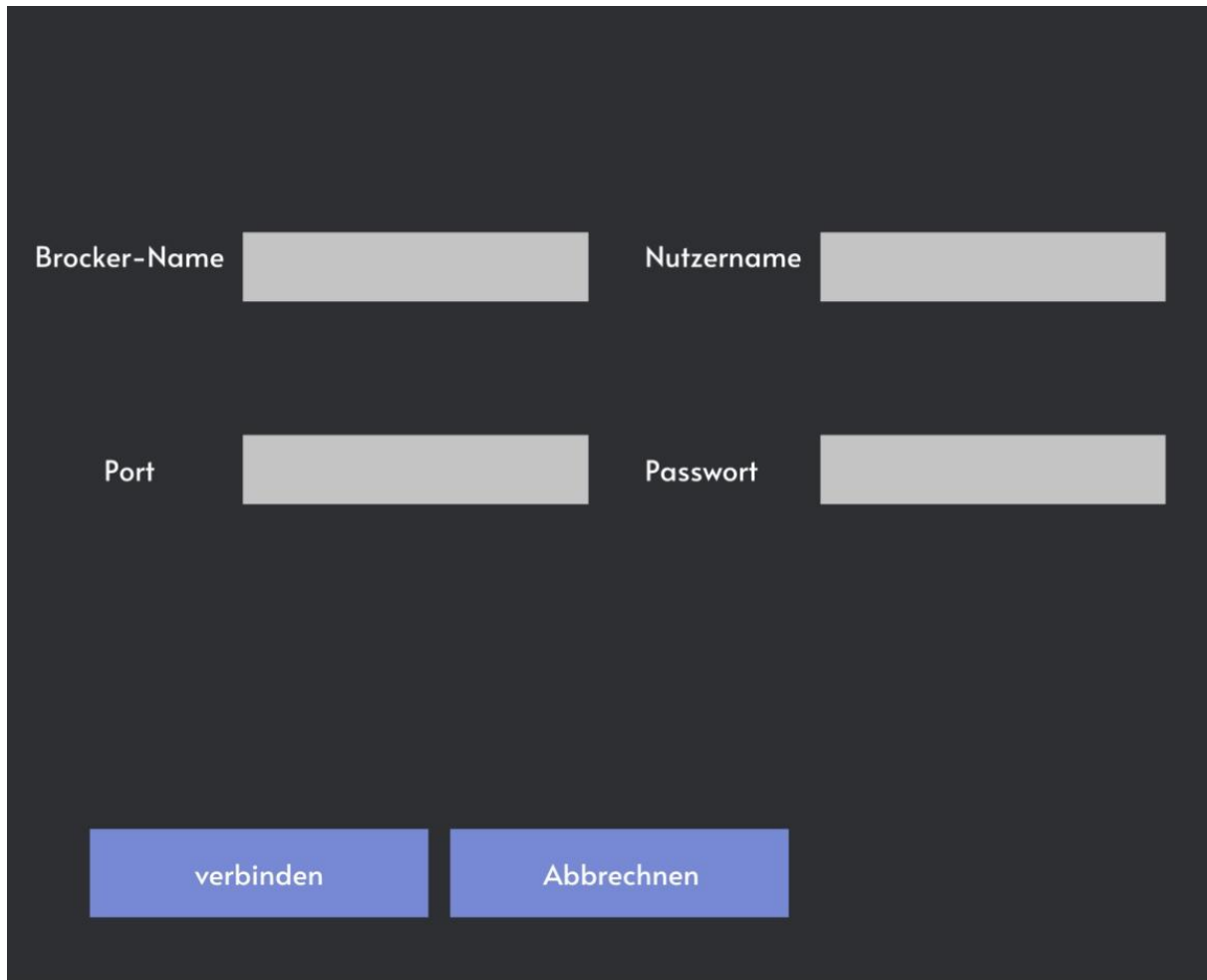
	Risiken	Maßnahmen
1.	Wenig Erfahrung als Team in großen Projekten zu arbeiten -> Fehleranfälligkeit bei Kommunikation	Frühzeitiges einführen von Projektmanagement Prinzipien, Bestimmung Projektleitung
2.	Zeitverzug durch unvorhergesehene Aufgaben/Probleme	Projektmanagement mit Meilensteinen, regelmäßigen Meetings und einplanen von Zeitpuffern
3.	Verwendung neuer Nuget Pakete und Features (z.B. MQTTnet etc.)	Experimentieren mit Paketen in Phase 1, einlesen in Dokumentationen, Besprechung der Pakete in Meetings
4.	Verzögerungen/Probleme durch Kollisionen in Gitlab	Herbeiführen und lösen von Testkollisionen, Aufteilung in Teilkomponenten, Gitlab-"Führerschein" absolviert
5.	Kompatibilität der Komponenten kann durch Änderungen an Komponenten reduziert werden	Verwendung von CommonInterfaces, Änderung des Interfaces nur in Absprache möglich

Tabelle 1: Risiken und Gegenmaßnahmen

2.3 GUI



Abbildung 1: Startseite- Auswahl zwischen ‚Broker Einstellungen‘, ‚Neue Sensorgruppe erstellen‘ und ‚Bestehende Sensorgruppe laden‘.



A dark-themed user interface for establishing a connection. It features four input fields arranged in a 2x2 grid. The top row contains 'Broker-Name' and 'Nutzername'. The bottom row contains 'Port' and 'Passwort'. Below the input fields are two blue buttons: 'verbinden' on the left and 'Abbrechen' on the right.

Abbildung 2: Verbindung mit dem Broker kann hier hergestellt werden- Einfügen von 'Broker-Name', 'Nutzername', 'Port', und 'Passwort'.

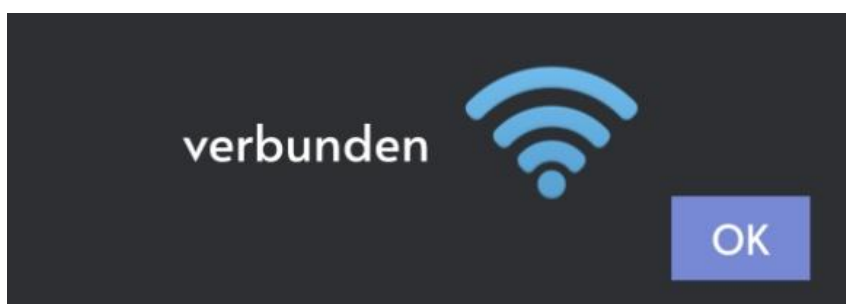


Abbildung 3: Verbindungsbestätigung nach erfolgreicher Broker Verbindung. Auswahl- 'OK'.

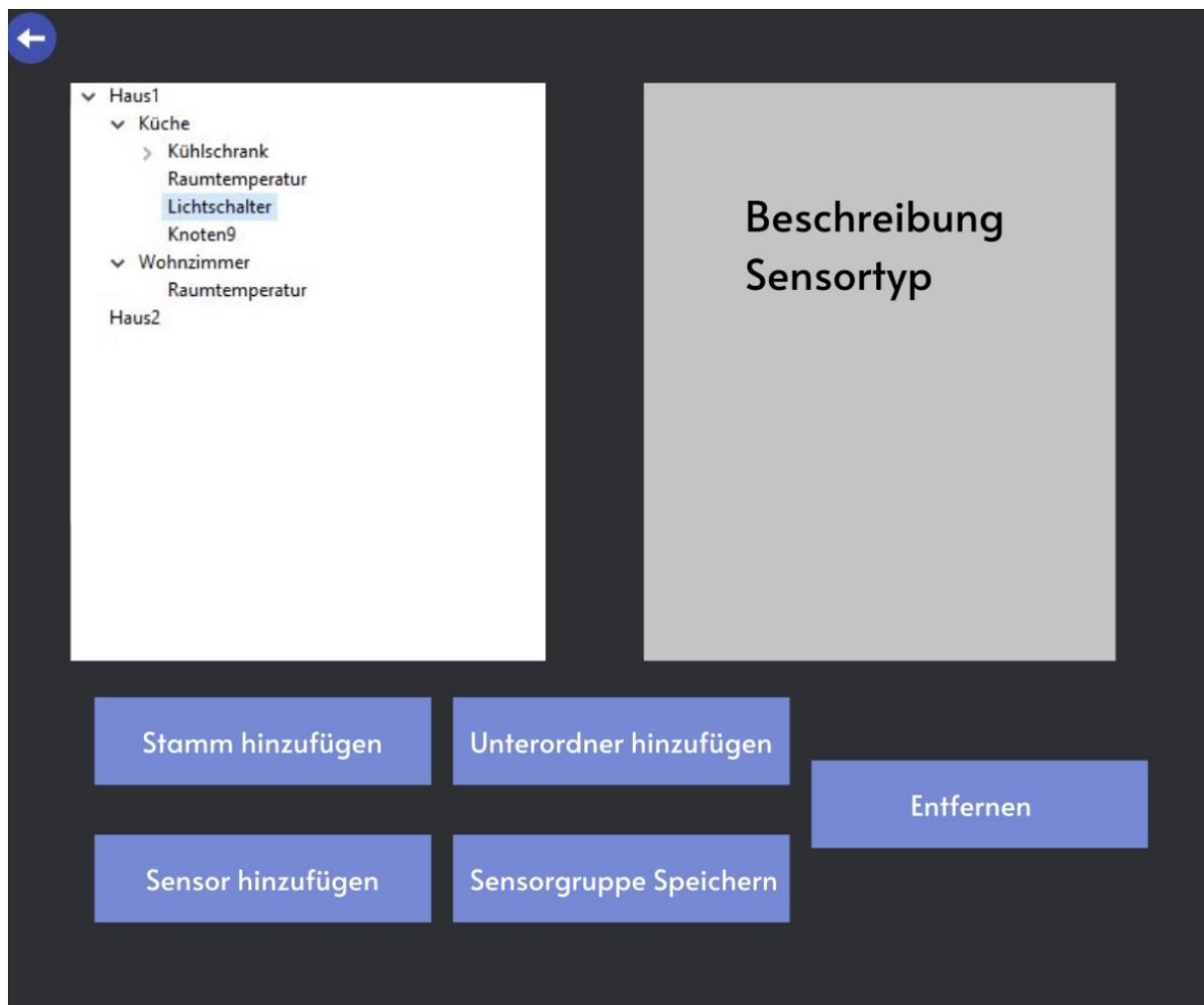


Abbildung 4: Hier kann eine neue Sensorgruppe erstellt und vorgenommene Änderungen angezeigt werden. Auswahl zwischen 'Stamm hinzufügen', 'Unterordner hinzufügen', 'Sensor hinzufügen', 'Sensorgruppe speichern' und 'Entfernen'.

The screenshot shows a web interface with a dark grey background. At the top, there are two large, light grey rectangular boxes. The left box is labeled 'Liste von Sensoren' and the right box is labeled 'Beschreibung Sensortyp'. Below these boxes, there are two blue buttons with white text: 'hinzufügen' on the left and 'Abbrechen' on the right.

Abbildung 5: Nach Erstellung einer neuen Sensorgruppe wird hier die Liste von Sensoren und die Beschreibung der Sensortypen angezeigt. Auswahl zwischen 'Hinzufügen' und 'Abbrechen' um zurück zur Startseite zu gelangen.

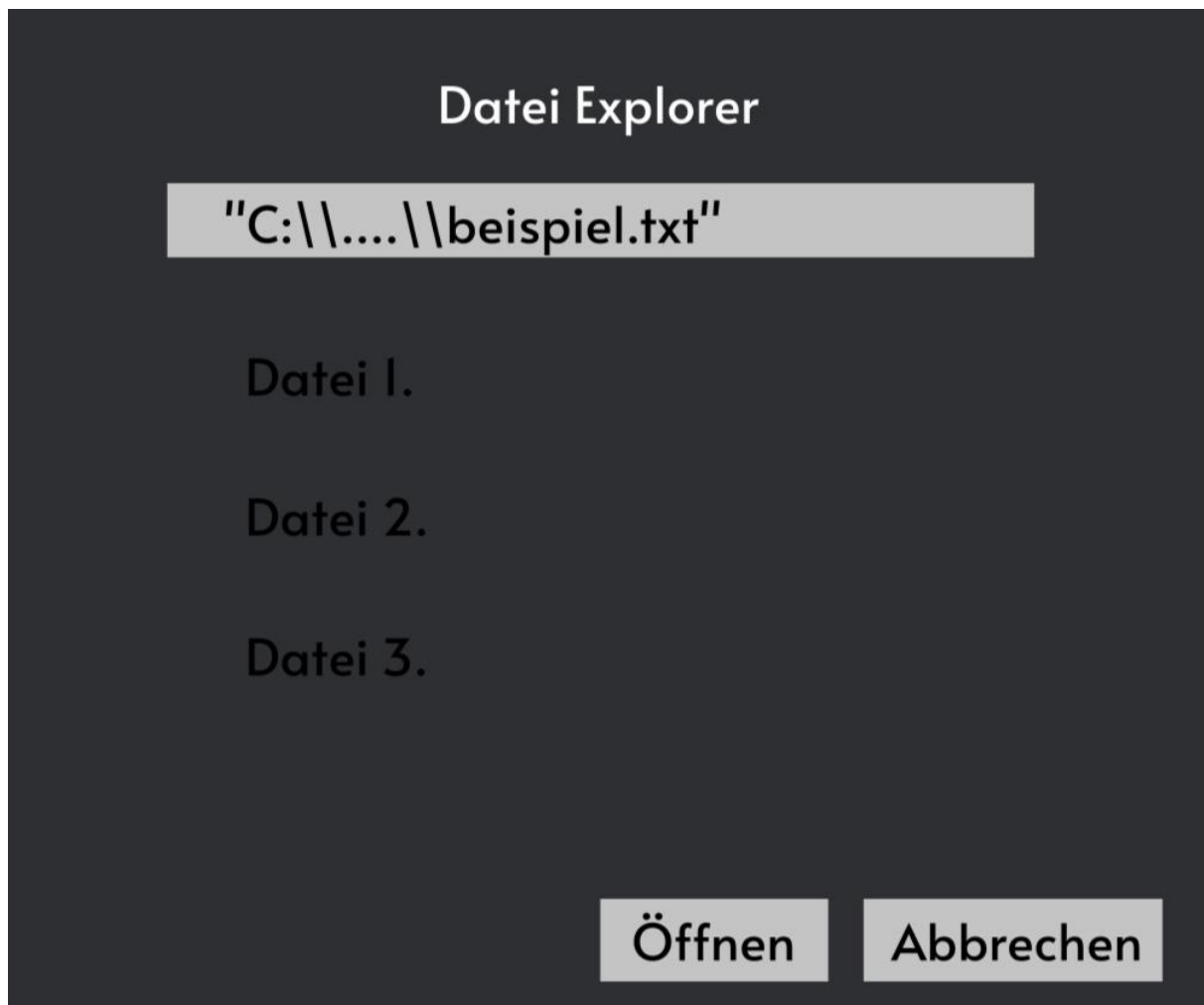


Abbildung 6: Dateieexplorer zum suchen abgespeicherter Sensorgruppen. Auswahl der Sensorgruppen in der Liste, 'Öffnen' zum laden der Sensorgruppe und 'Abbrechen' um zurück zur Startseite zu gelangen.

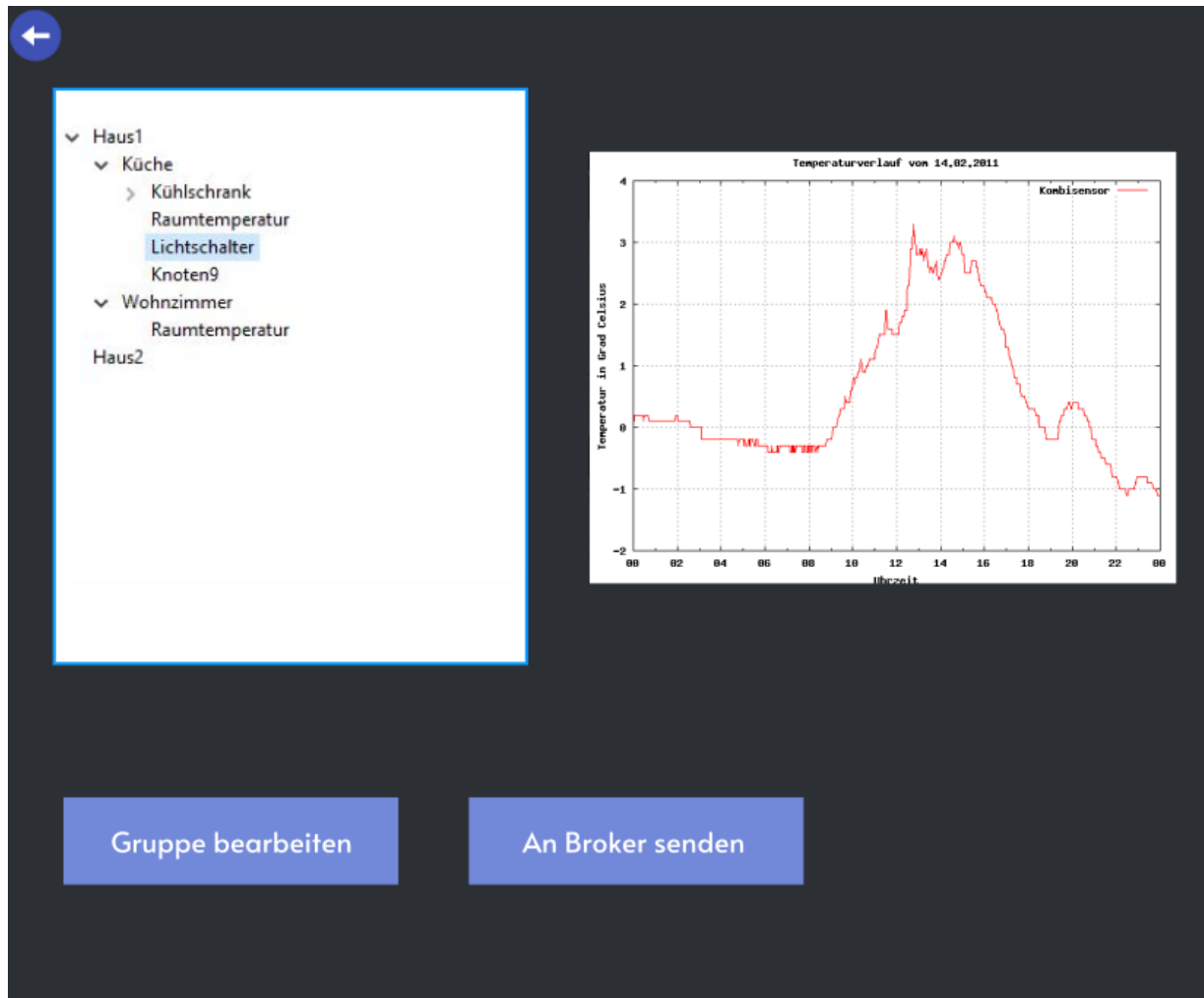


Abbildung 7: Hat man eine bestehende Sensorgruppe geladen, gelangt man auf diese Seite in der die Sensorgruppe und die passende Visualisierung angezeigt werden. Auswahl- 'Gruppe bearbeiten' um Abbildung 4 zu gelangen. Über den Button "An Broker senden" werden die Daten des ausgewählten Sensors an den eingestellten MQTT Broker gesendet

3 Realisierung

3.1 Allgemeines

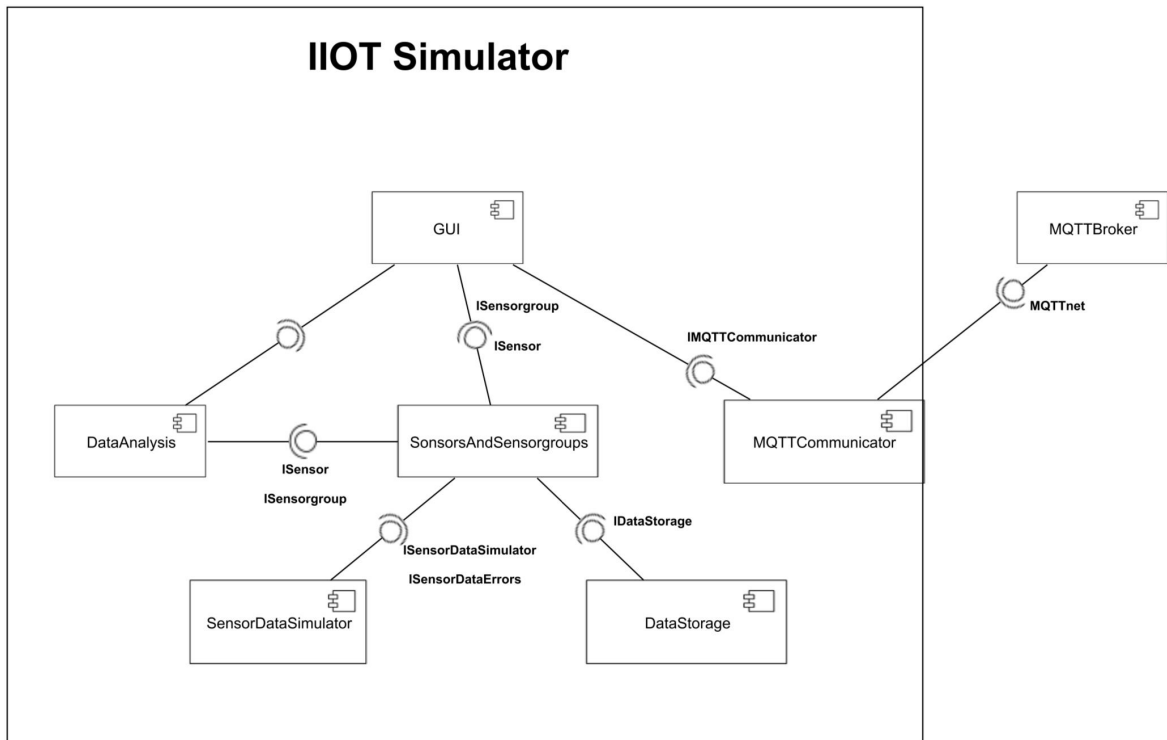


Abbildung 8: Komponentendiagramm des Projektes

GUI:

Benutzeroberfläche des Programms.

SensorsAndSensorgroups:

Stellt Sensor- und Sensorgruppen-Klassen mit den entsprechenden Methoden bereit.

SensorDataSimulator:

Stellt Klassen zur Erzeugung von Messwerten mit und ohne Fehlerkomponenten bereit.

DataStorage:

Komponente zum Laden und Speichern von Sensoren, Sensorgruppen und den MQTT Einstellungen.

MQTTCommunicator:

Erstellt die Verbindung zum MQTT Broker.

DataAnalysis:

Komponente, die Sensordaten für die Visualisierungen aufbereitet.

(Diese Komponente wird nach aktuellem Stand nicht benötigt, wird aber in Absprache mit dem Auftraggeber dennoch aufgeführt.)

Komponente	wird bearbeitet von..
<i>SensorDataSimulator</i>	<i>Paul</i>
<i>DataStorage</i>	<i>Housseem</i>
<i>SensorAndSensorgroups</i>	<i>Housseem, Paul</i>
<i>MQTTCommunicator</i>	<i>Alexander</i>
<i>GUI</i>	<i>Mjellma</i>
<i>DataAnalysis</i>	<i>Nach Bedarf/ Workload</i>

Unter zuhilfenahme des Komponentendiagramms wurden die Klassendiagramme erstellt.

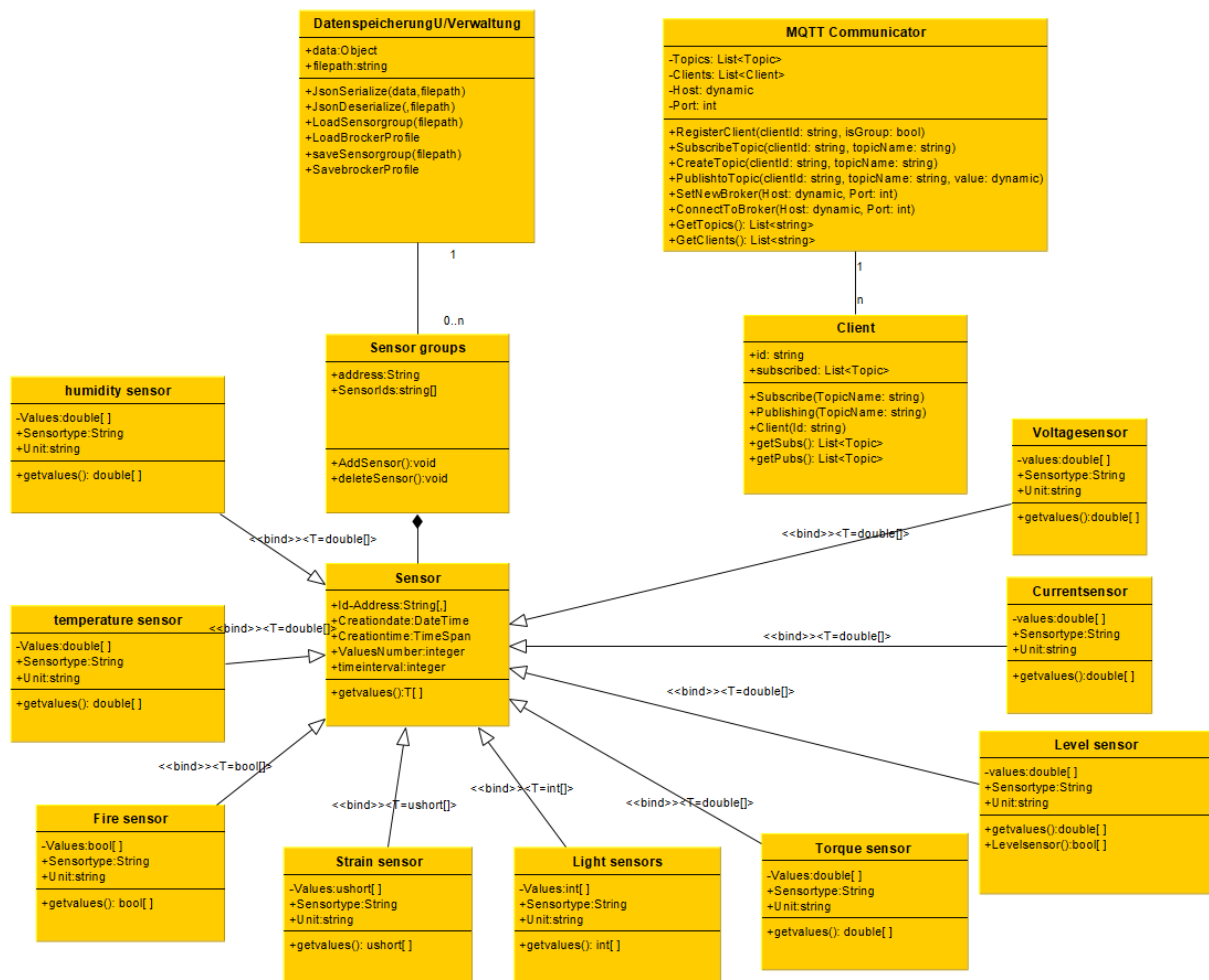


Abbildung 9: Klassendiagramm Teil 1.

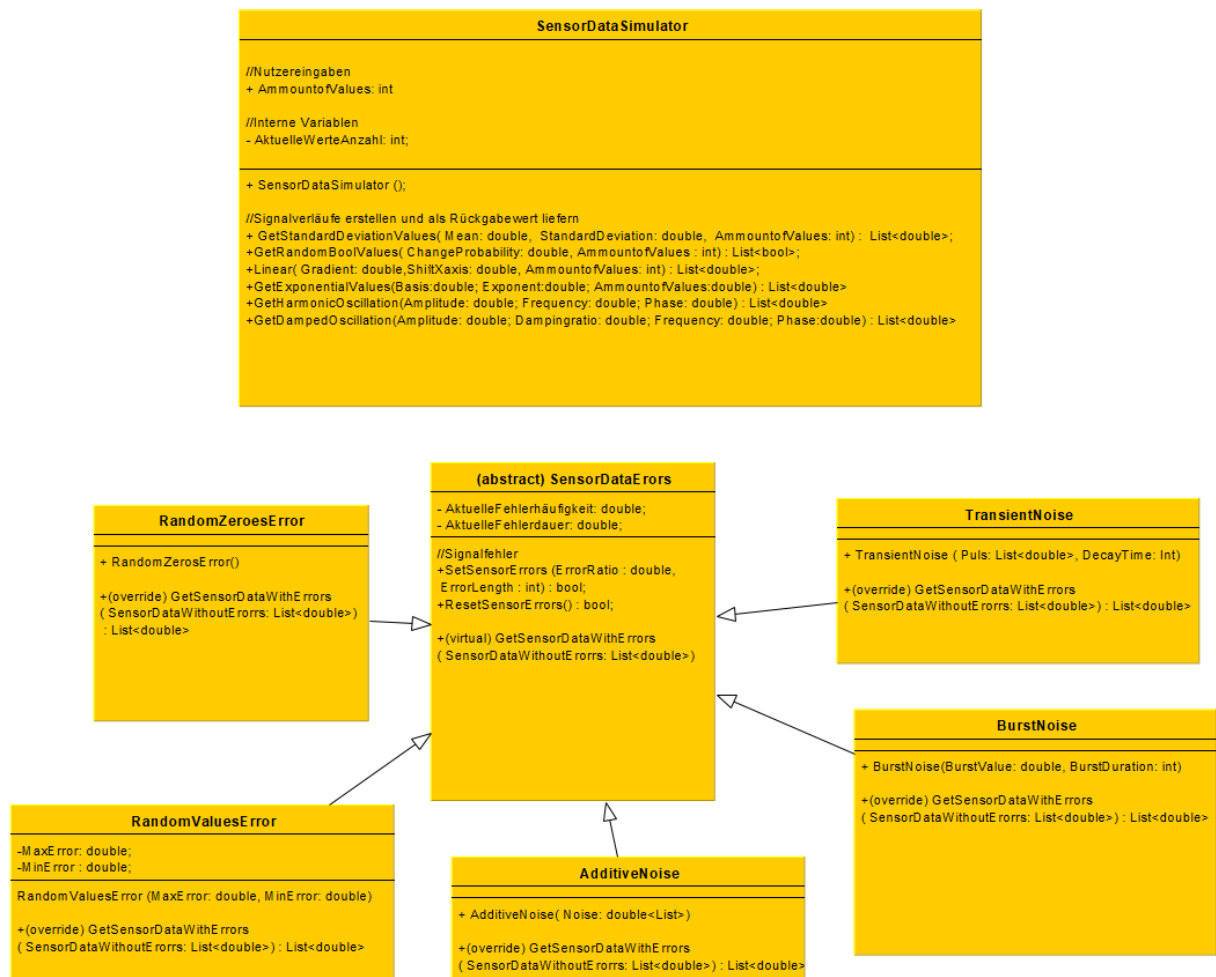


Abbildung 10: Klassendiagramm Teil 2.

3.2 Interne Schnittstellen

Aus den entworfenen Klassendiagrammen wurden die CommonInterfaces abgeleitet.

```
public interface ISenor<T>
{
    public string Sensor_id { get; set; }

    // 2 Dimensionales Array
    public string[,] Id_Adresse { get; set; }
    public string Sensortype { get; }
    public string Einheit { get; set; }
    public DateTime CreationDate { get; }
    public TimeSpan CreationTime { get; }
    public int Werteanzahl { get; }
    public int Timeinterval { get; }
    //bekommt die Daten von der Sensoren
    public abstract List<T> Getvalues();
}
```



```
public interface ISensorGroups<T> where T : ISenor<T>
{
    // allgemeine Adresse für die Sensorgruppe
    0 Verweise | Paul S, Vor 5 Minuten | 1 Autor, 1 Änderung
    string Adresse { get; set; }

    // Das Verzeichnis, welche Sensoren sich wo befinden. (vorläufig)
    0 Verweise | Paul S, Vor 5 Minuten | 1 Autor, 1 Änderung
    List<List<object>> GroupDirectory { get; set; }

    // alle Ids die in diesem Gruppe sind, evtl nicht mehr benötigt
    0 Verweise | Paul S, Vor 5 Minuten | 1 Autor, 1 Änderung
    string[] SensorIds { get; set; }

    /// <summary>
    /// Ein Sensor_Id in der SensorIds Liste hinzufügen
    /// </summary>
    /// <param name="sensorids"> die Liste von Sensorids </param>
    /// <param name="sensorid"> das id zu hinzufügen zur Id_liste </param>
    0 Verweise | Paul S, Vor 5 Minuten | 1 Autor, 1 Änderung
    public void Sensorhinzufuegen( T Sensor);

    /// <summary>
    /// Ein Sensor_Id von der SensorIds Liste loeschen
    /// </summary>
    /// <param name="sensorids"> die Liste von Sensorids </param>
    /// <param name="sensorid"> das id zu loeschen von der Liste </param>
    0 Verweise | Paul S, Vor 5 Minuten | 1 Autor, 1 Änderung
    public void Sensorloeschen(string sensorid);

    //Stamm hinzufügen
    0 Verweise | Paul S, Vor 5 Minuten | 1 Autor, 1 Änderung
    public void AddBase(string BaseName);

    // Unterordner hinzufügen
    0 Verweise | Paul S, Vor 5 Minuten | 1 Autor, 1 Änderung
    public void AddNode(string NodeName, string[] NodeAdress);

    //Löschen von Stamm/Unterordner
    0 Verweise | Paul S, Vor 5 Minuten | 1 Autor, 1 Änderung
    public void DeleteNodeBase(string[] Adress);
}
```

```
public interface IMQTTCommunicator
{
    // Überlegung ob Rückgabewerte benötigt wird für Rückmeldung von Erfolg/Nichterfolg
    //Welche Informationen werden zur Registrierung des Clients benoetigt? /Paul
    0 Verweise | Paul S, Vor 6 Minuten | 1 Autor, 1 Änderung
    public void ConnectToBroker(dynamic Host, int Port);

    0 Verweise | Paul S, Vor 6 Minuten | 2 Autoren, 2 Änderungen
    public void RegisterClient(string clientId, bool isGroup);

    0 Verweise | Aleksandr Terekhov, Vor 17 Stunden | 1 Autor, 2 Änderungen
    public void SubscribeTopic(string clientId, string topicName);

    0 Verweise | Aleksandr Terekhov, Vor 17 Stunden | 1 Autor, 2 Änderungen
    public void CreateTopic(string clientId, string topicName);

    0 Verweise | Aleksandr Terekhov, Vor 17 Stunden | 1 Autor, 2 Änderungen
    public void PublishToTopic(string clientId, string topicName, dynamic value);

    0 Verweise | Aleksandr Terekhov, vor 2 Tagen | 1 Autor, 1 Änderung
    public void SetNewBroker(dynamic Host, int Port);

    0 Verweise | Aleksandr Terekhov, Vor 15 Stunden | 1 Autor, 1 Änderung
    public List<string> GetTopics();
    0 Verweise | Aleksandr Terekhov, Vor 15 Stunden | 1 Autor, 1 Änderung
    public List<string> GetClients();
}
```

```
public interface IDatastorage
{
    // die Daten die von der Sensoren kommt
    0 Verweise | Paul S, Vor 6 Minuten | 2 Autoren, 2 Änderungen
    public object Data { get; }
    //Dateipfad der Datei
    0 Verweise | Paul S, Vor 6 Minuten | 2 Autoren, 4 Änderungen
    public string filepath { get; }
    /// <summary>
    /// serialise die Daten zu Textdatei
    /// </summary>
    /// <param name="data"> die Daten zu speichern </param>
    /// <param name="filepath"> Dateipfad, wo die Daten werden gespeichert </param>
    0 Verweise | Houssem Hfasa, Vor 10 Stunden | 1 Autor, 4 Änderungen
    public void JsonSerializer(object data, string filepath);
    /// <summary>
    /// deserialise Textdatei zu Json datei ,um die gespeicherte Datei zu laden
    /// </summary>
    /// <param name="filepath"> Dateipfad, wo die Daten sind gespeichert </param>
    0 Verweise | Houssem Hfasa, Vor 10 Stunden | 1 Autor, 5 Änderungen
    public Object JsonDeserialize(string filepath);
    /// <summary>
    /// deserialise Textdatei zu Json datei ,um die gespeicherte sensorgruppe zu laden
    /// </summary>
    /// <param name="filepath"> Dateipfad, wo die Daten sind gespeichert </param>
    0 Verweise | Houssem Hfasa, Vor 10 Stunden | 1 Autor, 3 Änderungen
    public object LoadSensorgroup(string filepath);
    /// <summary>
    /// deserialise Textdatei zu Json datei ,um die gespeicherte BrockerProfile zu laden
    /// </summary>
    /// <param name="filepath"> Dateipfad, wo die Daten sind gespeichert </param>
    0 Verweise | Houssem Hfasa, Vor 10 Stunden | 1 Autor, 3 Änderungen
    public object LoadBrockerProfile(string filepath);
    /// <summary>
    /// speichern die Sensorgruppe
    /// </summary>
    /// <param name="data"> die liste mit der Sensor_ids des gruppes </param>
    /// <param name="filepath"> Dateipfad, wo die Sensorgroup werden gespeichert </param>
    0 Verweise | Houssem Hfasa, Vor 10 Stunden | 1 Autor, 2 Änderungen
    public void SaveSensorgroup(object data, string filepath);
    /// <summary>
    /// speichern die BrockerProfile
    /// </summary>
    /// <param name="data"> die BrockerProfileDaten </param>
    /// <param name="filepath"> Dateipfad, wo die BrockerProfileDaten werden gespeichert </param>
    0 Verweise | Houssem Hfasa, Vor 10 Stunden | 1 Autor, 2 Änderungen
    public void SavebrockerProfile(object data, string filepath);
}
```

```
public interface ISensorDataErrors
{
    //Fehlerrate
    0 Verweise | Paul S, Vor 6 Minuten | 1 Autor, 1 Änderung
    public double ErrorRatio { get; }
    // Fehlerdauer
    0 Verweise | Paul S, Vor 6 Minuten | 1 Autor, 1 Änderung
    public int ErrorLength { get; }
    //FehlerMaximum
    0 Verweise | Paul S, Vor 6 Minuten | 1 Autor, 1 Änderung
    public double ErrorMax { get; }
    //FehlerMinimum
    0 Verweise | Paul S, Vor 6 Minuten | 1 Autor, 1 Änderung
    public double ErrorMin { get; }

    //Einstellung des Messfehlers/Signalfehlers, Fehlerwerte werden ebenfalls beim Konstruktor als Parameter übergeben
    /// <summary>
    /// Setzt die Parameter für Fehlerwerte
    /// </summary>
    /// <param name="ErrorRatio"> Fehlerhäufigkeit. Wert muss zwischen 0.0 und 1.0 liegen </param>
    /// <param name="ErrorLength"> Länge des Fehlers </param>
    /// <param name="ErrorMax"> Maximalwert des Fehlers</param>
    /// <param name="ErrorMin"> Minimalwert des Fehlers</param>
    0 Verweise | Paul S, Vor 6 Minuten | 1 Autor, 1 Änderung
    bool SetSensorErrors(double ErrorRatio, int ErrorLength, double ErrorMax, double ErrorMin);

    /// <summary>
    /// Setzt die Parameter für Fehlerwerte zurück auf 0
    /// </summary>
    0 Verweise | Paul S, Vor 6 Minuten | 1 Autor, 1 Änderung
    bool ResetSensorErrors();

    /// <summary>
    /// Nimmt eine Liste von Messwerten mit double Werten, baut Fehler/Rauschen ein und gibt die editierte Liste zurück. Fehlererzeugungart siehe KlassenDokumentation
    /// </summary>
    /// <param name="SensorDataWithoutErrors"> ursprüngliche Sensordaten/Messwerte </param>
    0 Verweise | Paul S, Vor 7 Minuten | 1 Autor, 1 Änderung
    List<double> GetSensorDataWithErrors(List<double> SensorDataWithoutErrors);
}
```

```
public interface ISensorDataSimulator
{
    0 Verweise | Paul S, vor 2 Tagen | 1 Autor, 2 Änderungen
    public int AmmountofValues { get; }

    //Funktionen zur Erzeugung von Messwerten

    /// <summary>
    /// Erzeugt eine Liste mit stetigen, normalverteilten double Werten der Anzahl AmmountofValues.
    /// </summary>
    /// <param name="Mean"> Mittelwert </param>
    /// <param name="StandardDeviation"> Standardabweichung </param>
    /// <returns>Liste zufälliger, normalverteilter double Werte</returns>
    0 Verweise | Paul S, Vor 1 Tag | 1 Autor, 2 Änderungen
    List<double> GetStandardDeviationValues(double Mean, double StandardDeviation, int AmmountofValues);

    /// <summary>
    /// Erzeugt eine Liste mit zufällig erzeugten bool Werten
    /// </summary>
    /// <param name="Toggleprobability"> Wert zw. 0-1. Umschaltwahrscheinlichkeit 0 -> 1 bzw. 1 -> 0 </param>
    0 Verweise | Paul S, Vor 1 Tag | 1 Autor, 3 Änderungen
    List<bool> GetRandomBoolValues(double Toggleprobability, int AmountofValues);

    /// <summary>
    /// Erzeugt Mithilfe einer harmonischen Schwingungsgleichung eine Liste an double Werten
    /// </summary>
    /// <param name="Amplitude"> Amplitude </param>
    /// <param name="Period"> Periodendauer </param>
    /// <param name="Phase"> Phasenverschiebung </param>
    0 Verweise | Paul S, Vor 14 Stunden | 1 Autor, 3 Änderungen
    List<double> GetHarmonicOscillation(double Amplitude, double Period, double Phase, int AmmountofValues);

    /// <summary>
    /// Erzeugt Mithilfe einer gedämpften harmonischen Schwingungsgleichung eine Liste an double Werten
    /// </summary>
    /// <param name="Amplitude"> Amplitude </param>
    /// <param name="Period"> Periodendauer </param>
    /// <param name="Dampingratio"> Dämpfungsrate </param>
    /// <param name="Phase"> Phasenverschiebung </param>
    0 Verweise | Paul S, Vor 14 Stunden | 1 Autor, 3 Änderungen
    List<double> GetDampedOscillation(double Amplitude, double Dampingratio, double Period, double Phase, int AmmountofValues);

    /// <summary>
    /// Überlagerung zweier Schwingungen/Messwertreihen
    /// </summary>
    /// <param name="Oscillation1"> Liste mit Sensordaten, bevorzugt schwingende Messwerte </param>
    0 Verweise | Paul S, Vor 1 Tag | 1 Autor, 2 Änderungen
    List<double> GetSuperposition(List<double> Oscillation1, List<double> Oscillation2);
}
```

3.3 Externe Schnittstellen

Name	Art	Typ der implementierung	die Komponente	Herausgeber
NUnit	Nuget Paket	Framework assembly	SensorDataSimulator, DataStorage, Sensors Sensor groups	Charlie Poole, Rob Prouse
Newtonsoft.Json	NuGet Paket	Dateizugriff auf Textdatei	DatenSpeicherung	James Newton-King/MIT-Lizenz
MQTTnet	NuGet Paket	MQTT-basierte Kommunikation	MQTTCommunicator	The contributors of MQTTnet

Tabelle .. Externe Schnittstellen, welche zur Hilfe der Umsetzung des Projektes benutzt werden.

Die **Newtonsoft.Json** Bibliothek stellt Klassen bereit, die zum Implementieren der Kerndienste des Frameworks verwendet werden. Es bietet Methoden zum Konvertieren zwischen .NET-Typen und JSON-Typen.

Sie wird verwendet, um die Daten zu serialisieren und deserialisieren.

MQTTnet als NuGet Paket ist eine .NET Bibliothek für MQTT-basierte Kommunikation. Es bietet ein MQTT Client und ein MQTT Server (Broker) an und unterstützt MQTT Protokoll bis zu Version 5.0. Neben dem reinen Push-Messaging durch die Subscribe/Publishing Architektur besitzt MQTT weitere nützliche Features:

- **Unterschiedliche Quality of Service Level:** Um sicherzustellen, dass eine gesendete Nachricht beim Empfänger ankommt.
- **Retained Messages:** Die letzte gesendete Nachricht eines Topics kann wahlweise am Broker hinterlegt werden.
- **Last Will and Testament:** Ein Client kann eine MQTT-Nachricht als „Letzten Willen“ beim Verbindungsaufbau am Broker hinterlegen.

- **Persistent Sessions:** In Anwendungsfällen, bei denen mit häufigen Verbindungsabbrüchen von Clients zu rechnen ist, kann der Broker eine persistente Session vorhalten.

NUnit ist ein Komponenten-Test Framework für .NET Sprachen. Es können Testfälle erzeugt werden, die über eine eigene Konsole ausgeführt und überprüft werden können. Die Testfälle überprüfen während der Programmierung, ob die getesteten Komponenten korrekt funktionieren.

4 Entwicklungs- und Teststrategie

Das Projekt ist anhand des Komponentendiagramms in einzelne Teilprojekte aufgeteilt. Diese werden von den zuständigen Entwicklern eigenständig implementiert. Über die CommonInterfaces wird die Interoperabilität der Komponenten sichergestellt. Eine Änderung erfolgt erst, nachdem diese in einem Meeting besprochen wurde bzw. betroffene Entwickler informiert wurden.

Die Funktionalität der Benutzeroberfläche kann anhand von Dummy-Komponenten überprüft werden. Diese geben z.B. einfache Werte zurück, ohne die dahinter liegenden Berechnungen/Schritte durchzuführen.

Um das Verhalten der Methoden der Komponenten SensorDataSimulator, DataStorage und SensorsAndSensorgroups (ehemals Datamanagent) auf unterschiedlichste Eingaben hin zu überprüfen, werden Unit Tests eingesetzt. So können besonders auftretende "Corner Cases" automatisiert überprüft werden.

Für andere Komponenten würden Unit Tests unser Auffassung nach zu Komplex oder können nicht alle Funktionalitäten überprüfen.

Daher werden alle weiteren Komponenten mit einer eigenen Konsolenanwendung getestet.

Für den MQTTCommunicator ist eine Überprüfung der gesendeten Daten auf den Broker nötig. Dies wird mit einem Zusatzprogramm bzw. einer Zusatzkomponente überprüft.

5 Lizenz

Copyright (c) <2022 > < HTW Berlin ProgrammierProjekt Gruppe 5(Aleksandr Terekhov, Mjellma Salihi,Housseem Hfasa, Paul Schult>

Jedem, der eine Kopie dieser Software und der zugehörigen Dokumentationsdateien (die "Software") erhält, wird hiermit kostenlos die Erlaubnis erteilt, ohne Einschränkung mit der Software zu handeln, einschließlich und ohne Einschränkung der Rechte zur Nutzung, zum Kopieren, Ändern, Zusammenführen, Veröffentlichen, Verteilen, Unterlizenzieren und/oder Verkaufen von Kopien der Software, und Personen, denen die Software zur Verfügung gestellt wird, dies unter den folgenden Bedingungen zu gestatten:

Der obige Urheberrechtshinweis und dieser Genehmigungshinweis müssen in allen Kopien oder wesentlichen Teilen der Software enthalten sein.

DIE SOFTWARE WIRD OHNE MÄNGELGEWÄHR UND OHNE JEDLICHE AUSDRÜCKLICHE ODER STILLSCHWEIGENDE GEWÄHRLEISTUNG, EINSCHLIEßLICH, ABER NICHT BESCHRÄNKT AUF DIE GEWÄHRLEISTUNG DER MARKTGÄNGIG-KEIT, DER EIGNUNG FÜR EINEN BESTIMMTEN ZWECK UND DER NICHTVERLET-ZUNG VON RECHTEN DRITTER, ZUR VERFÜGUNG GESTELLT. DIE AUTOREN ODER URHEBERRECHTSINHABER SIND IN KEINEM FALL HAFTBAR FÜR ANSPRÜCHE, SCHÄDEN ODER ANDERE VERPFLICHTUNGEN, OB IN EINER VERTRAGS- ODER HAFTUNGSKLAGE, EINER UNERLAUBTEN HANDLUNG ODER ANDERWEITIG, DIE SICH AUS, AUS ODER IN VERBINDUNG MIT DER SOFTWARE ODER DER NUTZUNG ODER ANDEREN GESCHÄFTEN MIT DER SOFTWARE ERGEBEN.