Daten mit dem Netzwerkprotokoll MQTT übertragen

Fach	Anwendungsentwicklung und Programmierung
Lernfeld	LF 8: Daten systemübergreifend bereitstellen
Querverweise zu weiteren Lernfeldern des Lehrplans	LF 7: Cyber-physische Systeme ergänzen (IT-Technik) LF 9: Netzwerke und Dienste bereitstellen (IT-Systeme) Fachenglisch (z.B. Informationen aus Fachartikeln, Bedienungsanleitungen, Konfigurationsanweisungen, Dokumentation)
Zeitrahmen	10 Unterrichtstunden
Benötigtes Material	ein Computer je Schüler, IDE, Beamer oder Smartboard, UML Designtool, ggf. eigener MQTT-Broker, Schreibmaterial für Plakate, ggf. Drucker für Informationsblätter, Klebepunkte

Kompetenzerwartungen

Die Schülerinnen und Schüler …

- ermitteln für einen exemplarischen Kundenauftrag gegebene Datenquellen und analysieren diese hinsichtlich der Struktur, Zugriffsmöglichkeiten und -mechanismen.
- entwickeln Konzepte zur Bereitstellung der heterogenen Datenquellen für die weitere Verarbeitung unter Beachtung der Informationssicherheit und wenden diese an.
- Sie implementieren arbeitsteilig, ihr Konzept mit vorhandenen sowie dazu passenden Entwicklungswerkzeugen und Produkten.
- übergeben ihr Endprodukt mit Dokumentation zur Handhabung, auch in fremder Sprache, an die Kunden (ggf. in Kombination mit Fachenglisch).
- reflektieren die Eignung der eingesetzten Entwicklungswerkzeuge hinsichtlich des arbeitsteiligen Entwicklungsprozesses und die Qualität der Dokumentation.

Vorbemerkung

Aufgrund der unterschiedlichen eingesetzten Hard- und Softwareplattformen sowie der Vielzahl an verwendeten Programmiersprachen und Entwicklungsumgebungen an den unterschiedlichen Berufsschulen, ist es kaum möglich, außerhalb der theoretisch zu vermittelnden Kenntnisse eine einheitliche, für alle Schulen passende Komplettlösung darzustellen. Das nachfolgende illustrierende Beispiel bezieht sich daher auf die Microsoft .NET Umgebung, die Programmiersprache C# sowie die MQTT Bibliothek MQTTnet. Ebenso zum Einsatz kommt der frei verfügbare MQTT Broker Mosquitto (verfügbar für Windows und UNIX basierte Systeme).
Eine Übertragung auf andere Programmier- und Hardwareumgebungen (von einfachen Einplatinenrechnern oder MQTT – fähigen Sensoren über Smart Devices bis hin zum Standard PC mit unterschiedlichen Betriebssystemen) ist aber problemlos möglich. Speziell für das angewandte Transportprotokoll MQTT stehen eine ganze Reihe frei verfügbarer Softwarebibliotheken für alle gängigen Programmiersprachen und Entwicklungsumgebungen zur Verfügung (siehe Links bei den Quellen und Literaturangaben).

Aufgabe

1. Orientieren:

Der Techniker eines mittelständischen Maschinenbaubetriebs tritt mit folgender Problemstellung an Ihre Ausbildungsfirma heran:

In einem Wasserturm befindet sich ein Hochwasserbehälter, der für die Produktionsanlagen seiner Fabrik Brauchwasser bereitstellt. Der Wasserstand erreicht bei maximaler Füllung des Tanks 4.00 Meter. Im Verlauf des Tages werden verschiedene Mengen an Wasser verbraucht. Zwei Pumpen mit unterschiedlicher Förderleistung sorgen dafür, dass sich der Tank nicht leert.
Ihre Aufgabe besteht darin, eine grafische Benutzerschnittstelle zu erstellen, welche den aktuellen Wasserstand anzeigt. Ebenso sollen die Pumpen manuell aus- und eingeschaltet werden können. Der Zustand der Pumpen soll in der GUI abzulesen sein.
Da eine spätere Erweiterung des Systems mit zusätzlichen Sensoren angedacht ist (z.B. Pumpentemperatur, aktuelle Leistung der Pumpen etc.), schlägt Ihr Ausbilder eine Lösung über **MQTT** als Protokoll vor.

2. Informieren:

Die Schülerinnen und Schüler erhalten anhand einer kurzen textuellen Beschreibung (z.B. Ausschnitte aus dem Lastenheft des Kunden) einen ersten Überblick über die Pumpensteuerung und bereits vorhandene Schnittstellen.

Für ein Informationsgespräch mit dem Kunden ist zuerst eine Kurzpräsentation zur prinzipiellen Funktionsweise von MQTT am Beispiel der Pumpensteuerung zu erstellen. Dies kann in arbeitsgleicher Gruppenarbeit mit Hilfe eines Informationsblattes, bei leistungsstarken Klassen auch mit deutschen bzw. englischen Fachartikeln (siehe Quellen- und Literaturangaben) erfolgen.

Da die zu implementierende Pumpensteuerung auch außerhalb der Firma genutzt soll, drängt Ihr Ausbilder auf die Berücksichtigung von Sicherheitsmechanismen bei der Datenübertragung mit MQTT.
Die Schülerinnen und Schüler führen eine Internetrecherche zum Thema „Sichere Datenübertragung mit MQTT“ durch und klären dabei Möglichkeiten, gängige technische Umsetzungen sowie Vor- und Nachteile verschiedener Mechanismen zur Datensicherheit ab. Entsprechend den Vorkenntnissen, dem Ausbildungsberuf sowie der Fachrichtung erhalten sie dabei Unterstützung bei der Auswahl der Suchbegriffe durch die Lehrkraft.

Die Sammlung und Einordnung wichtiger Schutzmechanismen und deren technische Umsetzungsmöglichkeiten erfolgt im Unterrichtsgespräch mit Hilfe der Tafel bzw. des Smartboards. Anschließend fassen die Schülerinnen und Schüler die Ergebnisse in arbeitsteiliger Gruppenarbeit zusammen und präsentieren diese.

Der Kunde möchte von Ihrer Ausbildungsfirma genauere Informationen zu den Möglichkeiten der Datensicherung erhalten.
Zur Sicherung und Überprüfung des Lernerfolgs formuliert jedes Team Vor- und Nachteile einer gewählten Maßnahme zur Datensicherheit und berät den Kunden in einem kurzen Rollenspiel.

a) Grundlagen zum MQTT - Protokoll

MQTT (Message Queue Telemetry Transport) hat sich als Transportprotokoll speziell für die Datenübertragung im IoT etabliert.
Die Entwicklung als Protokoll zur **Maschine zu Maschine Kommunikation** (M2M) reicht bis auf ein Projekt aus dem Jahr 1999 zurück, bei dem IBM und Arcom Control Systems die Überwachung einer Ölpipeline implementierten. 2010 erfolgte die Veröffentlichung der Version 3.1 unter freier Lizenz, ab 2014 die Normung durch das not-for-profit Standardisierungsgremium OASIS bis zum aktuellen Standard 5.0 (Stand März 2019). Zahlreiche Open-Source-Implementierungen sowie die Übergabe ihrer MQTT-Bibliotheken durch IBM und Eurotech (vormals Arcom) an die Eclipse Foundation beschleunigten die weitere Verbreitung.

Kennzeichen von MQTT:

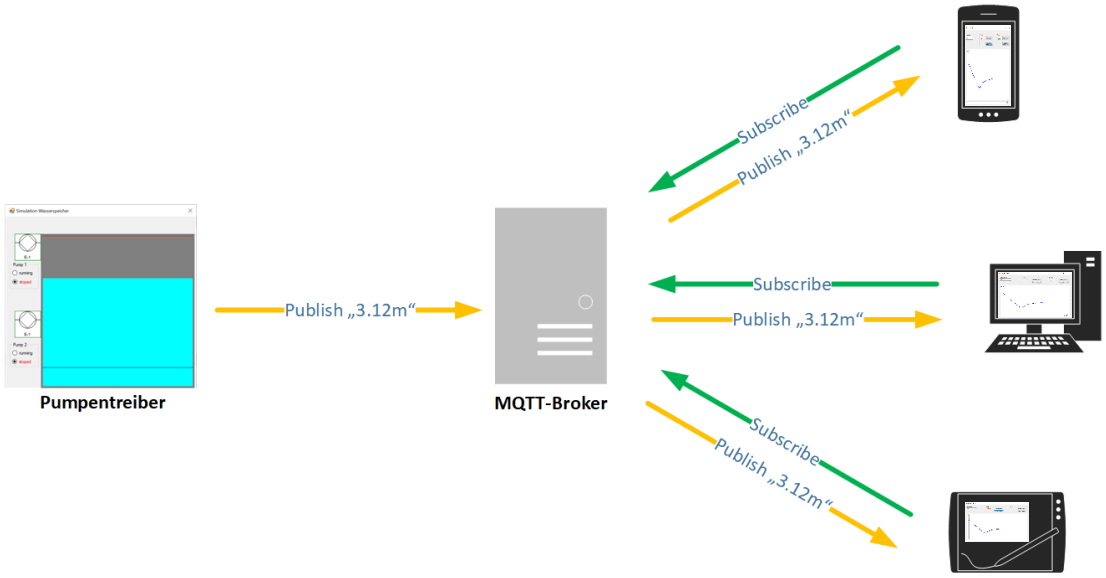
- Hardwaregeräte mit geringer Rechenleistung (z.B. Sensoren) können durch eine einfache Umsetzung mit minimaler Breitband- und Ressourcennutzung Daten untereinander austauschen.
- In instabilen Netzen wird eine Datenübertragung durch unterschiedliche QoS (Quality of Service) Einstellungen sichergestellt.
- Struktur und Aufbau der Daten spielen bei der Übertragung mit Hilfe von MQTT keine Rolle.
- Metainformationen werden serverseitig auf dem Broker gespeichert, um bei einem Verbindungsabriss nicht erneut gesendet werden zu müssen.

b) Publish/ Subscribe Pattern

MQTT arbeitet nach dem Publish/ Subscribe Pattern. Dabei wird an Stelle einer Punkt zu Punkt Verbindung ein zentraler Server (Service Broker) eingesetzt, welcher als Verbindungsknoten zwischen Datensender und Datenempfänger dient.
Zum Empfangen von Daten registriert sich ein Empfänger unter einem bestimmten **Topic** beim Broker (**subscribe**) z.B. ISBTest/Pumpensteuerung/Wasserstand. Sendet eine Datenquelle (z.B. ein Sensor) nun unter dem gleichen Topic Daten an den Broker (publish), so werden diese an alle registrierten Empfänger weitergereicht. Dabei spielt es keine Rolle, ob Sender und Empfänger voneinander wissen. Der gesamte Datenaustausch erfolgt über die Topics und den Broker, so dass Datenquellen und Empfänger unterschiedlichster Art miteinander verbunden werden können (z.B. Sensoren, mobile Endgeräte, PCs usw.).

Schema zur Pumpensteuerung

Für die Pumpensteuerung kann daher folgendes Schema skizziert werden:



(1) Die Endgeräte abonnieren (Subscribe) das Topic *ISBTest/Pumpensteuerung/Wasserstand.*

(2) Der Pumpentreiber sendet (Publish) den Wasserstand mit dem gleichen Topic *ISBTest/Pumpensteuerung/Wasserstand* an den Broker.

(3) Der Broker reicht die Daten an die eingetragenen Endgeräte weiter.

c) MQTT Topics

Topics garantieren die Flexibilität und können auch in Kombination mit vordefinierten Wildcards verwendet werden:

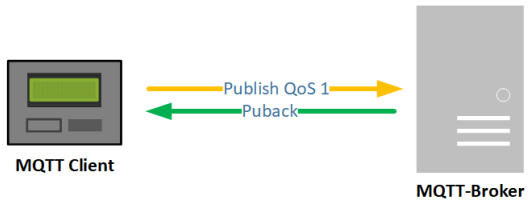
- Jeder Slash definiert eine Hierarchieebene. Das Topic sollte nicht mit einem ‚/‘ beginnen, da hierbei nur eine weitere, leere Ebene eingeführt wird:
  ISBTest/Pumpensteuerung/Wasserstand
  Der Client abonniert genau ein bestimmtes Topic.
- Das +-Zeichen symbolisiert eine Hierarchieebene
  ISBTest/+/Wasserstand
  Der Client abonniert eine Ebene unterhalb von ISBTest alle Wasserstände.
- Das #-Zeichen steht immer am Ende einer Hierarchie und symbolisiert alle folgenden Ebenen und Unterebenen:
  ISBTest/#
  Der Client abonniert alle Topics unterhalb der obersten Hierarchieebene ISBTest.

d) QoS (Quality of Service)

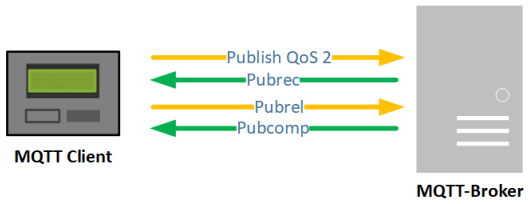
Für die Datenübertragung sind im MQTT Protokoll drei Service Level (QoS) definiert:

QoS 0:
Die Daten werden genau einmal gesendet, ohne eine Bestätigung des Empfängers zu erwarten

QoS 1:
Die Daten werden mindestens einmal gesendet. Der Sender wartet auf eine Bestätigung (Puback) und wiederholt den Vorgang falls diese fehlt.
Mehrfachversendungen sind möglich.



QoS 2:
Die Daten werden exakt einmal ausgeliefert. Dazu wird eine doppelte Empfangsbestätigung eingesetzt.



e) Mechanismen zur sicheren Datenübertragung

- *Passwortgeschützter Zugang*:
  Für den Aufbau der Datenverbindung zwischen Client und Broker kann in MQTT ein Benutzername und Passwort konfiguriert werden. Clients ohne passende Kennung erhalten keine Verbindung zum Server. Da die Datenübertragung aber im Klartext abläuft, können Benutzerkennungen und Passwörter im Netzwerkverkehr problemlos mitgelesen werden. Die Datenübertragung sollte daher mit verschlüsselten Zugangsdaten oder komplett verschlüsselt (siehe TLS) starten.
  Zusätzlich zur Authentifizierung des Clients beim Broker, sollte eine Autorisierung für bestimmte Zugriffe und Transportmöglichkeiten erfolgen. Auf Seite des Brokers ist es möglich, den Publish und Subscribe – Zugriff oder QoS Einstellungen bestimmter Clients auf genau definierte Topics per Konfiguration zu beschränken.
- *TLS (Transport Layer Security) mit X509 Certificate*

Zur Verschlüsselung des gesamten Datentransports kann das TLS Protokoll eingesetzt werden. Abfangen, Auslesen oder Manipulieren von Informationen durch Dritte (Man-In-The-Middle-Attacke) lassen sich dadurch deutlich erschweren. Vor der Aufnahme der Datenverbindung erfolgt ein TLS – Handshake zwischen Client und Broker. Der Verbindungsaufbau kann folgendermaßen beschrieben werden:

(1) Der Client startet einen Verbindungsaufbau mit dem Broker.
(2) Der Broker sendet dem Client ein X509 Zertifikat, das von einer CA (Certificate Authority) ausgestellt sein sollte.
(3) Der Client prüft die Gültigkeit des Zertifikats und bricht die Verbindung bei einem ungültigen Zertifikat ab.
(4) Bei gültigem Brokerzertifikat erstellt der Client mit Hilfe des Public Keys aus dem X509 Zertifikat einen codierten Schlüssel zur Datenübertragung und sendet diesen an den Broker.
(5) Der Key wird zur symmetrischen Verschlüsselung der folgenden Datenübertragung genutzt.

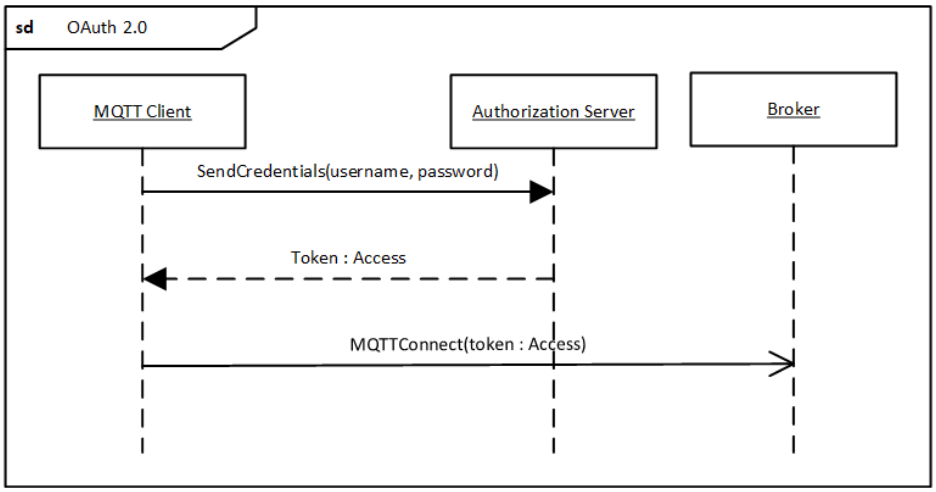
Im beschriebenen Szenario mit dem Serverzertifikat wird die Identität des Brokers sichergestellt. Erweitert man das Szenario um ein Clientzertifikat, so können auch der Client beim Broker vor der Verbindungsannahme identifiziert und der Schutz dadurch erhöht werden.

(6) Vor der Verbindungsaufnahme übersendet der Client ein X509 Zertifikat an den Broker, das mit dem gleichen Private Key wie das Serverzertifikat generiert werden muss.
(7) Der Broker identifiziert den Client anhand des Zertifikats und lehnt den Verbindungsaufbau ggf. ab.

Beim Einsatz von TLS werden Client und Broker bereits auf der Transportschicht identifiziert. Die Verbindung zu nicht authentifizierten Sitzungsteilnehmern wird vor dem Versenden des ersten Pakets abgebrochen. Als Nachteil, besonders bei größeren IoT Systemen mit tausenden Clients, erweist sich die Verwaltung der Zertifikate. X509 Zertifikate besitzen eine beschränkte Gültigkeitsdauer und müssen in regelmäßigen Zeitabständen erneuert werden. Dies gestaltet sich besonders bei ressourcenbeschränkter Hardware in ungesicherten Umgebungen als schwierig.

- *OAuth 2.0*

  OAuth 2.0 ist ein Autorisierungs-Framework und entkoppelt die Authentifizierung zwischen Client und MQTT Broker durch eine Zwischenstation, einen Authorization Server. Dieser verwaltet die Clientcredentials (z.B. Username, ClientID, Passwort etc.) und erzeugt auf Anfrage ein Token.
  Mit Hilfe des Tokens erhält der Client anschließend beim MQTT Broker Zugriff auf Ressourcen.



  Die einfache Steuerung des Ressourcenzugriffs durch den Token (z.B. zeitliche Gültigkeit, Zugriff auf Topics usw.) wiegt bei großen Systemen die Verwaltung eines extra Authorization Servers auf.

- *Verschlüsselung der Payload*

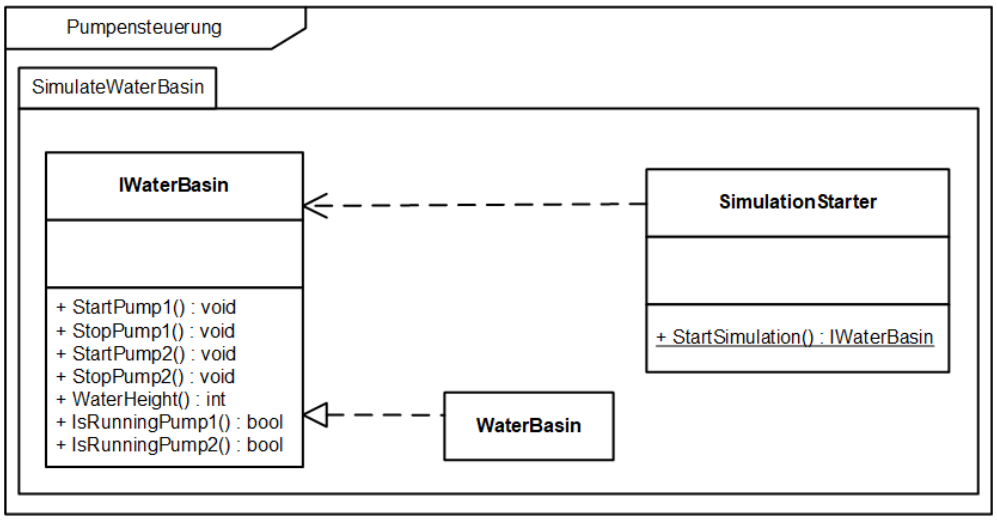
  Hardware mit beschränkter Rechenkapazität (z.B. MQTT fähige Sensoren) ermöglichen teils keine TLS Verschlüsselung bei der Datenübertragung. Hier kann es sinnvoll sein, zumindest die Payload (Nutzdaten) beim Publish Paket zu verschlüsseln. Potentielle Angriffe (Man-In-The-Middle-Attacke) lassen sich damit nicht verhindern, die übertragenen Daten können jedoch nicht ausgelesen werden. Für ressourcenbeschränkte Hardwaregeräte kommen dabei in der Regel symmetrische Verschlüsselungsverfahren zum Einsatz, da asymmetrische Verschlüsselung aus demselben Grund wie TLS ausscheidet. Sender und Empfänger müssen dazu aber über einen gemeinsamen Key (z.B. mit Hilfe eines Passwortes oder Zertifikats) zum Ver- und Entschlüsseln der Daten verfügen.

3. Planen:

Die Schülerinnen und Schüler planen ein Modell für die Implementierung der geforderten Steuerkonsole.
Für die Pumpensteuerung ist von Kundenseite aus bereits eine .NET basierte Treibersoftware eines Drittanbieters installiert, für welche die folgende Dokumentation bereitliegt:

Die Pumpensteuerung kann über das Interface IWaterBasin angesteuert werden. Die statische Methode StartSimulation( ) der Klasse SimulationStarter liefert ein initialisiertes IWaterBasin Objekt zurück und startet die Überwachung des Hochwasserbehälters.

Folgendes Klassendiagramm verdeutlicht die Funktionalität:



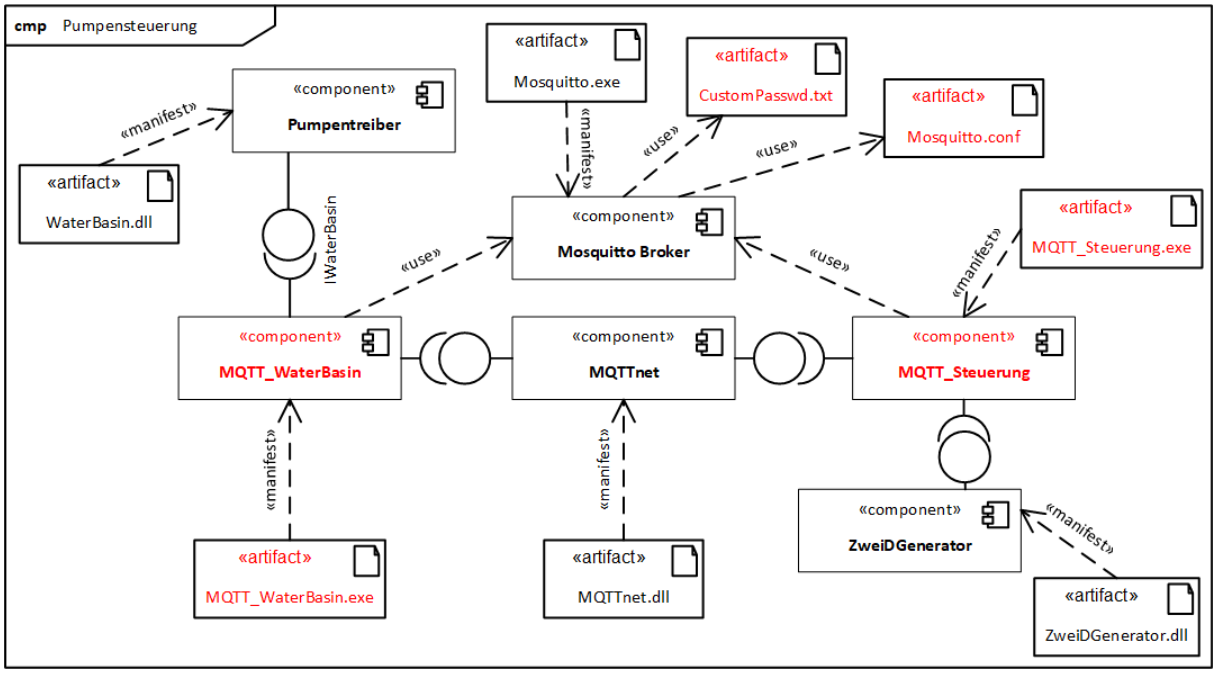
Beide Klassen liegen im Namespace „SimulateWaterBasin“.

Methoden von IWaterBasin:

StartPump1( )	*Startet die Pumpe 1.*
StartPump2( )	*Startet die Pumpe 2.*
StopPump1( )	*Stoppt die Pumpe 1.*
StopPump1( )	*Stoppt die Pumpe 2.*
WaterHeight	*Property. Liefert den aktuellen Wasserstand (0 – 400) in Zentimetern als Integer zurück*
IsRunningPump1( )	*Prüft,ob Pumpe 1 läuft. Rückgabe vom Typ bool.*
IsRunningPump2( )	*Prüft,ob Pumpe 2 läuft. Rückgabe vom Typ bool.*

Nach der Kurzanalyse der Schnittstellenbeschreibung erstellen die Schülerinnen und Schüler unter Berücksichtigung der analysierten Anforderungen in arbeitsgleichen Teams ein Komponentendiagramm für die Applikation. Dazu verwenden Sie ein computergestütztes Design- und Modellierungstool. Da die vorhandene Treibersoftware des Kunden .NET basiert ist, fällt die Wahl auf eine frei verfügbare MQTT Bibliothek für

das .NET Framework (z.B. MQTTnet oder M2Mqtt.NET). Als Broker soll der freie MQTT Server Moquitto zum Einsatz kommen, da keine besonders hohen Ansprüche an die Leistungsfähigkeit (z.B. tausende Clients) gestellt werden, sondern eher die Projektkosten im Vordergrund stehen. Dieser Broker ist für Windows und LINUX Umgebungen einsetz- und konfigurierbar.
Anschließend an eine kurze Präsentation der Ergebnisse der Modellierung erfolgt die Abstimmung auf eine gemeinsame Musterlösung im Klassenverbund.



Die rot gekennzeichneten Komponenten und Artefakte sind zu implementieren bzw. konfigurieren. Ausgehend vom Komponentendiagramm wird die Klasse in zwei Entwicklergruppen – MQTT_Steuerung und MQTT_WaterBasin – mit arbeitsgleichen Unterteams zu zwei bis drei Schülerinnen und Schülern aufgeteilt.
Die geplante Entwicklung verläuft iterativ in vier Schritten:

- Schritt 1: Funktionalität mit GUI ohne Sicherheitsmechanismen, Daten im Klartext
- Schritt 2: Authentifizierung und Autorisierung durch Passwort und Zugriffsbeschränkungen auf dem Broker
- Schritt 3: TLS Verschlüsselung der Datenübertragung
- Schritt 4: Funktionalität zur Payload-Verschlüsselung

Die Teams einigen sich auf gemeinsame Schnittstellen (Passwörter, Brokerinstallation, Topics, Datenformate) und halten diese in einer Schnittstellenbeschreibung fest.

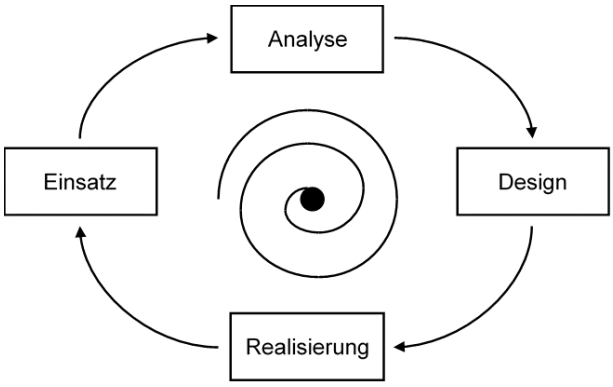
4. Durchführen

Entsprechend der Einteilung aus der Planungsphase implementieren die Schülerinnen und Schüler die geforderten Softwarebausteine mit Hilfe einer geeigneten IDE in Teamarbeit.
Synchron zum Erstellen der Programmschritte wird auf die Kommentierung des Quellcodes geachtet sowie eine grundlegende Benutzerdokumentation erstellt, welche in Kombination mit Fachenglisch auch in eine englische Version übersetzt werden kann.

Aufgrund des geringen Projektumfangs und der zeitlichen Einschränkungen im Fach Anwendungsentwicklung und Programmierung in der 11. Jahrgangsstufe wird auf den Einsatz von komplexeren Vorgehensmodellen wie beispielsweise SCRUM oder tiefergreifende Inhalte zum Projektmanagement verzichtet. Stattdessen kommt ein vereinfachtes Spiralmodell zum Einsatz.

Die Unit Tests erfolgen dabei innerhalb der Parallelgruppen in der Realisierungsphase.

Vor der Freigabe eines Zwischenreleases findet ein Integrationstest auf Ebene der gesamten Schulklasse statt.



Vertiefende Inhalte und Kompetenzen zum Projektmanagement finden sich in allen technischen IT-Berufen in der 12. Jahrgangsstufe in den Lernfeldern 12, bzw. 12a-d und bilden dort einen Schwerpunkt.

5. Kontrollieren und Bewerten

Um sicherzustellen, dass die Schülerinnen und Schüler sich nachhaltig mit dem Thema beschäftigt haben, erfolgt nach dem Ende des letzten Implementierungsschrittes des Programmierauftrags ein Review der Ergebnisse.
Neben dem Vergleich der Endergebnisse mit den Kundenanforderungen werden auch der Projektverlauf, (Schnittstellenvereinbarungen, Testverfahren, Versionskontrolle usw.) sowie die erstellte Dokumentation aus Kundensicht analysiert und von den Schülerinnen und Schülern bewertet. Dazu entwickeln diese einen Kriterienkatalog und bewerten die

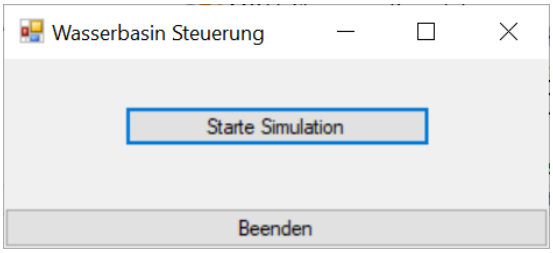
einzelnen Kriterien über eine Punktabfrage mit Klebepunkten bzw. ein digitales Abstimmungstool.
Zusätzliche Erweiterungen des bestehenden Systems mit Sensoren oder der langfristigen Speicherung und Auswertung der Daten lassen sich in einem Brainstorming finden.

Beispiele für Produkte und Lösungen der Schülerinnen und Schüler

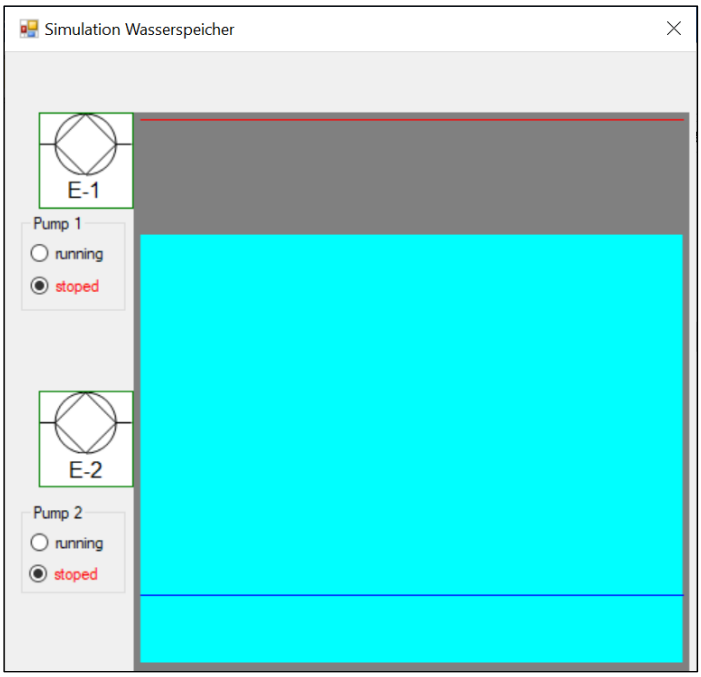
1. Kurzanleitung zur erstellten Software

Zur interaktiven Steuerung der Pumpen des Hochwasserbehälters verfügt die Anwendung über die beiden ausführbaren Dateien MQTT_WaterBasin.exe und MQTT_Steuerung.exe mit drei Dialogfenstern sowie einen MQTT Broker:

MQTT_WaterBasin.exe



Die MQTT_WaterBasin.exe startet mit einem Dialogfeld zur Verbindung mit der Pumpensteuerung über die Steuersoftware des Drittanbieters (SimulateWaterBasin.dll).
Mit der Betätigung des **Starte Simulation** – Buttons wird die Pumpensteuerung initialisiert und angezeigt. Ebenso startet die Verbindung zum MQTT_Broker mit der Übertragung der Wasserstandsdaten.
Der **Beenden** Button schließt die Pumpensteuerung.

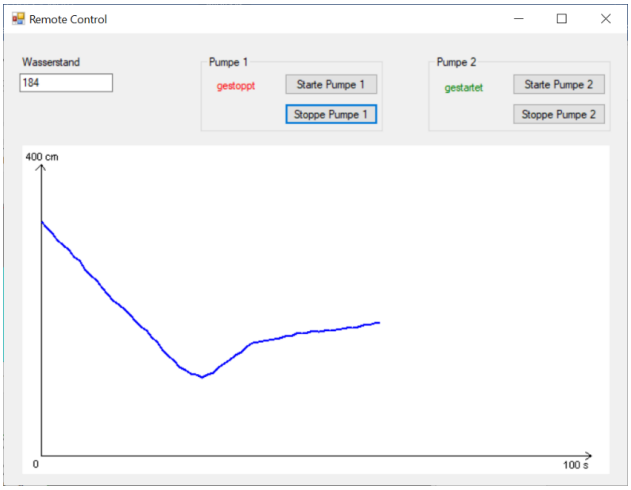


Mit der Initialisierung der Steuersoftware erscheint der Simulationsdialog zum Hochwasserbehälter des Drittanbieters.
Die enthaltene Grafik symbolisiert den Wasserstand des Basins. Der aktuelle Wasserstand wird alle 0,5 Sekunden an den MQTT Broker weitergeleitet.

Über die Optionsfelder **running** und **stoped** lassen sich die Pumpen manuell ansteuern. Die Zustände der Pumpen werden ebenfalls über den MQTT Broker an die Remote Steuerung weitergegeben und dort angezeigt.

MQTT_Steuerung.exe

Das eigentliche Kernstück der Anwendung bildet die Fernsteuerung (Remote Control) der Pumpensteuerung.



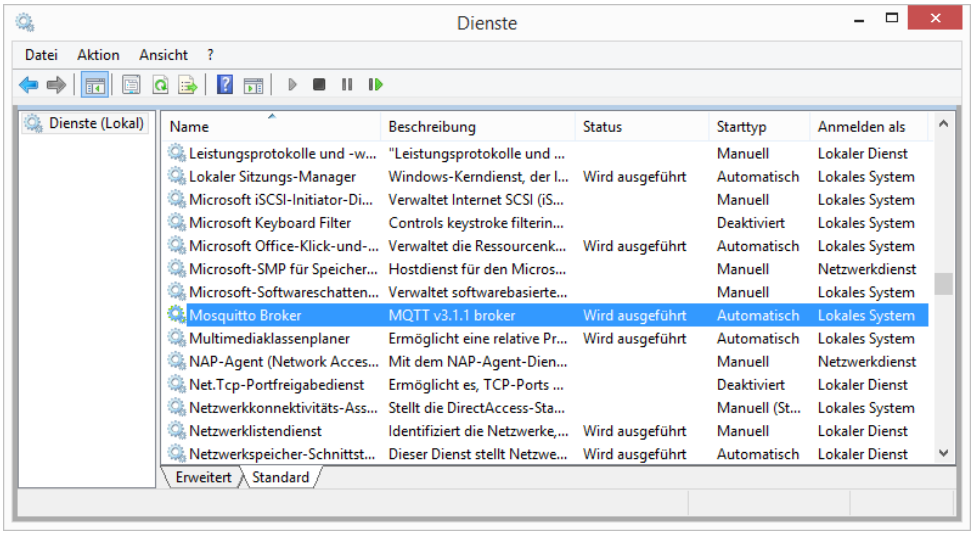
Nach dem Start der MQTT_Steuerung.exe Datei verbindet sich die Applikation mit dem MQTT Broker, zeigt den aktuellen Wasserstand in Zentimetern an und modelliert den zeitlichen Verlauf des Wasserstandes in einer Grafik. Die aktuellen Zustände der Pumpen (gestoppt/ gestartet) sind ebenfalls erfasst.

Mit den **Starte Pumpe/ Stoppe Pumpe** Buttons lassen sich die Pumpen manuell fernsteuern.

Bei vorgegebenen unteren Wasserständen (20 cm bzw. 10 cm) starten die Pumpen, um ein Leerlaufen des Behälters zu vermeiden. Bei einem Überschreiten der maximalen Wasserstände (370 cm bzw. 380 cm) stoppen die Pumpen, um den Überlauf des Basins zu verhindern.

MQTT_Broker Mosquitto

Der MQTT Broker läuft als Windowsdienst auf jedem aktuellen Windows System.



Starten oder Beenden des Service erfolgt über die Systemsteuerung/ Dienste.

Die dazu erforderlichen Passwortdateien, Konfigurationsdateien und Zertifikate werden bei der Projektübergabe mitgeliefert.

2. Schnittstellenbeschreibung:

Passwortdatei C:\mosquitto\custompasswd.txt

User	Passwort	Einsatz
ISBBasin	*ISBTest21!*	*Wasserbasin Applikation MQTT_WaterBasin.exe*
ISBRemote	*ISBTest12!*	*Remote Control Applikation MQTT_Steuerung.exe*

Topics auf dem Broker

Applikation	Topic	Einsatz
MQTT_WaterBasin.exe	*ISBTest/pumpcontrol/waterlevel*	*Publish,* *Wassertand in cm*
	ISBTest/pumpcontrol/pumpcode	*Publish,* *0 – keine Pumpe läuft* *1 – P1 läuft* *2 – P2 läuft* *3 – P1 und P2 laufen*
	ISBTest/pumpcontrol/pump	*Subscribe,* *startp1 - startet P1* *startp2 - startet P2* *stopp1 - stoppt P1* *startp2 - stoppt P2*
MQTT_Steuerung.exe	*ISBTest/pumpcontrol/waterlevel*	*Subscribe, siehe oben*
	ISBTest/pumpcontrol/pumpcode	*Subscribe, siehe oben*
	ISBTest/pumpcontrol/pump	*Publish, siehe oben*

Rechte auf Topics über acl Datei C:\mosquitto\customacl.txt

User	Topic	Recht
ISBBasin	*ISBTest/pumpcontrol/waterlevel*	*write*
	ISBTest/pumpcontrol/pumpcode	*write*
	ISBTest/pumpcontrol/pump	*read*
ISBRemote	*ISBTest/pumpcontrol/waterlevel*	*read*
	ISBTest/pumpcontrol/pumpcode	*read*
	ISBTest/pumpcontrol/pump	*write*

3. Codings und Konfigurationen

Folgende Codings basieren auf der Programmiersprache C# und der freien MQTT Bibliothek MQTTnet (siehe Quellen- und Literaturangaben). Das Arbeiten mit dem MQTT Protokoll verläuft in den Grundzügen jedoch immer gleich, so dass eine Übertragung auf andere Programmiersprachen unter Nutzung verschiedenster Bibliotheken möglich ist.

Im Prinzip sind folgende Aufgaben abzuarbeiten:

- Client konfigurieren und erstellen,
- Verbindung zum Broker herstellen,
- Topics per Subscribe abonnieren,
- Daten an Topics per Publish versenden,
- Verbindung zwischen Client und Broker am Programmende schließen.

Version 1: Funktionalität mit GUI ohne Sicherheitsmechanismen, Daten im Klartext

Codeausschnitte MQTT_WaterBasin:

```
public partial class MQTTBasainForm : Form
{
        // Steuerung des Wasserspeichers
        private SimulateWaterBasin.IWaterBasin waterBasin;
        private string strCommand;

        // Attribute für MQTT
        private IMqttClient client;
        private string clientId;
        private IMqttClientOptions options;

        // Attribut für Rücksprung in Hauptthread mit GUI zum GUI Update in .NET
        private Dispatcher mainThreadDispatcher;

        /// <summary>
        /// Konstruktor
        /// </summary>
        public MQTTBasainForm()
        {
            InitializeComponent();
            sendTimer.Interval = 500; // Timer mit 500ms für Wasserstandsmeldungen setzen

            // Dispatcher für den Rücksprung in den Hauptthread zum GUI Update
            // (muss bei Windows im Hauptthread erfolgen)
            mainThreadDispatcher = Dispatcher.CurrentDispatcher;
        }


        /// <summary>
        /// Anzeige mit Initialisierungen starten
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void startButton_Click(object sender, EventArgs e)
        {
            // Wasserspeicher initialisieren
            waterBasin = SimulateWaterBasin.SimulationStarter.StartSimulation();
            sendTimer.Start();

            // MQTT Client initialisieren
            InitMQTTClient();
        }
```

```
/// <summary>
/// MQTT Client erzeugen, Verbindung zum Broker herstellen
/// und Handler Methode für eingehende Botschaften erstellen
/// </summary>
private async void InitMQTTClient()
{
    try
    {
        // string BrokerAddress mit Properties setzen (app.config Datei) z.B test.moquitto.org
        // oder einer Adresse des eigenen Mosquitto Brokers
        string brokerAddress = MQTT_WaterBasin.Properties.Settings.Default.BrokerAddress;

        // Eindeutige Client ID erzeugen
        clientId = Guid.NewGuid().ToString();

        // Optionen für Client setzen, TCP Connection mit Port 1883 (unverschlüsselt)
        options = new MqttClientOptionsBuilder()
            .WithClientId(clientId)
            .WithTcpServer(brokerAddress, 1883)
            .WithProtocolVersion(MQTTnet.Formatter.MqttProtocolVersion.V500)
            .Build();

        // Client erzeugen
        client = new MqttFactory().CreateMqttClient();

        //client mit Broker verbinden
        // token für Abbruch der asynchronen Methode erstellen
        System.Threading.CancellationToken token;
        await client.ConnectAsync(options, token);

        // Subscribe für Pumpensteuerung
        await client.SubscribeAsync(new TopicFilterBuilder()
                                    .WithTopic("ISBTest/pumpcontrol/pump")
                                    .Build());

        // Handler für eingehende Messages auf "ISB/pumpcontrol/pump" registrieren
        //client.UseApplicationMessageReceivedHandler(OnClientMqttMessageReceived);
        client.UseApplicationMessageReceivedHandler(arg =>
       {
            // empfangene Nachricht encodieren, Payload ist Nachricht vom Broker
            string ReceivedMessage = Encoding.UTF8.GetString(arg.ApplicationMessage.Payload);

            // über Dispatcher in den Mainthread der GUI springen und dort ausführen
            mainThreadDispatcher.Invoke(delegate
            {
                    strCommand = ReceivedMessage;
                    if (waterBasin != null)
                        UpdateUI();
            });
       });
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message + " Verbunden: " + client.IsConnected,
            "Fehler beim Verbindungsaufbau", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

/// <summary>
/// Publishing bei neuem Wasserstand
/// </summary>
/// <param name="topic">Topic für Publishing</param>
/// <param name="payload">Daten für das Publishing</param>
private async void Publish(string topic, string payload)
{
    // publish mit Quality of Service exactly once...
    var message = new MqttApplicationMessageBuilder()
        .WithTopic(topic)
        .WithPayload(payload)
        .WithExactlyOnceQoS()
        .Build();

    System.Threading.CancellationToken token;
    if(client.IsConnected)
```

```
            await client.PublishAsync(message, token);
        }

        /// <summary>
        /// Applikation stoppen und Verbindung zum Broker trennen
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnStop_Click(object sender, EventArgs e)
        {
            sendTimer.Stop();
            Close();
            Dispose();
        }

        /// <summary>
        /// In Abständen von 500 Millisekunden den Waserstand und Pumpencode veröffentlichen
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        /// <returns></returns>
        private void sendTimer_Tick(object sender, EventArgs e)
        {
            // Wasserstand senden
            int waterLevel = waterBasin.WaterHeight;
            Publish("ISBTest/pumpcontrol/waterlevel", waterLevel.ToString());

            // Pumpenzustand senden
            int code = 0;
            if (waterBasin.IsRunningPump1())
            {
                code += 1;
            }
            if (waterBasin.IsRunningPump2())
            {
                code += 2;
            }
            Publish("ISBTest/pumpcontrol/pumpcode", code.ToString());
        }

        /// <summary>
        /// Benutzeroberfläche in Hauptthread aktualisieren...
        /// </summary>
        private void UpdateUI()
        {
            if(strCommand.Equals("startp1"))
                waterBasin.StartPump1();
            else if(strCommand.Equals("startp2"))
                waterBasin.StartPump2();
            if(strCommand.Equals("stopp1"))
                waterBasin.StopPump1();
            if (strCommand.Equals("stopp2"))
                waterBasin.StopPump2();
        }

        /// <summary>
        /// Beim Schließen des Forms die Verbindung zum Broker schließen
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void MQTTBasainForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Verbindung zum Broker beenden
            if (client!= null && client.IsConnected)
            {
                client.DisconnectAsync();
            }
        }
    }
```

Codeausschnitte MQTT_Steuerung:

```
public partial class RemoteForm : Form
{
        private string waterLevel = "0";
        private string pumpCode = "0";

        // Attribute für MQTT
        private IMqttClient client;
        private string clientId;
        private IMqttClientOptions options;

        // Attribut für Rücksprung in Hauptthread mit GUI zum GUI Update in .NET
        private Dispatcher mainThreadDispatcher;

        /// <summary>
        /// Konstruktor
        /// </summary>
        public RemoteForm()
        {
            InitializeComponent();
            try
            {
                // Dispatcher für den Rücksprung in den Hauptthread zum GUI Update
                mainThreadDispatcher = Dispatcher.CurrentDispatcher;

                // MQTT Client initialisieren
                InitMQTTClient();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Fehler beim Start",
                                MessageBoxButtons.OK, MessageBoxIcon.Stop);
            }

        }

        /// <summary>
        /// MQTT Client erzeugen, Verbindung zum Broker herstellen
        /// und Handler Methode für eingehende Botschaften erstellen
        /// </summary>
        private async void InitMQTTClient()
        {
            try
            {
                //string BrokerAddress mit Properties setzen (app.config Datei)
                string brokerAddress = MQTT_Steuerung.Properties.Settings.Default.BrokerAddress;

                // Eindeutige Client ID erzeugen
                clientId = Guid.NewGuid().ToString();

                // Optionen für Client setzen, TCP Connection ohne Verschlüsselung mit Port 1883
                options = new MqttClientOptionsBuilder()
                    .WithClientId(clientId)
                    .WithTcpServer(brokerAddress, 1883)
                    .WithProtocolVersion(MQTTnet.Formatter.MqttProtocolVersion.V500)
                    .Build();

                // Client erzeugen
                client = new MqttFactory().CreateMqttClient();

                // Client mit Broker verbinden
                // Token für Abbruch der asynchronen Methode erstellen
                System.Threading.CancellationToken token;
                await client.ConnectAsync(options, token);

                // Subscribe für Pumpensteuerung mit Wasserstand
                await client.SubscribeAsync(new TopicFilterBuilder()
                                            .WithTopic("ISBTest/pumpcontrol/waterlevel").Build());
                // Subscribe für Pumpenzustand
                await client.SubscribeAsync(new TopicFilterBuilder()
                                            .WithTopic("ISBTest/pumpcontrol/pumpcode").Build());
```

```
            // Handler für eingehende Messages auf "ISBTest/pumpcontrol/waterlevel" registrieren
            client.UseApplicationMessageReceivedHandler(arg =>
            {
                // Nachricht encodieren
                string ReceivedMessage = Encoding.UTF8.GetString(arg.ApplicationMessage.Payload);
                bool flag = false;
                // Wasserlevel oder Pumpe?
                if (arg.ApplicationMessage.Topic == "ISBTest/pumpcontrol/waterlevel")
                {
                    flag = true;
                }
                // über Dispatcher in den Mainthread der GUI springen und dort ausführen
                mainThreadDispatcher.Invoke(delegate
                {
                        if (flag)
                            waterLevel = ReceivedMessage;
                        else
                            pumpCode = ReceivedMessage;
                        UpdateUI(flag);
                });
            });
        }
        catch(Exception ex)
        {
            MessageBox.Show(ex.Message + " Verbunden: " + client.IsConnected,
                "Fehler beim Verbindungsaufbau", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    /// <summary>
    /// Publishing bei neuem Wasserstand zum Starten/Stoppen der Pumpen
    /// </summary>
    /// <param name="topic">Topic für Publishing</param>
    /// <param name="payload">Daten für das Publishing</param>
    private async void Publish(string topic, string payload)
    {
        // publish mit Quality of Service exactly once...
        var message = new MqttApplicationMessageBuilder()
            .WithTopic(topic)
            .WithPayload(payload)
            .WithExactlyOnceQoS()
            .Build();

        System.Threading.CancellationToken token;
        if (client.IsConnected)
            await client.PublishAsync(message, token);
    }

    /// <summary>
    /// Handler für manuellen Start der Pumpe 1
    /// </summary>
    private void pump1_Click(object sender, EventArgs e)
    {
        Publish("ISBTest/pumpcontrol/pump", "startp1");
    }

    /// <summary>
    /// Handler für manuellen Start der Pumpe 2
    /// </summary>
    private void pump2_Click(object sender, EventArgs e)
    {
        Publish("ISBTest/pumpcontrol/pump", "startp2");
    }

    /// <summary>
    /// Handler für manuellen Stop der Pumpe 1
    /// </summary>
    private void stopPump1_Click(object sender, EventArgs e)
    {
        Publish("ISBTest/pumpcontrol/pump", "stopp1");
    }
```

```
/// <summary>
/// Handler für manuellen Stop der Pumpe 2
/// </summary>
private void stopPump2_Click(object sender, EventArgs e)
{
    Publish("ISBTest/pumpcontrol/pump", "stopp2");
}

/// <summary>
/// GUI aktualisieren und Pumpensteuerungslogik
/// </summary>
/// <param name="flag">true = Wasserstand geändert, false = Pumpenzustand</param>
private void UpdateUI(bool flag)
{
    if (flag)
    {
        SetLevel();
        int waterHeight = int.Parse(waterLevel);

        // Grafik zeichnen
        draw2DGraphics1.AddPoint(waterHeight);

        // Pumpenlogik
        // Level 1 unterschritten -> Starte Pumpe 1
        if (waterHeight <= 20)
            pump1.PerformClick();
        // Level 2 unterschritten -> Starte Pumpe 2
        if (waterHeight <= 10)
            pump2.PerformClick();
        // Level 1 übererschritten -> Starte Pumpe 1
        if (waterHeight > 370)
            stopPump1.PerformClick();
        // Level 2 übererschritten -> Starte Pumpe 2
        if (waterHeight > 380)
            stopPump2.PerformClick();
    }
    else
    {
        // Pumpenzustand auf der GUI aktualisieren
        switch(int.Parse(pumpCode))
        {
            case 1:
                lblPump1.Text = "gestartet";
                lblPump1.ForeColor = Color.Green;
                lblPump2.Text = "gestoppt";
                lblPump2.ForeColor = Color.Red;
                break;
            case 2:
                lblPump1.Text = "gestoppt";
                lblPump1.ForeColor = Color.Red;
                lblPump2.Text = "gestartet";
                lblPump2.ForeColor = Color.Green;
                break;
            case 3:
                lblPump1.Text = "gestartet";
                lblPump1.ForeColor = Color.Green;
                lblPump2.Text = "gestartet";
                lblPump2.ForeColor = Color.Green;
                break;
            default:
                lblPump1.Text = "gestoppt";
                lblPump1.ForeColor = Color.Red;
                lblPump2.Text = "gestoppt";
                lblPump2.ForeColor = Color.Red;
                break;
        }
    }
}
```

```
        /// <summary>
        /// Wasserstand in der Anzeige setzen
        /// </summary>
        private void SetLevel()
        {
            txtWaterLevel.Text = waterLevel;
        }

        /// <summary>
        /// Programmende, Verbindung zum Broker schließen
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void RemoteForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Verbindung zum Broker beenden
            if (client != null && client.IsConnected)
            {
                client.DisconnectAsync();
            }
        }
    }
```

Einstellungen für den Mosquitto Broker:

Für den Mosquitto Broker sind in Version 1 keinerlei Einstellungen notwendig. Für den Testlauf kann neben dem eigenen Service Broker ggf. sogar ein freier Broker (z.B. test.mosquitto.org oder broker.hivemq.com) verwendet werden.

Version 2: Authentifizierung und Autorisierung durch Passwort und Zugriffsbeschränkungen auf dem Broker

Ab Version 2 werden nur noch die zusätzlichen Änderungen im Code angegeben, welche für die zusätzliche Funktionalität erforderlich sind.

Codeausschnitte MQTT_WaterBasin:

```
                // Usercredentials mit Username und Passwort erzeugen
                // Diese werden in Version 4 über eine verschlüsselte Credentials Datei gelesen
                MqttClientCredentials credit = new MqttClientCredentials();
                credit.Username = "ISBBasin";
                credit.Password = Encoding.UTF8.GetBytes("ISBTest21!");

// Optionen für Client setzen, TCP Connection mit Port 1883 (unverschlüsselt)
                options = new MqttClientOptionsBuilder()
                    .WithClientId(clientId)
                    .WithTcpServer(brokerAddress, 1883)
                    .WithCredentials(credit) // Credentials Objekt für Broker übergeben
                    .WithProtocolVersion(MQTTnet.Formatter.MqttProtocolVersion.V500)
                    .Build();

                // Client erzeugen
                ...
```

Codeausschnitte MQTT_Steuerung:

```
                 // Usercredentials mit Username und Passwort erzeugen
                // Diese werden in Version 4 über eine verschlüsselte Credentials Datei gelesen
                MqttClientCredentials credit = new MqttClientCredentials();
                credit.Username = "ISBRemote";
                credit.Password = Encoding.UTF8.GetBytes("ISBTest12!");

                // Optionen für Client setzen, TCP Connection ohne Verschlüsselung mit Port 1883
                options = new MqttClientOptionsBuilder()
                    .WithClientId(clientId)
                    .WithTcpServer(brokerAddress, 1883)
                    .WithCredentials(credit) // Credentials Objekt für Broker übergeben
                    .WithProtocolVersion(MQTTnet.Formatter.MqttProtocolVersion.V500)
                    .Build();

                // Client erzeugen
```

Für die passwortgeschützte Authentifizierung und Autorisierung erstellt man in beiden Applikationen ein zusätzliches MqttClientCredentials Objekt. Dieses wird bei den Optionen des Clients gesetzt.
Vorsicht! Die Datenübertragung erfolgt immer noch über Port 1883 in Klartext. Sowohl die Daten als auch Benutzerkennung und Passwort sind lesbar.

Konfigurationsdateien im Ordner des Mosquitto MQTT Brokers

(1) Passwort:
Zum Erstellen der Passwortdatei verwendet man das mitgelieferte Kommandozeilentool mosquitto_passwd.exe. Das Tool erzeugt eine Passwortdatei (z.B. custompwd.txt) mit verschlüsseltem Passwort:

```
C:\mosquitto>mosquitto_passwd custompwd.txt ISBBasin
```

```
…
ISBBasin:$6$ZFHXq01kCzmJNH+Z$u0f2uPjfupjHs3hYzKHqJletVBLgLVL6awsO87BPbocUdj3ngsBVaUgdfL/FRzn5jc38eQ1ZRunr7nPjPkb9sg==
ISBRemote:$6$GyEYLlIsEBnwkeIA$J3mzMO+0x7IzLDvEeVvRXNb+74JA4ee6Z750gOKU4rag4nHivik70ZRkbpK23nerB6HIvD6v/9v5Zu4G8aOoYA=
```

Für die Autorisierung verfügt der Mosquitto Broker die Möglichkeit, eine acl Datei zu generieren. Hier werden angemeldete Clients (siehe Passwortdatei) über die Topics mit Rechten verbunden (Beispieldatei customacl.txt):

```
# Rechte für Remote Control
user ISBRemote

topic read ISBTest/pumpcontrol/waterlevel
topic read ISBTest/pumpcontrol/pumpcode
topic write ISBTest/pumpcontrol/pump

#Rechte für Basin App
user ISBBasin

topic write ISBTest/pumpcontrol/waterlevel
topic write ISBTest/pumpcontrol/pumpcode
topic read ISBTest/pumpcontrol/pump
```

In der mosquitto.config Datei im Verzeichnis des MQTT Servers sind dann die entsprechenden Konfigurationseinstellungen zu setzen. Das #-Zeichen wird für Zeilenkommentare verwendet:

```
...
# =================================================================
# Security
# =================================================================
...
# Boolean value that determines whether clients that connect
# without providing a username are allowed to connect.
# ...
allow_anonymous false
# ----------------------------------------------------------------
# Default authentication and topic access control
# ----------------------------------------------------------------

# Control access to the broker using a password file. This file can be
# generated using the mosquitto_passwd utility.
# ...
password_file c:\mosquitto\custompwd.txt
# ...
```

```
# Control access to topics on the broker using an access control list
# file. If this parameter is defined then only the topics listed will
# have access.
# ...
acl_file c:\mosquitto\customacl.txt
...
```

Nach dem Neustart des MQTT Dienstes gilt die neue Konfiguration.

Version 3: TLS Verschlüsselung der Datenübertragung

Für den TLS-Handshake sind Zertifikate notwendig, die normalerweise kostenpflichtig über eine CA (Certificate Authority) ausgestellt werden. Beim Testlauf genügen aber selbsterstellte Zertifikate, welche sich über Makecert.exe, die Windows Powershell oder das Freeware-Tool Open-SSL generieren lassen.

Codeausschnitte MQTT_WaterBasin/ MQTT_Steuerung:

Die Codeänderungen zur Verschlüsselung der Datenübertragung über TLS sind für die Basin Applikation und deren Steuerung identisch.

```
// Client - Zertifikate für Übergabe an MQTT Broker generieren
// ca.crt CA Zertifikat muss mit ca.cert auf MQTT Broker übereinstimmen...
// ca.crt in "Vertrauenswürdige Stammzertifizierungsstellen" (Root) installieren
// client.crt in "Zwischenzertifizierungsstellen" (CertificateAuthority) installieren
// client.pfx in "Eigene Zertifikate" (My) installieren

// Rootzertifikat aus Speicher auslesen
X509Store store = new X509Store(StoreName.Root, StoreLocation.CurrentUser);
store.Open(OpenFlags.OpenExistingOnly | OpenFlags.ReadOnly);
X509Certificate rootCert;
rootCert = store.Certificates.Find(X509FindType.FindBySubjectName, "FE12C01", false)[0];
store.Close();

// Clientzertifikat aus Speicher auslesen
store = new X509Store(StoreName.CertificateAuthority, StoreLocation.CurrentUser);
store.Open(OpenFlags.OpenExistingOnly | OpenFlags.ReadOnly);
X509Certificate clientCert;
clientCert = store.Certificates.Find(X509FindType.FindBySubjectName, "MqttTestClient", false)[0];
store.Close();

// TLS Parameterobjekt mit Zertifikaten konfigurieren
MqttClientOptionsBuilderTlsParameters tlsParam = new MqttClientOptionsBuilderTlsParameters()
{
       UseTls = true,
       IgnoreCertificateChainErrors = false,
       IgnoreCertificateRevocationErrors = false,
       AllowUntrustedCertificates = false,
       SslProtocol = SslProtocols.Tls12,

       // Liste der Übergabezertifikate des Clients an den MQTT Broker Mosquitto
       Certificates = new List<X509Certificate> { new X509Certificate2(clientCert),
       new X509Certificate2(rootCert),},

       // Callback für Überprüfung des Serverzertifikats:
       CertificateValidationCallback = (X509Certificate x, X509Chain y, SslPolicyErrors z,
                                        IMqttClientOptions o) =>
       {
          // x wird vom Broker an den Client gesendet
          // Gültigkeit des vom Server übergebenen X509 Zertifikat prüfen
          X509Certificate2 cert = new X509Certificate2(x);

          // bei von gültigen CAs - ausgestellten Zertifikaten die .NET Bibliothek verwenden
          // Windows überprüft anhand des Zertifikatspeichers
          // return cert.Veryfy();

          // bei selbsterstellten Testcertifikaten ohne echte CA
          // eigene Logik mit bekannten Inhalten verwenden...
          // Serverzertifikat des Brokers in "Zwischenzertifizierungsstellen"
```

```
            // (CertificateAuthority) installieren!
            // Serverzertifikat des Brokers aus Zertifikatsspeicher auslesen...
            store = new X509Store(StoreName.CertificateAuthority, StoreLocation.CurrentUser);
            store.Open(OpenFlags.OpenExistingOnly | OpenFlags.ReadOnly);
            X509Certificate2 serverCert;
            serverCert = store.Certificates.Find(X509FindType.FindBySubjectName, "FE12C01", false)[0];
            store.Close();

            if (cert.Subject == serverCert.Subject && cert.Issuer == serverCert.Issuer
                && cert.NotAfter > System.DateTime.Today && cert.Thumbprint == serverCert.Thumbprint)
            {
                return true;
            }
            return false;
        }
};

                ...

// Optionen für Client setzen, TCP Connection mit Port 1883 (unverschlüsselt)
                // oder Port 8883 (TSL verschlüsselt) und tlsParam setzen
options = new MqttClientOptionsBuilder()
                .WithClientId(clientId)
                .WithTcpServer(brokerAddress, 8883) // TCP Port auf 8883 ändern
                .WithCredentials(credit)
                .WithTls(tlsParam) // TLS Parameter übergeben
                .WithProtocolVersion(MQTTnet.Formatter.MqttProtocolVersion.V500)
                .Build();

                // Client erzeugen
```

Schlüsel und X509 Zertifikate generieren

Für die Zertifikate und Keys können Tools wie Makecert.exe, die Windows Powershell oder das Freeware-Tool Open-SSL verwendet werden. Folgende Schritte sind auszuführen:

Serverzertifikate:

(1) Erstellen eines CA Schlüsselpaares mit Passwort (ca.key),

(2) Erstellen eines CA X509 - Zertifikats basierend auf dem Schlüssel des 1. Arbeitsschritts (ca.crt),

(3) Erstellen eines Broker Schlüsselpaares ohne Passwort (server.key),

(4) Erstellen eines Broker X509 - Zertifikats basierend auf dem Schlüssel des 3. Arbeitsschritts (server.crt),

(5) Signieren des Brokerzertifikats (Schritt 4) mit dem CA Zertifikat aus dem 2. Arbeitsschritt,

(6) Die vier Zertifikate und Schlüssel (ca.crt, ca.key, server.crt, server.key) sind in einen Unterordner des MQTT Brokers Mosquitto zu kopieren z.B. Certificates). Das CA Zertifikat und ggf. das Serverzertifikat (falls nicht von einer offiziellen CA erstellt) wird später ebenso auf dem Client zur Authentifizierung des Servers beim TLS Handshake benötigt.

Für das Erzeugen der Clientzertifikate muss der gleiche CA Key wie beim Serverzertifikat verwendet werden, da Client und Server beim TLS – Protokoll verschlüsselt kommunizieren.

Clientzertifikate:

(7) Erstellen eines Client Schlüsselpaares ohne Passwort (client.key),

(8) Erstellen eines Client X509 - Zertifikats basierend auf dem Schlüssel des 7. Arbeitsschritts (client.crt),

(9) Signieren des Clientzertifikats (Schritt 8) mit dem CA Zertifikat aus dem 2. Arbeitsschritt (gleiche CA wie beim Serverzertifikat),

(10) Erstellen einer PFX - Datei durch Kombination von Client Key und Clientzertifikat (client.pfx),

(11) Alle drei Dateien müssen an den Client verteilt und in den entsprechenden Zertifikatsspeichern (bei Windows Systemen) installiert werden.

Konfiguration Mosquitto

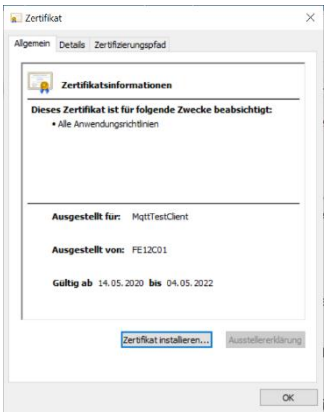
In der mosquitto.config Datei im Verzeichnis des MQTT Servers sind weitere Konfigurationseinstellungen für die Verwendung von TLS und den Zertifikaten zu setzen. Im folgenden Ausschnitt sind die Änderungen auf TLS und Port 8883 beschrieben.

```
...
# ================================================================
# Default listener
# ================================================================
...
# Port to use for the default listener.
port 8883

# Choose the protocol to use when listening.
protocol mqtt
...
# ----------------------------------------------------------------
# Certificate based SSL/TLS support
# ----------------------------------------------------------------
# The following options can be used to enable SSL/TLS support for
# this listener.
...
# At least one cafile or capath must be defined. They both
# define methods of accessing the PEM encoded Certificate
# Authority certificates that have signed your server certificate
# and that you wish to trust.
...
cafile c:\mosquitto\Certificates\ca.crt

# Path to the PEM encoded server certificate.
certfile c:\mosquitto\ Certificates \server.crt

# Path to the PEM encoded keyfile.
keyfile c:\mosquitto\ Certificates \server.key
...
# By setting require_certificate to true,
# the client must provide a valid certificate in order for the network
# connection to proceed.
...
require_certificate true
...
# This option defines the version of the TLS protocol to use for this listener.
...
tls_version tlsv1.2
```

Zertifikate auf dem Client installieren

Für die Funktionalität der MQTT Clients sind die oben beschriebenen Zertifikate bei Windows Systemen an bestimmten Stellen im Zertifikatsspeicher zu installieren. Andere Plattformen und Bibliotheken nutzen die Clientzertifikate analog beim TLS Verbindungsaufbau.

Zertifikat	Speicherort
ca.cert	aktueller Benutzer, „Vertrauenswürdige Stammzertifikate“
client.crt	Aktueller Benutzer, automatische Auswahl des Stammzertifikatsspeichers („Zwischenzertifizierungsstellen“)
client.pfx	Aktueller Benutzer, automatische Auswahl des Stammzertifikatsspeichers („Eigene Zertifikate“)
cerver.crt	Aktueller Benutzer, automatische Auswahl des Stammzertifikatsspeichers („Zwischenzertifizierungsstellen“)



Versionsion 4: Funktionalität zur Payload-Verschlüsselung

Im letzten Durchgang wird die Verschlüsselung der Payload mit Hilfe eines symmetrischen Verschlüsselungsverfahrens implementiert. Die Verschlüsselung findet ebenso bei der Sicherung des MQTT - Client Passwortes ihren Einsatz.

Der eingesetzte Key zur symmetrischen Verschlüsselung der Payload wird über den Private Key des Clientzertifikats (client.pfx) generiert. Leistungsstärkere MQTT Clients könnten aber über Public und Private Key auch ressourcenintensivere asymmetrische Verschlüsselungsverfahren anwenden.

Für die Verschlüsselung wird die eigene statische C# Klasse CustomSecurityClass mit den Methoden GetEncrypted(), GetDecrypted und ReadPWD() generiert.

Codeausschnitte CustomSecurityClass:

```
// Attribute für den symmetrischen Algorithmus
private static Rfc2898DeriveBytes rgb;
private static AesManaged algorithm;
private static byte[] rgbKey;
private static byte[] rgbIV;

/// <summary>
/// Konstruktor
/// </summary>
static CustomSecurityClass()
{
   // Client - Zertifikat für Key
   // client.pfx in "Eigene Zertifikate" (My) installieren
   // Clientzertifikat aus Speicher auslesen
   X509Store store = new X509Store(StoreName.My, StoreLocation.CurrentUser);
   store.Open(OpenFlags.OpenExistingOnly | OpenFlags.ReadOnly);
   X509Certificate2 cert;
   cert = store.Certificates.Find(X509FindType.FindBySubjectName, "MqttTestClient", false)[0];
   store.Close();

   // private Key des Zertifikats auslesen
   byte[] key = ((RSACryptoServiceProvider)cert.PrivateKey).ExportCspBlob(false);

   // Key zur Erstellung des Zufallsgenerators für den Symmetrischen Schlüssel
   // und IV (Initialization Vector) verwenden
   rgb = new Rfc2898DeriveBytes(key, Encoding.Unicode.GetBytes("ISBSalt"), 10);

   // Algorithmusobjekt erstellen
   algorithm = new AesManaged();

   // Key und IV in passender Größe (Byte) per Zufallsgenerator erstellen
   rgbKey = rgb.GetBytes(algorithm.KeySize / 8);
   rgbIV = rgb.GetBytes(algorithm.BlockSize / 8);
}

/// <summary>
/// Verschlüsselt einen String und liefert die verschlüsselten
/// bytes als Bytearray zurück
/// </summary>
/// <param name="message">zu verschlüsselnder Text</param>
/// <returns>Bytearry mit verschlüsselter Message</returns>
public static byte[] GetEncrypted(string message)
{
   // Verschlüsselungsobjekt erstellen
   var encryptor = algorithm.CreateEncryptor(rgbKey, rgbIV);
   byte[] ret;
   byte[] bytesToTransform = Encoding.UTF8.GetBytes(message);

   // Bytes über einen MemoryStream und über den CrytoStream verschlüsseln
   using (MemoryStream buffer = new MemoryStream())
   {
      using (CryptoStream cryptoStream = new CryptoStream(buffer, encryptor, CryptoStreamMode.Write))
      {
         // Bytes verschlüsseln und in Stream schreiben
         cryptoStream.Write(bytesToTransform, 0, bytesToTransform.Length);
         cryptoStream.FlushFinalBlock();
         ret = buffer.ToArray();
         ret = buffer.ToArray();
      }
   }
   return ret;
}
```

```
/// <summary>
/// Entschlüsselt ein Bytearray und liefert die entschlüsselten
/// bytes als String zurück
/// </summary>
/// <param name="message">zu entschlüsselnde Bytes</param>
/// <returns>String mit entschlüsselter Message</returns>
public static string GetDecrypted(byte[] message)
{
   // Entschlüsselungsobjekt erstellen
   var decryptor = algorithm.CreateDecryptor(rgbKey, rgbIV);
   string ret = null;

   // Bytes über einen StreamReader und über den CrytoStream entschlüsseln
   using (MemoryStream buffer = new MemoryStream(message))
   {
      using (CryptoStream cryptoStream = new CryptoStream(buffer, decryptor, CryptoStreamMode.Read))
      {
         using (StreamReader srDecrypt = new StreamReader(cryptoStream))
         {
            // Bytes entschlüsseln und in String speichern
            ret = srDecrypt.ReadToEnd();
         }
      }
   }
   return ret;
}

/// <summary>
/// Liest ein verschlüsseltes Passwort aus einer Datei
/// </summary>
/// <param name="path">Pfad der Passwortdatei</param>
/// <returns>entschlüsseltes Passwort als Bytearray</returns>
public static byte[] ReadPWD(string path)
{
   byte[] b;

   // Entschlüsselungsobjekt erstellen
   var decryptor = algorithm.CreateDecryptor(rgbKey, rgbIV);

   // Bytes über einen FileStream und über den CrytoStream entschlüsseln
   using (FileStream file = new FileStream(path, FileMode.Open, FileAccess.Read))
   {
      b = new byte[file.Length];
      using (CryptoStream cryptoStream = new CryptoStream(file, decryptor, CryptoStreamMode.Read))
      {
         // Bytes entschlüsseln und in Bytearray speichern
         cryptoStream.Read(b, 0, b.Length);
      }
   }
   return b;
}
```

Codeausschnitte MQTT_WaterBasin/ MQTT_Steuerung:

Für die MQTT Clients werden neben dem Einlesen des verschlüsselten Client - Passwortes über die Dateien BasinPassWD bzw. ControlPassWD auch die Handler Methoden zu den erhaltenen Nachrichten vom Broker sowie die Publish Methoden abgeändert.

```
...
// Usercredentials mit Username und Passwort erzeugen
// Passwort wird über ControlPassWD Datei gelesen
MqttClientCredentials credit = new MqttClientCredentials();
credit.Username = "ISBRemote";
credit.Password = CustomSecurityClass.ReadPWD("ControlPassWD");
...

...
// Handler für eingehende Messages auf "ISBTest/pumpcontrol/…" registrieren
client.UseApplicationMessageReceivedHandler(arg =>
```

```
{
   // empfangene Nachricht per CustomSecurity Klasse entschlüsseln
   string ReceivedMessage = CustomSecurityClass.GetDecrypted(arg.ApplicationMessage.Payload);
...

...
// publish mit Quality of Service exactly once...
var message = new MqttApplicationMessageBuilder()
   .WithTopic(topic)
   // Nachricht verschlüsseln
   .WithPayload(CustomSecurity.CustomSecurityClass.GetEncrypted(payload))
   .WithExactlyOnceQoS()
   .Build();
...
```

Konfiguration Mosquitto
An der Konfiguration des Brokers sind keine weiteren Änderungen notwendig. Zusätzlich zum eingestellten TLS- Datentransport werden die übertragenen Daten verschlüsselt.

4. Ergebnissicherung

Zur Überprüfung der neuerworbenen Kompetenzen bearbeiten die Schüler **Verständnisfragen** und weitergehende **Fallbeispiele zur Datenübertragung durch das MQTT - Protokoll** z.B.:

1. Erläutern Sie das Publish/ Subscribe Pattern beim Einsatz des Übertragungsprotokolls MQTT!
   Beim MQTT Protokoll werden Nachrichten über sogenannte Topics auf einem MQTT – Broker ausgetauscht. Der Broker empfängt und versendet Nachrichten und verbindet Datenquellen und Empfänger unterschiedlichster Art miteinander. Das Datenformat spielt dabei keine Rolle. Um Nachrichten für ein bestimmtes Topic zu erhalten, abonniert der Client ein entsprechendes Topic (Subscribe). Per Publish auf das gleiche Topic kann jeder beliebige Client dann Nachrichten über den Broker an den empfangenden Client weiterleiten.

2. Beschreiben Sie die QoS Modi beim MQTT – Protokoll! Weshalb wird für die Steuerung des Hochwasserbehälters QoS exactly once verwendet?
   Das MQTT – Protokoll definiert drei QoS (quality of service) Level:
   - *QoS 0: Die Daten werden genau einmal gesendet, ohne eine Bestätigung des Empfängers zu erwarten,*
   - *QoS 1: Die Daten werden mindestens einmal gesendet. Der Sender wartet auf eine Bestätigung (Puback) und wiederholt den Vorgang falls diese fehlt. Mehrfachversendungen sind möglich.*

- *QoS 2: Die Daten werden exakt einmal ausgeliefert. Dazu wird eine doppelte Empfangsbestätigung eingesetzt.*

Bei der Steuerung des Hochwasserbehälters werden die Pumpen anhand des übermittelten Wasserstands gestartet oder gestoppt. Durch die Übertragung ohne Bestätigung könnte ein Leerlaufen oder Überlaufen des Basins eintreten.

3. Erklären Sie den prinzipiellen Aufbau einer TLS - Verbindung zwischen MQTT - Client und MQTT - Broker!

   (1) Der MQTT - Client baut eine TLS - Verbindung über Port 8883 zum Broker auf.

   (2) Der MQTT - Broker präsentiert sein Zertifikat.

   (3) Der Client überprüft dieses mit Hilfe des installierten CA-Zertifikats auf Echtheit.

   (4) Danach erfolgt die nur für den Broker lesbare Übertragung des zufällig erzeugten Schlüssels durch den Client.

   (5) Mit dem nun auf beiden Seiten vorhandenen Schlüssel kann eine symmetrische Datenverschlüsselung beginnen.

   (6). Diese symmetrische Verschlüsselung verursacht eine geringere Prozessorlast als die zum Übertragen der Schlüssel eingesetzten unsymmetrischen Verfahren.

   (7) Je nach Konfiguration überträgt auch der Client ein Zertifikat an den Broker, um seine Identität zu garantieren.

4. Ändern Sie die CustomSecurityClass zur Nutzung eines asymmetrischen Verschlüsselungsverfahrens ab!

   Für die asymmetrische Verschlüsselung kann das bereits vorhandene Clientzertifikat client.pfx mit public und private Key verwendet werden. Über den public Key werden die Daten verschlüsselt, der private Key entschlüsselt die Daten. Wichtig ist, dass Sender und Empfänger über das gleiche Zertifikat mit dem Schlüsselpaar verfügen.

```
...
// Client - Zertifikat für Key
// client.pfx in "Eigene Zertifikate" (My) installieren
// Clientzertifikat aus Speicher auslesen
X509Store store = new X509Store(StoreName.My, StoreLocation.CurrentUser);
store.Open(OpenFlags.OpenExistingOnly | OpenFlags.ReadOnly);
X509Certificate2 cert;
cert = store.Certificates.Find(X509FindType.FindBySubjectName, "MqttTestClient", false)[0];
store.Close();
...
// public Key des Zertifikats zum Verschlüsseln
var encryptor = (RSACryptoServiceProvider)cert.PublicKey.Key;
byte[] encryptedBytes = encryptor.Encrypt(dataBytes, true);
...
// private Key des Zertifikats zum Entschlüsseln
var decryptor = (RSACryptoServiceProvider)cert.PrivateKey;
byte[] decryptedBytes = decryptor.Encrypt(encryptedBytes, true);

...
```

5. Stellen Sie Vor- und Nachteile von symmetrischen und asymmetrischen Verschlüsselungsverfahren in einer Tabelle gegenüber!

Symmetrische Verschlüsselung

Vorteil	*Nachteil*
• *keine Begrenzung der zu verschlüsselnden Datenmenge* • *schnell und ressourcensparend*	• *nur ein Schlüssel ist für Ver- und Entschlüsselung in Verwendung, wird dieser gehackt, sind Daten öffentlich zugänglich* • *bei Verwendung verschiedener Keys für die Verschlüsselung wird die Verwaltung der Keys sehr aufwendig*

Asymmetrische Verschlüsselung

Vorteil	*Nachteil*
• *basiert auf einem Schlüsselpaar, durch die Verteilung der Schlüssel kann der Zugriff auf die Daten einfach geregelt werden* • *längere Keys, schwieriger durch brute force Attacken zu brechen*	• *Einschränkung die der Größe der übertragbaren Daten* • *Verschlüsselung läuft im Vergleich zur symmetrischen Verschlüsselung sehr langsam (hoher Ressourcenverbrauch)*

6. Erstellen Sie einen MQTT basierten Chat Client. Ihre Nachrichten sollten dazu natürlich verschlüsselt werden!
   Die Schülerinnen und Schüler implementieren einen Chat Client auf Basis des MQTT – Protokoll inklusive Verschlüsselung der Payload (übertragene Nachrichten). Jeder Client kann per Publish und Subscribe am Chat teilnehmen. Topic und ggf. Brokeradresse sollten nicht hart codiert, sondern frei wählbar sein.

Hinweise zum Unterricht

Der Schwerpunkt der skizzierten Lernsituation liegt auf dem Kennenlernen und Anwenden des Transportprotokolls MQTT durch die Schülerinnen und Schüler. Dabei lassen sich Daten unterschiedlichster Datenquellen erfassen. Ebenso werden Mechanismen der Datensicherheit implementiert und bewertet.
Die Simulation des Wasserbehälters inklusive Pumpen wird von der Lehrkraft zur Verfügung gestellt. Die Wasserentnahme simuliert ein Zufallszahlengenerator. Zur grafischen Darstellung des Wasserstandes im Koordinatensystem kommt eine Bibliotheksdatei (ZweiDGenerator.dll) zum Einsatz, welche von den Schülerinnen und Schülern bereits in einer vorangehenden Lernsituation entwickelt wurde. Diese arbeitet mit dem Device Context (Graphics Object) und lässt sich in unterschiedlichsten Programmiersprachen nachstellen.
Für die verschiedenen Fachrichtungen/ Berufe sollten neben den grundlegenden Kompetenzen zusätzliche Schwerpunkte herausgearbeitet werden:

Fachinformatikerin/Fachinformatiker Fachrichtung Anwendungsentwicklung:

- Erweiterung der Logik um Fehlerprüfungen bei der Datenübertragung, Nutzen von zusätzlichen Funktionalitäten des Protokolls MQTT (last will, etc.),
- GUI für Smart Devices entwickeln.

Fachinformatikerin/Fachinformatiker Fachrichtung Systemintegration, IT-System-Elektronikerin/ IT-System-Elektronikerin:

- Aufsetzen, Konfigurieren und Verwalten eines eigenen MQTT Service Brokers (z.B. Mosquitto) auf unterschiedlichen Betriebssystemen (z.B. Windows, LINUX).

Fachinformatikerin/Fachinformatiker Fachrichtung Daten- und Prozessanalyse:

- Erweiterung der Logik zum langfristigen Speichern und Analysieren der übertragenen Daten z.B. in einer Datenbank,
- Ermitteln, Auswerten und Präsentieren von Möglichkeiten der Prozessverbesserung (z.B. Strategie zur optimalen Auslastung der Pumpen).

Fachinformatikerin/Fachinformatiker Fachrichtung Digitale Vernetzung:

- Erweiterung der Hard- und Software um zusätzliche Sensoren aus der Automatisierungstechnik (z.B. Motortemperatur).

Querverweise zu anderen Fächern / Fachrichtungen

Die vorgestellte Lernsituation lässt sich problemlos mit den Fächern IT – Technik (Lernfeld 7: Cyber-physische Systeme ergänzen) sowie IT-Systeme (Lernfeld 9: Netzwerke und Dienste bereitstellen) verknüpfen. Das MQTT – Protokoll spielt im IoT eine gewichtige Rolle und bildet eine Schnittstelle zwischen klassischer Anwendungsentwicklung und Cyber-physischen Systemen. Jedes MQTT-fähige Endgerät kann an der Datenübertragung teilnehmen. Themengebiete wie TLS-Handshake, Datenverschlüsselung, Erstellen und Bereitstellen von Zertifikaten wiederum finden sich im Bereich der Netzwerke und Dienste. Speziell der Umgang mit Zertifikaten oder der Handshake sollte über das Fach IT-Systeme bereits in einer größeren Tiefe erarbeitet werden, so dass in der skizzierten Lernsituation aus dem Fach Anwendungsentwicklung und Programmierung auf fundierte Kenntnisse zurückgegriffen werden kann.

In Fachenglisch lassen sich Informationen zur Funktionsweise von MQTT aus englischen Texten eruieren oder englische Texte verfassen (z.B. Informationen aus Fachartikeln, Bedienungsanleitungen, Konfigurationsanweisungen des Brokers, Dokumentation der Software).

Beim allgemeinbildenden Unterricht finden sich Verbindungspunkte bei der Gestaltung von Softwaredokumentationen für den Kunden (Deutsch) oder der Bewertung der Auswirkungen von Digitalisierung und dem IoT (Sozialkunde).

Quellen- und Literaturangaben

- https://www.informatik-aktuell.de/betrieb/netzwerke/mqtt-leitfaden-zum-protokoll-fuer-das-internet-der-dinge.html#c21362 (Zugriff 15-05-2020 15:39 MESZ)
- https://www.heise.de/developer/artikel/MQTT-Protokoll-fuer-das-Internet-der-Dinge-2168152.html?seite=all (Zugriff 15-05-2020 15:40 MESZ)
- http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html (Zugriff 15-05-2020 15:41 MESZ)
- https://m.heise.de/developer/artikel/Sichere-IoT-Kommunikation-mit-MQTT-Teil-1-Grundlagen-3645209.html?seite=all (Zugriff 15-05-2020 15:41 MESZ)
- https://m.heise.de/developer/artikel/Sichere-IoT-Kommunikation-mit-MQTT-Teil-2-Weitere-Sicherungsmassnahmen-3803338.html?seite=all (Zugriff 15-05-2020 15:42 MESZ)
- https://www.eclipse.org/paho/ (Zugriff 15-05-2020 15:43 MESZ)
- https://www.hivemq.com/mqtt-5/ (Zugriff 15-05-2020 15:43 MESZ)
- https://github.com/chkr1011/MQTTnet (Zugriff 15-05-2020 15:44 MESZ)
- https://mosquitto.org/ (Zugriff 15-05-2020 15:44 MESZ)
- http://www.steves-internet-guide.com/mosquitto-tls/ (Zugriff 15-05-2020 15:45 MESZ)