Hochschule RheinMain
Fachbereich DCSM - Informatik
Prof. Dr. Georg Hinkel

# Verteilte Systeme
**SS 2024**
**LV 4132**

**Übungsblatt 9**
**Praktische Übungen**
**Abgabe: 30.06.2024, 23:59 Uhr MESZ**

## Aufgabe 9.1 (Projekt „Eine IoT-Anwendung für das Hamsterasyl aka. Hamster-Instrumentierung“, 20 Punkte):

Das neue Management des westhessischen Hamsterverwahrungsunternehmen war sehr zufrieden mit der neunen REST-basierten Schnittstelle und hat niemanden gefeuert. Stattdessen wurde das inzwischen recht große Unternehmen von einem kleinen Zwei-Mann-Start-Up aufgekauft.

Das Start-Up ist (bzw. soll nach eigener Überzeugung werden) der weltweit führende Anbieter für Kleintier-Management und macht natürlich IoT (“The Internet of Things”). Die beiden Gründer haben BWL und Marketing studiert (einer hat sogar einen Schein in “Informatik” und schon mal Google gepingt, weswegen er natürlich der CTO ist) und viel Investorengeld eingesammelt nur leider noch keinen Anwendungsfall (und kein Produkt, keine Entwickler, keine Technologie …). Da kommt ihnen das uns bekannte westhessische Hamsterverwahrungsunternehmen und die dort arbeitenden Entwickler gerade recht, insbesondere nachdem sie feststellen mussten, dass Embedded-Entwickler (die braucht man für IoT wohl) stark nachgefragt, schwer zu bekommen sind und dann oftmals sogar mehr Geld verdienen, als ein CEO eines weltweit führenden Start-Ups. Mit dem Geld der zweiten Finanzierungsrunde das eigentlich für Billardtische und Bällchenbäder vorgesehen war, wird die alteingesessene Firma und damit auch Sie als deren Mitarbeiter übernommen.

Natürlich kommt auf Sie jetzt die nach Meinung der neuen Geschäftsleitung recht triviale Aufgabe zu, ein IoT-Produkt zu entwickeln, bzw. genauer ein richtiges IoH aufzubauen (“Internet of Hamsters”). Jeder Hamster soll jetzt an das IoH angebunden werden und 24/7 überwacht und gemanagt werden können. Dazu soll die typische IoT-Technologie MQTT[1] verwendet werden.

### Zusammenfassung der Aufgabe

- Entwickeln Sie eine IoT-Anwendung, um die abgegeben Hamster überwachen zu können. Verwenden Sie dazu das IoT-Protokoll MQTT.
- Die Überwachung des Hamsters soll über Konsoleneingaben simuliert werden. Der Code zum Parsen der dafür notwendigen Befehle ist in den Vorlagen bereits enthalten.

[1] Einen guten Einstieg in MQTT finden Sie neben den Vorlesungsfolien auch in folgendem Artikel: `https://www.heise.de/developer/artikel/MQTT-Protokoll-fuer-das-Internet-der-Dinge-2168152.html?seite=all`

- Die Anwendung stellt für den jeweiligen Hamster einige Status-Informationen zur Verfügung und es gibt einen Rückkanal um dem Hamster-Instrumentator Befehle geben zu können.

- Nachrichten für den eigenen Hamster sollen auf der Konsole ausgegeben werden.

- Die Topics sind bei MQTT in "Namespaces" gegliedert (Verzeichnisbaum/URI-artig). Es gibt zwei Topics für alle und dann mehrere Topics für jeden Hamster. Die Hamster werden über die bekannte HamsterID (aus der Lib) identifiziert.

- MQTT bietet die Möglichkeit Benutzer über Namen und Passwort authentifizieren zu lassen. Um die Komplexität zu reduzieren verwenden alle IoT-Anwendung den selben Usernamen und kein Passwort (User: "hamster").

- Es gibt einen zentralen MQTT-Broker für die Aufgabe (`hamsteriot.local.cs.hs-rm.de`), aber auch eine Testsuite, die für jeden Test einen lokalen Broker startet.

- Jeder MQTT-Client benötigt eine eindeutige ID. Diese kann automatisch generiert oder übergeben werden. Verwenden Sie für diese MQTT-Client-ID die ID des jeweilig betroffenen Hamsters (Hamster anlegen, oder eine Zufallszahl nehmen).

- Sie können aber auch lokal einen Broker starten, in der Vorlage ist im Verzeichnis /tests ein einfacher Broker auf Basis von .NET Core MQTTnet enthalten. Diesen können Sie für Testzwecke mit dem Befehl `dotnet tests/TestServer.dll` starten.

- Neben der unverschlüsselten Verbindung zu dem MQTT-Broker (std. Port 1883) gibt es die Möglichkeiten sich verschlüsselt über TLS zu verbinden (std. Port 8883), und dass der Client sich zertifikatsbasiert beim Server authentifiziert

- Über den Programmparameter "`-e`" soll die Option für die Verbindung über TLS und Zertifikate aktiviert werden und über "`-c` die Authentifizierung des Clients.

- Verwenden Sie für die verschlüsselte Verbindung das bereit gestellte Zertifikat der CA (`mqtt_ca.crt`).

- Verwenden Sie OpenSSL um für die IoT-Anwendung ein RSA Schlüsselpaar anzulegen und dafür ein CSR (Certificate Signing Request) zu erzeugen.

- Schicken Sie den CSR an ihren Praktikumsleiter. Dieser signiert diesen und schickt Ihnen das Ergebnis. Importieren Sie dieses in ihren Schlüssel und verwenden Sie dieses für die Authentifizierung.

### Die vorgesehenen MQTT-Topics

Es gibt eine Reihe von vorgesehenen MQTT-Topics für diese Aufgabe. Diese sind in Namespaces unterteilt, wobei die Hamster-ID Teil des Namespaces ist.

Der Server beschränkt die Nutzung von Topics. Jedes erlaubte Topic kann Lese- und Schreibrechte haben. Hinter "ACL" sind die Rechte für die beiden User "hamster" und "admin" aufgeführt.

MQTT bietet verschiedene Quality of Service (QoS) Optionen. Hinter QoS sind die geforderten Eigenschaften aufgeführt.

`/pension/livestock` Jeder neue Hamster publiziert hier seine ID . ACL: hamster-rw, admin-rw. QoS: Ein neu hinzukommender Subscriber soll direkt die Nachricht bekommen, aber nur genau einmal.

`/pension/hamster/{ID}/wheels` Wenn der Hamster in einem Rad läuft, wird hier die Anzahl an Runden publiziert. ACL: hamster-rw, admin-rw QoS: Best-Effort

`/pension/hamster/{ID}/state` Aktueller Zustand des Hamsters. Publizierung bei Änderungen. ACL: hamster-rw, admin-rw QoS: Gesicherte Zustellung mindestens einmal. States:

- RUNNING
- SLEEPING
- EATING
- MATEING

`/pension/hamster/{ID}/position` Aktuelle Position des Hamsters in der Pension. ACL: hamster-rw, admin-rw QoS: Gesicherte Zustellung, mindestens einmal. Positions:

- A
- B
- C
- D

`/pension/hamster/{ID}/fondle` Rückkanal zum jeweiligen Hamster. Anzahl an Belohnungen in Form von Streicheleinheiten. ACL: hamster-r, admin-rw. QoS: Gesicherte Zustellung, genau einmal.

`/pension/hamster/{ID}/punish` Rückkanal zum jeweiligen Hamster. Anzahl an Bestrafungen. ACL: hamster-r, admin-rw. QoS: Gesicherte Zustellung, genau einmal.

`/pension/room/{A,B,C,D}` Publiziert eine Liste der Hamster-IDs die in diesem Raum sind. Basiert auf der korrekten Publizierung in `/pension/hamster/{ID}/position`. ACL: hamster-r, admin-rw. QoS: best effort

Hierbei ist der Payload für *fondle* und *punish* jeweils eine 32bit Zahl im Big Endian Format, in allen anderen Fällen eine Zeichenkette mit UTF-8 bzw. ASCII-Encoding.

### Tests mit dem Mosquitto-Broker

Der mosquitto-MQTT-Broker bietet generische Publish- und Subscribe-Anwendungen. Diese sollten installiert sein. Mit `mosquitto_sub` können Sie beliebige Topics abonnieren und die Nachrichten auf der Konsole ausgeben und mit `mosquitto_pub` Nachrichten an Topics schicken.

Der MQTT-Broker ist so konfiguriert, dass nicht beliebige Topics erzeugt und verwendet werden können. Zum Testen gibt es aber das Topic "`/Test`".

Subscriben Sie das Topic mit `mosquitto_sub -t "/Test" -h hamsteriot.local.cs.hs-rm.de`

In einer zweiten Konsole können Sie dann Nachrichten an das Topic schicken mit: `mosquitto_pub -t "/Test" -h hamsteriot.local.cs.hs-rm.de -m "NACHRICHT"`

Im Default-Fall wird eine Client-ID für die MQTT-Endpunke erzeugt. Für die Aufgabe verwenden Sie die IDs der Hamster (oder eine eigene Zufallszahl).

Beginnen Sie als nächstes, den Client aufzusetzen. Senden Sie Daten an das Test-Topic und überprüfen Sie dies mit dem fertigen Subscriber-Programm.

Probieren Sie danach das Subscriben des Topics im Clients aus (Achtung, dies sollte man ein- bzw. ausschalten können.)

### Tests mit dem TestServer

Im Unterschied zum Mosquitto-Server ist der Testserver so programmiert, dass er sämtliche eingehenden Nachrichten annimmt, keine Authentifizierung benötigt und einfach sämtliche empfangenen Nachrichten auf der Konsole ausgibt. Wenn Sie mit dem TestServer arbeiten, benötigen Sie also kein fertiges Tool, um die Funktionsweise der veröffentlichten Nachrichten zu testen.

### Testsuite

Für dieses Übungsblatt haben Sie wieder eine Testsuite, die dieses mal jedoch in C# geschrieben ist und nicht als Kompilat vorliegt. Daher benötigen Sie zum Ausführen der Tests ein installiertes .NET SDK 7. Dieses ist für alle gängigen Plattformen verfügbar.

Sie können die Tests generell mit dem Befehl `dotnet test` starten. Die Ausführung der Tests variiert je nach Programmiersprache etwas. Am besten schauen Sie für den konkreten Aufruf der Testsuite wieder in die Datei `.gitlab-ci.yml`, entfernen Sie lokal aber den Flag `--no-restore`, dieser ist nur auf GitLab notwendig, um ein besseres Caching-Verhalten der heruntergeladenen Pakete zu erreichen.

### Hamster-Status publishen

Sehen Sie vor, dass wenn ihr Programm gestartet wird, die Hamster-ID in dem geforderten Topic publiziert wird.

Denken Sie daran, dass Sie bei der Verbindung mit dem Mosquitto-Broker den Usernamen mit angeben müssen!

Überprüfen Sie die Funktionalität mit dem fertigen Subscriber-Programm.

Setzen Sie dann die entsprechende QoS-Klasse.

Die Anwendung soll interaktiv einen Hamster “simulieren” können. Zu diesem Zweck ist in der Vorlage bereits eine Terminal-Struktur implementiert. Sie müssen nur noch die eigentliche Verbindung zum Broker implementieren.

Setzen Sie dies für die Topics wheels, state und position um. Überprüfen Sie dies mit den fertigen Subscriber-Programm.

### Hamster-Control subscriben

Erweitern Sie ihr Programm, so dass Nachrichten für ihren Hamster aus den Control-Topics sinnvoll auf der Konsole ausgegeben werden. In der Vorlage sind bereits Methoden für eine geeignete Konsolenausgabe enthalten.

Verwenden Sie das fertige Publisher-Programm um dies zu testen. Denken Sie dabei daran, dass dieses den User “admin” verwenden muss.

### Verschlüsselte Kommunikation mit TLS

Der Broker nimmt auch mit TLS verschlüsselte Verbindungen auf Port 8883 an. Damit der Client weiß, dass es der richtige Server ist, wird eine PKI (public key infrastructure) basierend auf X.509 Zertifikaten verwendet. Eine CA (Certificate Authority) hat das Zertifikat des Servers signiert. Wenn Sie dieser CA vertrauen, dann können Sie feststellen ob der Server auch vertrauenswürdig ist.

Erweitern Sie ihr Programm, so dass eine verschlüsselte Verbindung zum Broker aufgebaut wird (einzuschalten über das `-e` Programmparameterflag). Verwenden Sie das CA-Zertifikat, um sicherzustellen, dass die Verbindung sicher ist.

Überprüfen Sie mit geeigneten Mitteln die Datenkommunikation. Werden die Nachrichten verschlüsselt übertragen?

Erzeugen Sie ein eigenes CA-Zertifikat und verwenden Sie dieses. Wird erkannt, dass dem Server nicht vertraut werden kann?

### Client-Authentifizierung mit X.509

Mittels X.509 Zertifikation kann der Client sich auch beim Server authentifizieren. Hierzu bietet der Mosquitto MQTT-Broker auf Port 8884 einen weiteren Zugang. Der Testserver (und der Server der Testsuite) akzeptiert clientseitige Zertifikate auch auf Port 8883.

Erzeugen Sie für ihren Client ein RSA-Schlüsselpaar mit mindestens 2048 Bit Länge und entsprechenden X.509 Zertifikat. Füllen Sie die notwendigen Angaben sinnvoll aus. Das Setzen eines Passworts ist optional.

Erzeugen Sie dann für ihr Client-Zertifikat ein CSR (Certificate Signing Request) und schicken Sie dieses ihrer/ihrem Lehrbeauftragten. Diese/dieser signiert es mit dem privaten CA-Schlüssel und gibt ihnen das Ergebnis.

Um den CSR zu erstellen, können Sie beispielsweise OpenSSL installieren und die folgenden Befehle auf der Kommandozeile ausführen:

```
openssl genrsa -out client.key
openssl req -new -key client.key -out client.csr
```

Sie können alternativ auch direkt das Subject angeben, in dem Sie den Kommandozeilenparameter `-subj` beim Erstellen des CSR anfügen. Falls Sie keines angeben, werden Sie von OpenSSL nach den einzelnen Feldern gefragt.

Dokumentieren Sie ihr Vorgehen. Legen Sie ihre erzeugten Zertifikatsdateien in den Ordner `certs` ab und fügen Sie diese dem Git-Repo hinzu.

Erweitern Sie ihre Anwendung so, dass diese ihr Zertifikat für die Authentifizierung gegenüber dem Broker verwendet. Dies soll über das Programmparameterflag `-c` aktiviert werden können.

### Hinweise C

Damit Sie Ihr Programm gegen die Eclipse Paho-Bibliothek kompilieren können, müssen Sie zunächst wieder erst die Bibliothek lokal bauen. Hierfür bietet der Quellcode der Paho-Bibliothek eine geeignete Unterstützung durch CMake an.

Für die Anwendung soll die Client-Bibliothek des Eclipse Paho-Projekts verwendet werden. Machen Sie sich also mit der API-Dokumentation der Paho-C-API vertraut `https://www.eclipse.org/paho/files/mqttdoc/MQTTClient/html/`

In der Dokumentation sind Beispiele aufgeführt. Probieren Sie diese aus.

Für die Aspekte der Verschlüsselung und Authentifizierung schauen Sie sich die API-Doku für die Struktur `MQTTClient_SSLOptions` an und wie sie in den Testprogrammen des Paho-Projekts verwendet wird (`https://github.com/eclipse/paho.mqtt.c/`).

Die Paho C Bibliothek verwendet für die Verschlüsselung Zertifikate und Schlüssel im PEM-Format.

### Hinweise Java

Für Java existiert ebenfalls eine Version der Eclipse Paho Bibliothek. Diese steht im Maven-Repository zur Verfügung und ist in der Vorlage bereits integriert. Leider sind die Möglichkeiten, eigene CAs anzugeben in Paho jedoch standardmäßig sehr begrenzt, es gibt nur einen sehr allgemeinen Erweiterungspunkt. Um diesen zu bedienen, gibt es in der Vorlage die Klasse `TlsUtil`, die eine passende `SocketFactory` für angegebene CA- und Clientzertifikate bereitstellt. Auch diese Klasse verwendet für die Verschlüsselung Zertifikate und Schlüssel im PEM-Format.

In der Java Vorlage gibt es auch bereits eine Klasse `SimulatedHamster`, die Methoden anbietet um Belohnungen und Bestrafungen so auf die Konsole zu schreiben, wie die Testsuite diese Ausgaben erwartet. Außerdem ist bereits ein Timer implementiert, der sobald gestartet alle 100ms einen Callback ausführen kann. Diesen Callback können Sie verwenden, um zu simulieren, dass der Hamster seine Runden dreht.

Bitte beachten Sie, sollten Sie testweise die verwendeten Server-Zertifikate durch eigene Zertifikate ersetzen wollen, Java besteht unsinnigerweise darauf, dass in den Zertifikaten ein Subject Alternative Name (SAN) gesetzt ist.

### Hinweise C#

Für C# verwendet die Vorlage die Bibliothek MQTTnet, mit der sich neben MQTT-Clients auch Server recht einfach implementieren lassen, wie Sie in der Implementierung der Testsuite und des Testservers sehen können. Diese Bibliothek ist in der Vorlage bereits eingebunden.

Auch in der C# Vorlage gibt es bereits eine Klasse `Hamster`, die Methoden anbietet um Belohnungen und Bestrafungen so auf die Konsole zu schreiben, wie die Testsuite diese Ausgaben erwartet. Außerdem ist bereits ein Timer implementiert, der sobald gestartet alle 100ms einen Callback ausführen kann. Diesen Callback können Sie verwenden, um zu simulieren, dass der Hamster seine Runden dreht.

Beachten Sie bitte, dass .NET unter Windows nicht mit sogenannten Ephemeral Keys umgehen kann und Sie daher das PEM-Format für die Zertifikate nicht verwenden können. Stattdessen müssen Sie die Zertifikate, die Sie von Ihrem Lehrbeauftragten zurück bekommen, noch nach PKCS12 konvertieren. Auch das können Sie gut über OpenSSL bewerkstelligen:

```
openssl pkcs12 -export -inkey client.key -in client.crt -out client.pfx
```