Verteilte Systeme: Protokoll

Anforderungsanalyse

Projekt:	*Anwendung aus dem Bereich des Logistik bzw. Supply Chain Monitoring*
Autoren:	*Anastasia Galeyev, Cheryl Yamoah*
Version:	*5.2*
Letzte Änderung:	*26.01.2022*

Inhaltsverzeichnis

Ziel des zu entwickelnden Systems

Im Rahmen des Praktikums Verteilte Systeme soll eine Anwendung aus dem Bereich des Logistik bzw. Supply Chain Monitoring entwickelt werden. Dazu sollen die Technologien Sockets, Remote Procedure Calls (RPCs) sowie Message Oriented-Middleware (MOM) verwendet werden.

Anforderungen

Funktionale Anforderungen	Nichtfunktionale Anforderungen
Termin 2 1. Sensoren erfassen Information von Ein- und Auslieferungen 2. Sensoren erfassen Information von Inventarisierungen (RFID-Leser oder Barcode-Leser) 3. Produktanzahl ändert sich korrekt mit jeder Ein-, Auslieferung oder Inventarisierung 4. Sensoren leiten Information an Lager mittels UDP weiter 5. Nachrichten vom Sensor werden auf der Standardausgabe (IP, Port, Sensor-Typ) ausgegeben 6. Nachrichten von Sensoren werden in einer Log-Datei festhalten	1. Information, die von Sensoren erfasst wird = Information, welche vom Lager ausgegeben und gespeichert wird → Test durch Vergleich 2. 1 simulierte Stunde = 4 Echtzeit Sekunden, 1 simulierter Tag = 96 Echtzeit Sekunden → Messen (z.B durch Standardausgaben) 3. In Log-Datei wird die Information mit einem Datum/einem Zeitpunkt gespeichert
Termin 3 1. HTTP-Server liest HTTP-GET Anfragen korrekt und vollständig ein und verarbeitet diese 2. HTTP-Server kann auf Sensordaten zugreifen 3. Einzelne Sensordaten können aufgerufen werden (mit eigener URL) 4. Historie der Sensordaten kann aufgerufen werden (mit eigener URL) 5. Zentrale kommuniziert gleichzeitig mit Sensoren und HTTP-Clients	1. URL-Format für einzelnen Sensor: /Sensor-ID/index 2. URL-Format für Historie: /Sensor-ID/all 3. Wenn die Sensor-ID oder der URL nicht existiert wird eine Hilfeseite aufgerufen 4. Chrome und Firefox Browser werden unterstützt
Termin 4 1. Lager übermitteln aktuelle Sensordaten über thrift an andere Lager 2. Ein Lager kann Produktbestand eines anderen Lagers abfragen 3. Es können Lieferungen gemacht werden	1. Die gelieferten Lagerdaten stimmen mit den eigentlichen Lagerdaten überein 2. Wenn eine Lieferung beantragt wird, erfasst der Sensor dies und die Produktanzahl wird angepasst 3. Wenn eine Lieferung nicht möglich ist, da z.B. ein Produkt nicht mehr vorhanden ist, wird dies mitgeteilt 4. Jedes Lager speichert die Daten persistent → Datenbank/Log-Datei (?)

Termin 5 1. Lager können andere Lager abonnieren 2. Lager können die Produkte anderer Lager einsehen 3. Lager können ihre Daten mit MQTT veröffentlichen und übermitteln	1. Lagerdaten = Zusammenfassung der Sensordaten 2. Daten werden mit MQTT veröffentlicht und übertragen 3. Wichtige Daten: Welche Daten sind nicht mehr im Bestand ?

Randbedingungen

- Verwendete Programmiersprache: C++
- Verwendetes Kommunikationsprotokoll: Apache Thrift
- Verwendeter MQTT Broker: Mosquitto
- Alle Komponenten können in Docker gestartet werden (Make File) und laufen als eigenständige Container

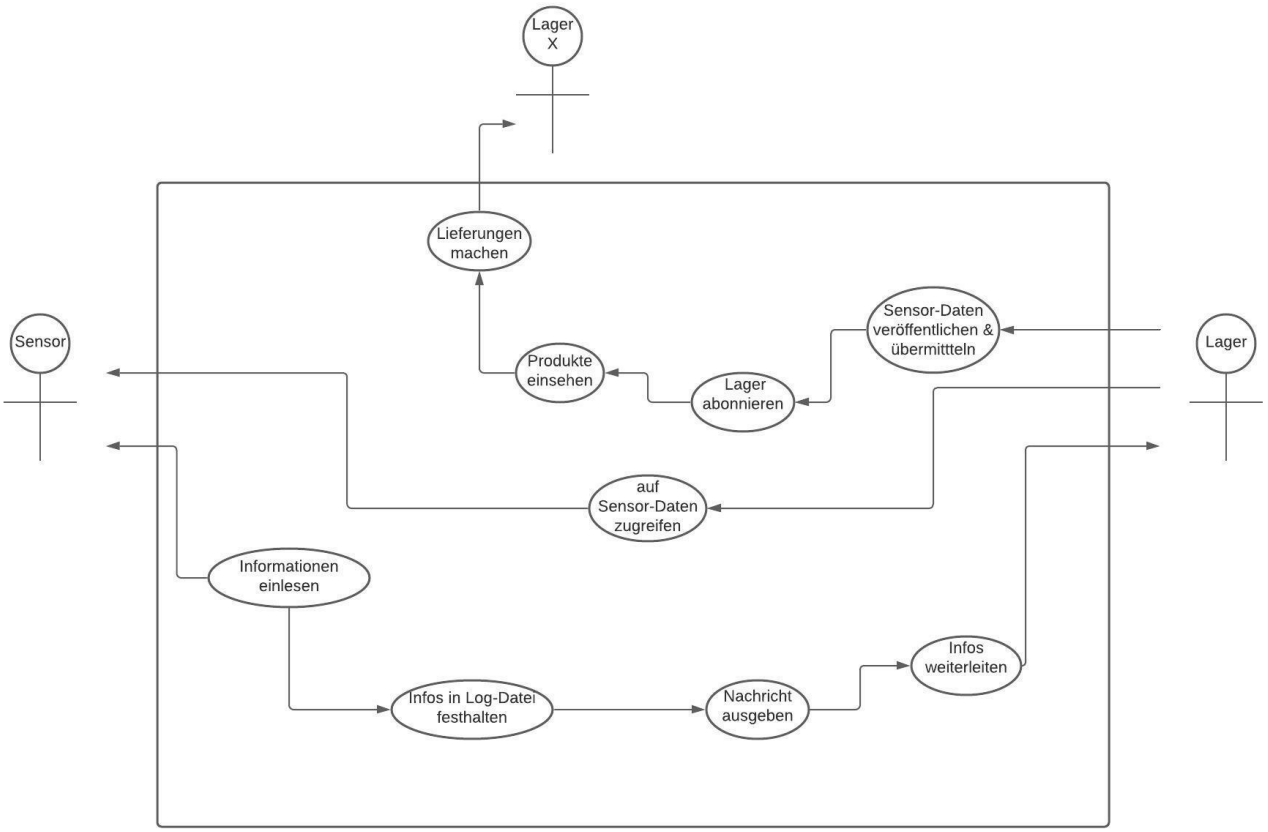
Änderungsübersicht

Version	Datum	Bearbeiter	Beschreibung
Eindeutige Versionsnummer	*Datum, an dem die Änderung abgeschlossen wurde*	*Mitarbeiter, welche(r) die Änderung durchgeführt hat*	*Allgemeine Beschreibung was geändert wurde*
2.0	08.11	Anastasia Galeyev	Sensor-Funktion *infoErfassen()* wurde verworfen, stattdessen wurde ein Vektor mit allen Produktinfos implementiert. Sensor-Funktionen *infoSenden(int), nachrichtAusgabe(string), nachrichtSpeichern(string)* wurden verworfen, da diese überflüssig sind.

2.1	08.11	Cheryl Yamoah	Lager-Funktion *datenSenden()* sind überflüssig, da das Lager keine Nachrichten an den Sensor sendet. Funktionen *processSensorInfo(), printSensorInfo()* wurden in Lager ergänzt. Funktion udp() wurde im Lager implementiert, um Nachrichten vom Sensor zu erhalten.
3.0	22.11	Cheryl Yamoah	HTTP-Server wurde implementiert, welcher *HTTP-GET* Befehle verarbeiten kann.
4.0	12.01.2022	Anastasia Galeyev & Cheryl Yamoah	Puffer-Termin verwendet -> Verschiebung der Aufgaben
4.1	12.01.2022	Anastasia Galeyev	Methode *orderProduct()*, um bestellen zu können wurde implementiert. Methode *addStock()* um den Lagerinhalt zu aktualisieren wurde implementiert.
4.2	12.01.2022	Cheryl Yamoah	Aufsetzen einer Thrift-File und einer Shell-File. Methode *seeProducts()*, um den Lagerinhalt anzugucken wurde implementiert. Methode *saveStock(),* um den Lagerinhalt zu speichern wurde implementiert.
5.0	25.01.2022	Anastasia Galeyev	Implementierung von dem MQTT-Subscriber.

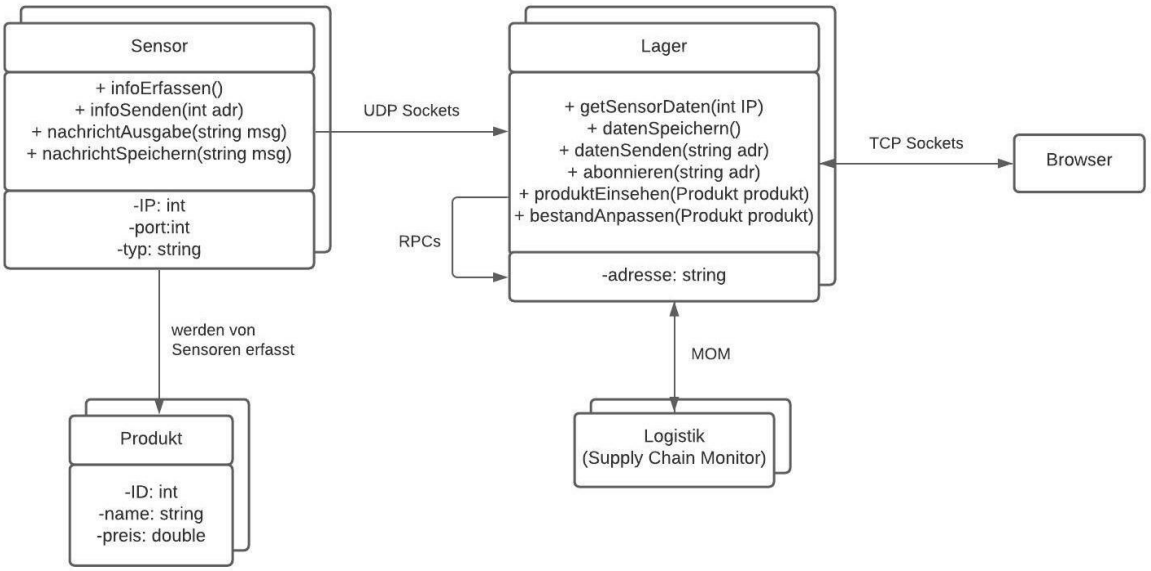
5.1	25.01.2022	Cheryl Yamoah	Implementierung von dem MQTT-Publisher.
5.2	26.01.2022	Anastasia Galeyev & Cheryl Yamoah	Implementierung von Tests

Use-Case-Diagramm

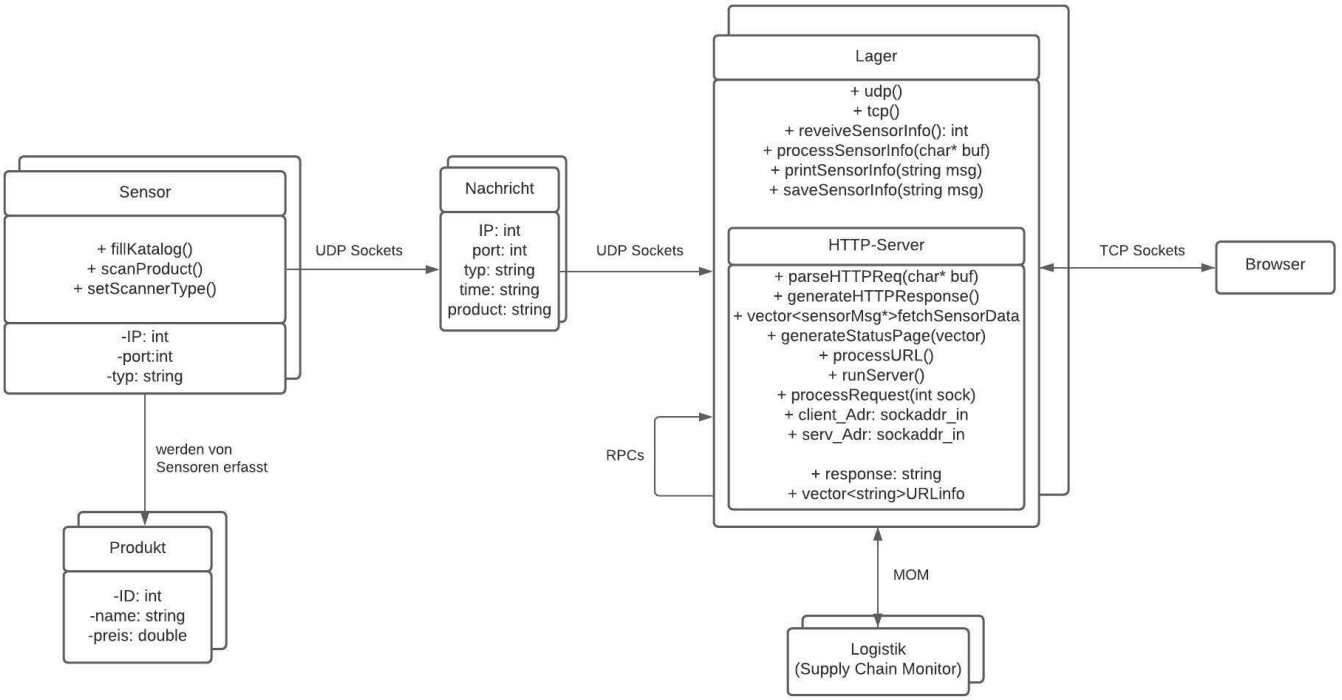


Systemdesign

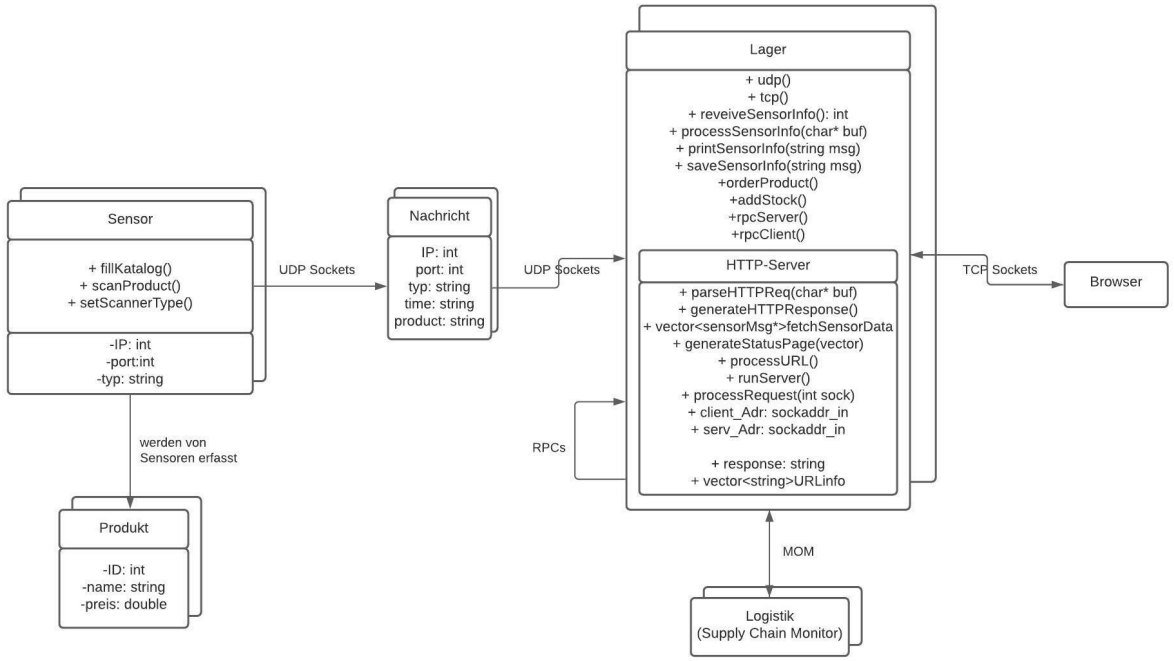
Version 1



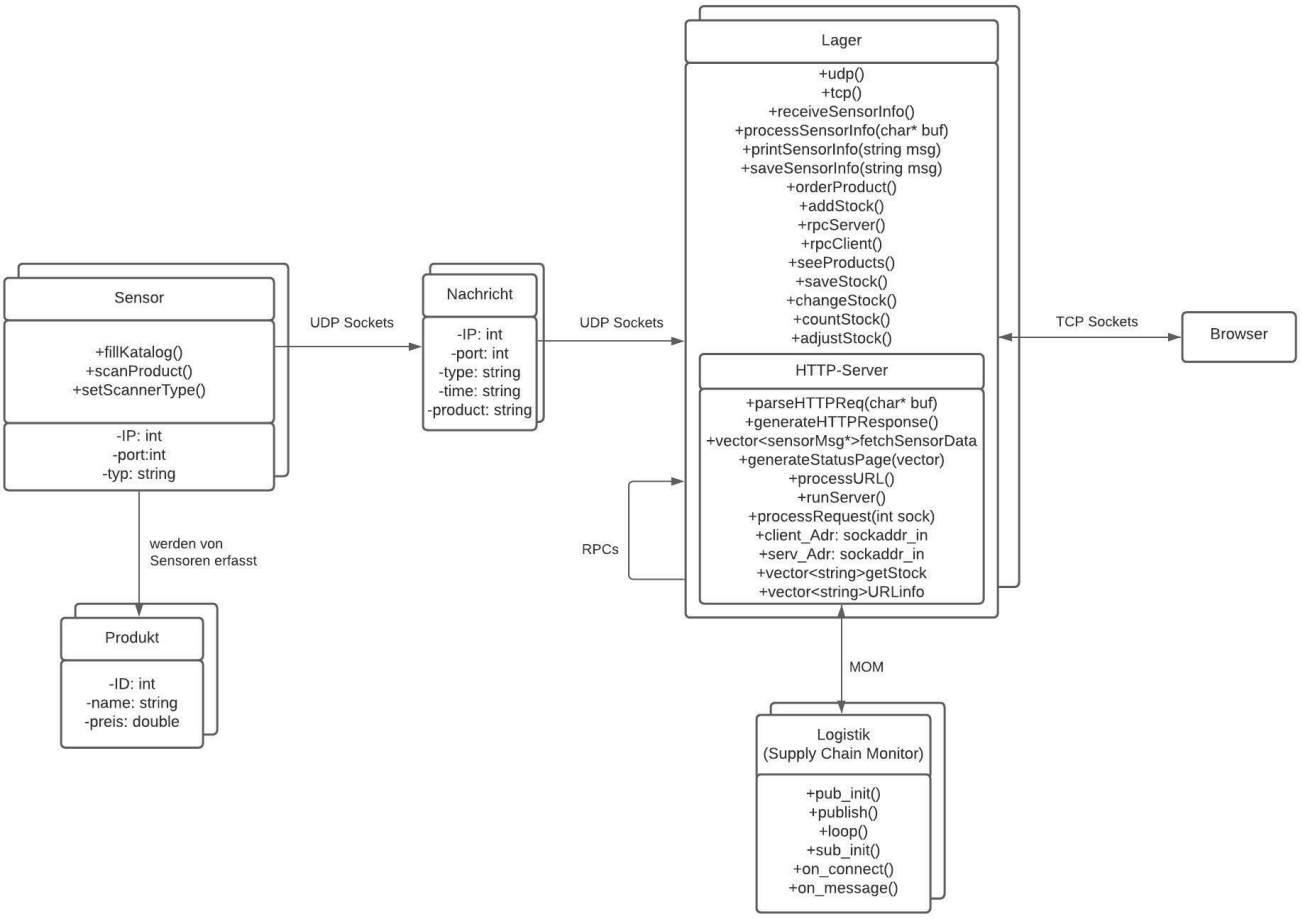
Version 2



Version 3



aktuelle Version (Version 4)



Projektplan: Tasks

Termin 2

- UDP Client (=Sensor) und Server (=Lager) erstellen
- Sensor soll Nachrichten mittels UDP an das Lager übermitteln können
- Lager soll Nachrichten mittels UDP vom Sensor erhalten können
- Nachrichten sollen Informationen über das jeweilige Produkt als auch den IP, Port, Typ des Sensors beinhalten
- Katalog mit Produkten erstellen und befüllen
- Implementierung, dass die es eine simulierte Zeit gibt, die nicht der Echtzeit entspricht

Termin 3

- HTTP-Server soll *HTTP-GET* Anfragen korrekt und vollständig einlesen und verarbeiten können
- HTTP-Server soll auf Sensordaten zugreifen können
- HTTP-Server soll die Auswahl zwischen einzelnen, allen Sensordaten oder der Historie der Sensordaten haben

Termin 4

- Alle Aufgaben für Termin 4 wurden auf Termin 5 verschoben, aufgrund von technischen und zeitlichen Problemen

Termin 5

- Lager soll mittels *thrift* Sensordaten an andere Lager übermitteln können
- Lager soll Produktbestand anderer Lager einsehen können
- Lager soll Lieferungen tätigen können
- Sensor soll bei Bestandsänderung die Produktanzahl anpassen können
- Sensor soll eine Meldung ausgeben können, falls ein Produkt nicht mehr vorhanden ist
- Persistente Speicherung der Daten sollte für jedes Lager möglich sein

Termin 6:

- Lager sollen ihre Daten mit *MQTT* veröffentlichen und übermitteln können
- Verfeinerung und Korrektur

Testbeschreibung

Termin 2

1. Hier wird überprüft, ob die vom Sensor gesendete Nachricht identisch zu der vom Lager erhaltenen Nachricht ist, durch Vergleich der Ausgaben.

Sensor:

```
Hello, I am a Sensor. I will send regular updats to the lager :)
172.20.0.2;8090;Inventarisierung;Schuhe;Thu Jan  1 00:00:00 1970

172.20.0.2;8090;Inventarisierung;Hemd;Thu Jan  1 01:00:00 1970

172.20.0.2;8090;Einlieferung;Jacke;Thu Jan  1 02:00:00 1970

172.20.0.2;8090;Einlieferung;Pullover;Thu Jan  1 03:00:00 1970

172.20.0.2;8090;Inventarisierung;Jacke;Thu Jan  1 04:00:00 1970

172.20.0.2;8090;Inventarisierung;Mantel;Thu Jan  1 05:00:00 1970

172.20.0.2;8090;Auslieferung;Mantel;Thu Jan  1 06:00:00 1970

172.20.0.2;8090;Auslieferung;Kleid;Thu Jan  1 07:00:00 1970
```

Lager:

```
Hello, I am a Lager. I will receive sensor information and relay it to the browser :)
Sensor Listener is online! Waiting for sensor information...
172.20.0.2;8090;Inventarisierung;Schuhe;Thu Jan  1 00:00:00 1970

172.20.0.2;8090;Inventarisierung;Hemd;Thu Jan  1 01:00:00 1970

172.20.0.2;8090;Einlieferung;Jacke;Thu Jan  1 02:00:00 1970

172.20.0.2;8090;Einlieferung;Pullover;Thu Jan  1 03:00:00 1970

172.20.0.2;8090;Inventarisierung;Jacke;Thu Jan  1 04:00:00 1970

172.20.0.2;8090;Inventarisierung;Mantel;Thu Jan  1 05:00:00 1970

172.20.0.2;8090;Auslieferung;Mantel;Thu Jan  1 06:00:00 1970

172.20.0.2;8090;Auslieferung;Kleid;Thu Jan  1 07:00:00 1970
```

⇒ Test ist erfolgreich

2. Hier wird überprüft, ob die Nachricht alle wichtigen Informationen enthält und, ob die Syntax korrekt ist. Im folgenden sieht man die gewünschten Informationen und Syntax.
   Timestamp (Datum & Uhrzeit)
   Sensor-ID
   IP
   Port
   Sensor-Typ
   Produktname

```
lager_test     | Timestamp: Thu Jan  1 05:00:00 1970
lager_test     | Sensor ID: 1
lager_test     | IP: 172.20.0.4
lager_test     | Port: 8090
lager_test     | Sensor Type: Inventarisierung
lager_test     | Product: Mantel
```

   ⇒ **Test ist erfolgreich**

Termin 3

1. Performance-Test: Anzahl von gesendeten vs. erhaltenen Nachrichten
   Hier wird sowohl die Anzahl der erhaltenen Nachrichten, als auch die Anzahl der gesendeten Nachrichten mithilfe von Countern in einer Textdatei namens “overview” festgehalten. Anschließend wird eine Rechnung durchgeführt, sodass man weiß wie viel Prozent der Nachrichten tatsächlich angekommen sind. Dies gibt einen besseren Überblick darüber, wie gut die Performance ist.

```
OVERVIEW
[Sensor Listener] message counter: 354
Sensor said it sent 375 packages
94.4% of sensor packages were received.
```

Termin 4

1. Hier wird überprüft, ob die *orderProduct()* Rechnung korrekt ist
   Dies wird gemacht, indem man die Rechnung selbst als ausgibt, um zu sehen, was tatsächlich in der Funktion passiert, und anschließend wird die Rechnung überprüft. Wenn das Ergebnis richtig ist, gibt es folgende Ausgabe: *“CALC TEST SUCCESSFUL”*, falls nicht wird *“ERROR: CALC TEST UNSUCCESSFUL”* ausgegeben.

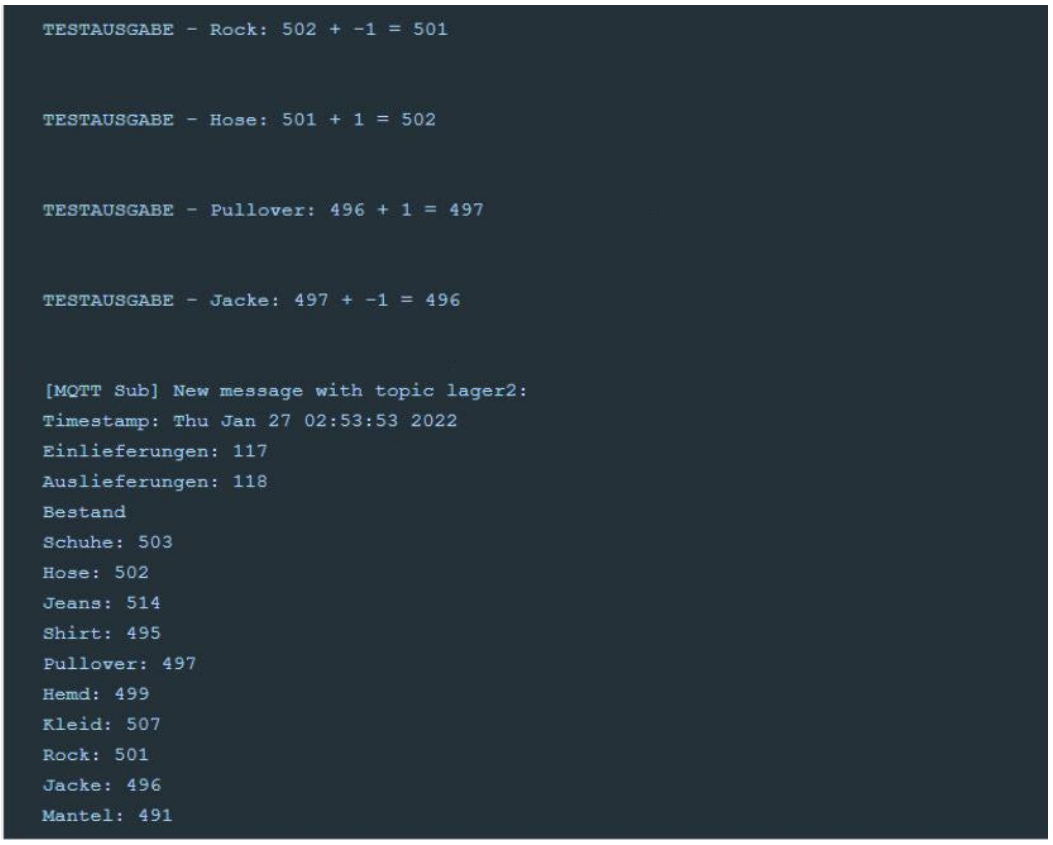
```
Status: Order was placed. 8 pieces of Hemd have been ordered to lager2


TESTAUSGABE - Hemd: 202 + 8 = 210
CALC TEST SUCCESSFUL

8 pieces of Hemd was added from lager2 to lager1. Stock was updated from 202 to 210
Published to topic: lager1
```

   ⇒ **Test ist erfolgreich**

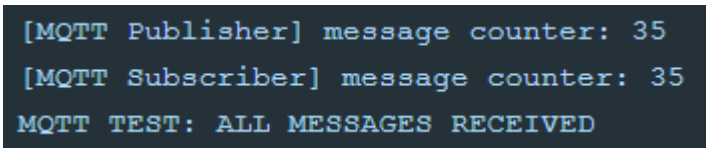
2. Hier wird überprüft, ob der Produktbestand über MQTT richtig ausgegeben wird
   Zum Einen,gibt es eine Ausgabe über die Rechnung und das Ergebnis, anschließend kann man eine Ausgabe, des Lagerbestands sehen. Es wird also verglichen, ob die Werte übereinstimmen.



⇒ **Tests sind erfolgreich**

Termin 5

1. Performance-Test: Anzahl von gesendeten vs. erhaltenen Nachrichten
   Hier wird die Anzahl der vom MQTT Publisher gesendeten Nachrichten und die Anzahl der vom MQTT Subscriber erhaltenen Nachrichten mithilfe eines Counters festgehalten. Jene Zahlen werden verglichen und wenn diese übereinstimmen, gibt es folgende Ausgabe: *“MQTT TEST: ALL MESSAGES RECEIVED”,* falls nicht wird *“MQTT TEST: X MESSAGES WERE NOT RECEIVED* ausgegeben. Wobei X die Anzahl der nicht erhaltenen Nachrichten ist.



⇒ **Test ist erfolgreich**

2. Hier gibt es eine Ausgabe, die alle Topics anzeigt, die abonniert wurden. Hiermit kann man überprüfen, ob diese Topics auch richtig sind, oder ob möglicherweise ein ungewolltes Topic abonniert wurde.

```
Topics subscribed to: lager1 lager3 lager4
```

⇒ **Test ist erfolgreich**

Overview Datei

Hier kann man im Detail sehen, welche Daten in der overview.txt festgehalten wurde. Diese Textdatei wurde für den Zweck gemacht, um einen Überblick über die wichtigen Testdaten zu haben zum Zeitpunkt, wo das Programm beendet wird. Von oben nach unten sind die Zeilen und die Daten, die darin enthalten sind zu sehen:

1. Anzahl der erhaltenen Nachrichten vom Sensor
2. Anzahl der gesendeten Nachrichten vom Sensor
3. Prozentangabe, wie viele Nachrichten erfolgreich angekommen sind (Performance)
4. Anzahl der vom MQTT Publisher gesendeten Nachrichten
5. Anzahl der vom MQTT Subscriber erhaltenen Nachrichten
6. Ausgabe, ob alle Nachrichten erhalten wurden, falls nicht, wie viele nicht erhalten wurden
7. Namen aller Topics, die abonniert wurden
8. Zeitausgabe
9. -10. Finaler Produktbestand des Lager, in welchem diese Ausgabe gemacht wird
11. Zeitangabe, wie lange das Lager, in dem die Ausgabe gemacht wird, online war.
12. Info-Ausgabe, dass diese Daten gespeichert wurden und dass das Lager nun beendet wird

```
OVERVIEW
[Sensor Listener] message counter: 353
Sensor said it sent 371 packages
95.1482% of sensor packages were received.
[MQTT Publisher] message counter: 35
[MQTT Subscriber] message counter: 35
MQTT TEST: ALL MESSAGES RECEIVED
Topics subscribed to: lager1 lager3 lager4
Timestamp: Mon Jan 31 09:21:25 2022
Schuhe: 503 Hose: 501 Jeans: 514 Shirt: 495 Pullover: 496 Hemd: 498
Kleid: 498 Rock: 502 Jacke: 495 Mantel: 491
Lager was online for 76 seconds.
Summary saved. LAGER SHUTDOWN
```