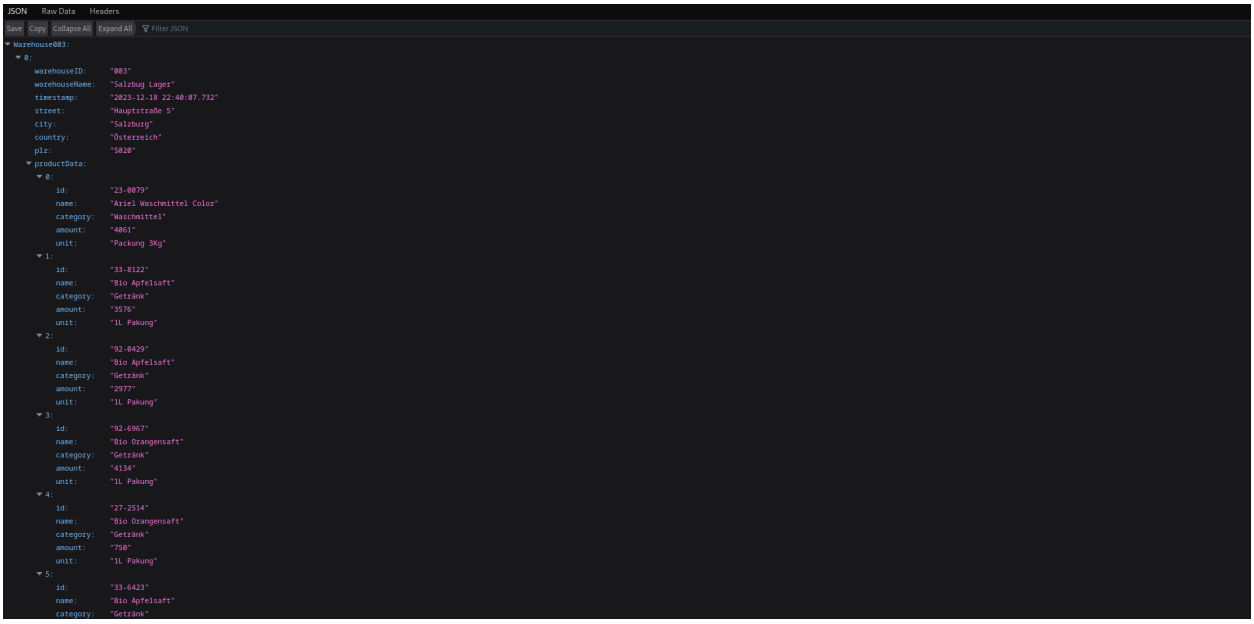
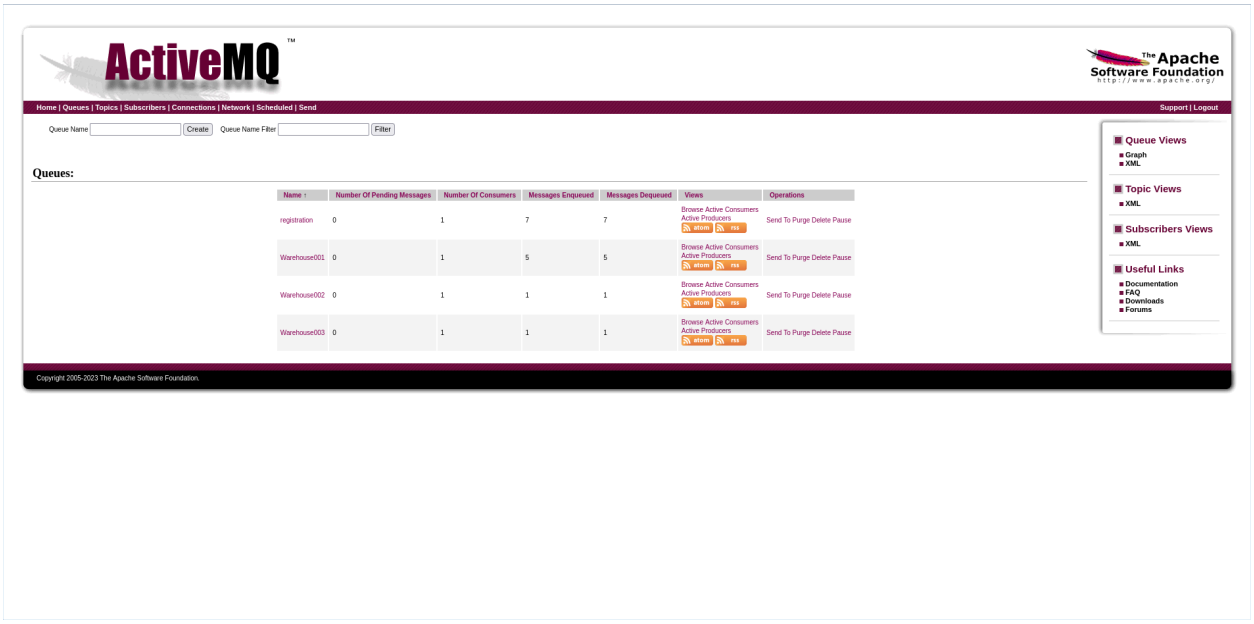
Warehouse 2

Code at GitHub





Setup

Installation vom ActiveMQ

Download von hier.

Extrahieren von ActiveMQ:

```
tar xvzf apache-activemq*.tar.gz -C /tmp/
```

Installation:

```
sudo mv /tmp/apache-activemq* /opt/
sudo chown -R root:root /opt/apache-activemq*
```

Code

Sender

WarehouseController

```
package at.czlabinger.lagerstandort.warehouse;

import at.czlabinger.model.WarehouseData;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;


@RestController
@CrossOrigin
public class WarehouseController {

    private static WarehouseSender registration =
            new WarehouseSender("registration");
    @Autowired private WarehouseService service;
```

```
@RequestMapping("/")
public String warehouseMain() {
    String mainPage =
            "This is the warehouse application! (DEZSYS_WARE
                    +
                    "<a href='http://localhost:8080/warehous
                    +
                    "<a href='http://localhost:8080/warehous
                    +
                    "<a href='http://localhost:8080/warehous
                    +
                    "<a href='http://localhost:8080/warehous
    return mainPage;
}

@RequestMapping(value = "/warehouse/{inID}/data",
        produces = MediaType.APPLICATION_JSON_VALUE)
public WarehouseData
warehouseData(@PathVariable String inID) {
    return service.getWarehouseData(inID);
}

@RequestMapping(value = "/warehouse/{inID}/xml",
        produces = MediaType.APPLICATION_XML_VALUE)
public WarehouseData
warehouseDataXML(@PathVariable String inID) {
    return service.getWarehouseData(inID);
}

@RequestMapping("/warehouse/{inID}/send")
public String sendData(@PathVariable String inID) {
    registration.sendMessage("Warehouse" + inID);
    WarehouseSender sender = new WarehouseSender("Warehouse"
    WarehouseData data = service.getWarehouseData(inID);
    sender.sendMessage(data);
```

```
        sender.stop();
        return "Data send :checkmark:";
    }

    @RequestMapping("/warehouse/{inID}/transfer")
    public String warehouseTransfer(@PathVariable String inID) {
        HttpHeaders responseHeaders = new HttpHeaders();
        return service.getGreetings("Warehouse.Transfer!");
    }
}
```

Diese Klasse gibt Handled die Mappings und fuehrt je nach URL die Methoden aus.

WarehouseSender

```
package at.czlabinger.lagerstandort.warehouse;

import org.apache.activemq.ActiveMQConnection;
import org.apache.activemq.ActiveMQConnectionFactory;

import javax.jms.Session;
import javax.jms.Connection;
import javax.jms.MessageProducer;
import javax.jms.Destination;
import javax.jms.ConnectionFactory;
import javax.jms.ObjectMessage;
import javax.jms.JMSException;
import javax.jms.DeliveryMode;

import java.io.Serializable;


public class WarehouseSender {

    private static final String USER = ActiveMQConnection.DEFAUL
    private static final String PASSWORD = ActiveMQConnection.DE
```

```
    private static final String URL = ActiveMQConnection.DEFAUL

    Session session = null;
    Connection connection = null;
    MessageProducer producer = null;
    Destination destination = null;

    public WarehouseSender(String subject) {

        System.out.println("Sender started.");

                //Erstellen der Connection
        try {

            ConnectionFactory connectionFactory = new ActiveMQC
            connection = connectionFactory.createConnection();
            connection.start();

            session = connection.createSession(false, Session.AU
            destination = session.createQueue(subject);

            producer = session.createProducer(destination);
            producer.setDeliveryMode(DeliveryMode.NON_PERSISTENT

        } catch (Exception e) {
            System.err.println("MessageProducer Caught: " + e);
        }
    }

    public void stop() {
        try {
            producer.close();
            session.close();
            connection.close();
        } catch (JMSException e) {
            System.err.println("Failed while closing " + e);
```

```
        }
    }

    public void sendMessage(Serializable obj) {
        try {

            ObjectMessage message = session.createObjectMessage(
            producer.send(message);
        } catch (JMSException e) {
            System.err.println("Error while sending Message: " +
        }
    }
}
```

Diese Klasse ist fuer das senden der Message zustaendig.

Wenn eine Instanz dieser Klasse erstellt wird, wird eine Connection zu ActiveMQ aufgebaiut und eine Queue erstellt.

Receiver

WarehouseController

```
package at.czlabinger.zentrale;

import at.czlabinger.model.WarehouseData;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;


@RestController
```

```
@CrossOrigin
public class WarehouseController {

    @RequestMapping("/zentrale")
    public String warehouseMain() {

        return "Warehouse Zentrale"
                +
                "<a href='http://localhost:8080/zentrale/data'>D
    }

    @RequestMapping(value = "/zentrale/data", produces = MediaTy
    public HashMap<String, ArrayList<WarehouseData>> getWarehous
        HashMap<String, ArrayList<WarehouseData>> data = new Has

        Registration.updateRegistrations();
        HashSet<String> keys = new HashSet<String>(Registration
        for (String key : keys) {
            data.put(key, Registration.get(key).getMessage());
        }
        return data;
    }
}
```

Diese Klasse ist fuer die Mappings zustaendig.

Wenn auf die `/zentrale/data` zugegriffen wird, werden zuerst die Registraions aktuallisiert damit bekannt ist welche Lagerhallen registriert sind. Dannach wird fuer jede registrierte Lagerhalle die Daten, falls welche vorhanden sind, in einer HashMap gemapped und als json returned.

WarehouseReceiver

```
package at.czlabinger.zentrale;

import java.util.ArrayList;
```

```
import java.util.HashMap;
import java.util.List;
import javax.jms.Connection;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageConsumer;
import javax.jms.ObjectMessage;
import javax.jms.Session;

import at.czlabinger.model.WarehouseData;
import org.apache.activemq.ActiveMQConnection;
import org.apache.activemq.ActiveMQConnectionFactory;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class WarehouseReceiver {

    private static HashMap<String, WarehouseReceiver> clients =
    private static final Logger LOGGER = LoggerFactory.getLogger
    private Session session = null;
    private Connection connection = null;
    private MessageConsumer consumer = null;
    private Destination destination = null;

    public WarehouseReceiver(String subject) {

        try {
            ActiveMQConnectionFactory connectionFactory =
                    new ActiveMQConnectionFactory(ActiveMQConnec
                            ActiveMQConnection.DEFAULT_PASSWORD,
                            ActiveMQConnection.DEFAULT_BROKER_UR
            connectionFactory.setTrustedPackages(
                    List.of("at.czlabinger.model", "java.util"));
            connection = connectionFactory.createConnection();
            connection.start();
            session = connection.createSession(false, Session.AU
```

```
            destination = session.createQueue(subject);
            consumer = session.createConsumer(destination);
        } catch (Exception e) {
            System.out.println("Registration Caught: " + e);
            stop();
        }
    }

    public void stop() {
        try {
            consumer.close();
            session.close();
            connection.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public ArrayList<WarehouseData> getMessage() {
        ArrayList<WarehouseData> msgList = new ArrayList<>();
        try {
            ObjectMessage message = (ObjectMessage)consumer.rece
            while (message != null) {
                WarehouseData value = (WarehouseData)message.get
                LOGGER.info("Received Message: " + value.toStri
                msgList.add(value);
                message.acknowledge();
                message = (ObjectMessage)consumer.receiveNoWait(
            }
        } catch (JMSException e) {
            System.err.println(e);
            return null;
        }

        return msgList;
```

```
    }
}
```

Diese Klasse ist fuer das Reveiven der WarehouseDaten zustaendig. Es wird bei der Instanzinierung dieser Klasse eine connection mit ActiveMQ aufgebaut und wenn die `getMessage()` Methode aufgerufen wird, wird durch alle Messages die in der Queue sind durch itteriert und in einer ArrayList gespeichert. Da die Messages acknowledged werden verschwienden sie nach einem refresh.

Fragestellung

- Nennen Sie mindestens 4 Eigenschaften der Message Oriented Middleware?
  - Ermoeglicht die Kommunikation zwischen verteilten Systemen durch den Austausch von Nachrichten. Diese Nachrichten koennen Daten, Anfragen oder Statusinformationen sein.
  - Bietet Mechanismen zur sicheren Übertragung von Nachrichten, einschliesslich Fehlerbehandlung, Wiederholung und Bestaetigungen, um sicherzustellen, dass Nachrichten korrekt zugestellt werden.
  - Ermoeglicht die Interaktion zwischen verschiedenen Plattformen und Anwendungen, unabhaengig von der Programmiersprache oder dem Betriebssystem, auf dem sie ausgeführt werden.
  - Leichter Ausbau, um mit steigender Last und Anzahl von Teilnehmern umgehen zu koennen.
- Was versteht man unter einer transienten und synchronen Kommunikation?
  - **Transiente Kommunikation:**
    - Hierbei werden Nachrichten ohne dauerhafte Speicherung uebermittelt. Das bedeutet, dass die Nachricht nach dem Senden oder Empfangen nicht auf Dauer gespeichert wird.
  - **Synchrone Kommunikation:**
    - Bei synchroner Kommunikation wartet der Absender auf eine unmittelbare Antwort vom Empfaenger, bevor er mit anderen Aufgaben fortfährt.

- Beschreiben Sie die Funktionsweise einer JMS Queue?
  - Eine JMS Queue ist eine Warteschlange, die nach dem Prinzip "First-In-First-Out" (FIFO) arbeitet.
  - Sender senden Nachrichten an die Queue, und Empfänger (Consumer) lesen Nachrichten in der Reihenfolge, in der sie eingetroffen sind.
  - Jede Nachricht wird nur von einem Consumer verarbeitet, um eine exklusive Verarbeitung sicherzustellen.
  - Die Nachricht bleibt in der Queue, bis sie erfolgreich von einem Consumer verarbeitet wurde.
- JMS Overview - Beschreiben Sie die wichtigsten JMS Klassen und deren Zusammenhang?
  - **ConnectionFactory:**
    - Erzeugt Verbindungen zur JMS-Broker.
  - **Connection:**
    - Repraesentiert die Verbindung zum JMS-Broker.
  - **Session:**
    - Kontext für Produktion und Konsum von Nachrichten.
  - **MessageProducer und MessageConsumer:**
    - Verantwortlich für die Erzeugung bzw. den Empfang von Nachrichten.
  - **Destination:**
    - Repräsentiert das Ziel (Queue oder Topic), auf das Nachrichten gesendet oder von dem sie empfangen werden.
- Beschreiben Sie die Funktionsweise eines JMS Topic?
  - Ein JMS Topic ermoeglicht die Veroeffentlichung und den Empfang von Nachrichten im Rahmen des Publish-Subscribe-Musters.
  - Sender senden Nachrichten an ein Topic, und alle Subscriber, die sich für dieses Thema interessieren, erhalten eine Kopie der Nachricht.

- Was versteht man unter einem lose gekoppelten verteilten System? Nennen Sie ein Beispiel dazu. Warum spricht man hier von lose?
  - **Lose gekoppeltes verteiltes System:**
    - Unabhängige Komponenten oder Subsysteme.
    - Geringe oder keine direkte Kenntnis voneinander.
    - Änderungen an einem Teil beeinflussen nicht zwangsläufig andere Teile.
  - **Beispiel: World Wide Web (WWW)**
    - Verschiedene Dienste und Ressourcen.
    - Webseiten können unabhängig voneinander entwickelt werden.
    - Benutzer können auf verschiedene Ressourcen zugreifen, ohne direkte Verbindung zwischen den Webseiten.
  - **Warum "lose gekoppelt"?**
    - Minimale Abhängigkeit zwischen den Komponenten.
    - Erleichtert Wartung, Erweiterung und Skalierung.
    - Gut definierte Schnittstellen zwischen den Komponenten.
    - Jede Komponente ist für ihre eigenen Aufgaben verantwortlich.