ÖZYEĞİN UNIVERSITY
FACULTY OF ENGINEERING

CS 300

SUMMER PRACTICE REPORT

HÜMEYRA ECEM ÖZ
S012019

INTERNSHIP COMPANY & DEPARTMENT:

ICTerra Information & Communication Technologies / Energy Software Group

19.08.19-16.09.19

SUMMER PRACTICE REPORT

STUDENT	
Name	**Hümeyra Ecem Öz**
Internship Start Date	**19.08.19**
Internship Completion Date	**16.09.19**
Total Working Days	**20**
COMPANY	
Name	**ICTerra Information & Communication Technologies**
Department	**Energy Software Group**
Address	**Galyum Blok Kat:2 No:3, ODTÜ Teknokent, 06800 Çankaya**
SUPERVISOR	
Name	**Nurettin Mert Aydın**
Title	**Group Manager**
Department	**Energy Software Group**
Phone	**+90 312 292 50 82**
E-Mail	**mert.aydin@icterra.com**

Abstract

The internship I conducted was in the firm named ICTerra Information & Communication Technologies. I worked in the Energy Software group. I focused on data transferring and message continuity for IoT and remote telemetry units. The issues about these was solved by invention of MQTT protocol. Indeed, the project I have prepared is about implementing a MQTT client – server Java application by taking advantages of Eclipse Paho Library and Spring Boot Framework. Also, it can be handled by implementing an application using DDS protocol, however, the way of solution should be chosen wisely according to project's expectations and priorities. In this report, the differences and common features of MQTT & DDS protocols, under what conditions one of them should be used, MQTT client-server application implementation steps, application architecture and flow are stated. Thanks to this project, I have learned what is Maven and libraries it ensures, Spring Boot framework in order to write the application. To test it, I used MQTTLens as a client, and Postman as REST API. The skills I acquired during the internship are software debugging, testing and documentation, problem solving, strategic planning, and so on in this internship.

Introduction

Intern project's subject which is implementing a MQTT client – server application by using Spring Boot was given in the very first day. After taking the assignment, research phase has officially started. According to the research I have done, before 1999, transferring data was not easy to implement. Additionally, ways to transfer data are not efficient enough. In order to provide message continuity with simple implementation, MQTT protocol was invented.

MQTT (Message Queue Telemetry Transport) is a messaging protocol that was created to address the need for a simple and lightweight method to transfer data to/from low powered devices, such as those used in industrial applications. With the increased popularity of IoT (Internet of Things) devices, MQTT has seen an increased use. The protocol supports a single messaging pattern, namely publish-subscribe pattern. Each message sent by a client contains an associated "topic" which is used by the broker to route it to subscribed clients. Topics' names can be simple strings. In order to receive messages, a client subscribes to one or more topics using its exact name.

The tools that are necessary to implement demanded program was Eclipse Paho Library, which includes all essential classes in order to provide various MQTT classes, Maven, which is also a necessity to specify the dependencies of the project and an option to import needed libraries. Additionally, I used Mosquitto in order to obtain an efficient server. In addition to that, MQTTLens can be used as a client tool for MQTT, and Postman is used to send data. Also,

I was required to make the project by using Spring Boot framework. Hence, I have learned how to use Spring Boot with a Java project and its useful annotations.

Company Description

In terms of the sector the firm is in, ICTerra is a software company that is expert not only in the defense and aerospace industry but also civilian and public sector. There are numerous groups in the firm such as Energy, Telecommunication, Avionics, Medical, Cyber Security, so on which ensures national or international clients such as Aselsan, Ayesaş, Atos, SDT, TRT World, Halkbank, Wacom, and Unify demanded or required software applications. That is to say, under each group, there are various projects going on.

The company builds strategic partnership partnerships with world's top IT companies such as BMC, Unify, Microsoft, IBM, and Oracle to provide value-added IT solutions and deliver high quality professional services based on unique business needs. According to the Top 500 IT Companies Research in Turkey, ICTerra ranked 6th in "software export", and 7th in "industry-specific software development" categories.

It has 4 headquarters which are in London, Munich, İstanbul, and the main office is in Ankara. Company has approximately 200 employees who are currently working.

Originally, ICTerra comes from Siemens. When the R&D team decided to work separately and Vedat Uslu took the company's management in 2013, ICTerra was established.

Workit, CEP, MyIT can be given as examples of company's outstanding products. Workit is a digital working platform like LinkedIn, which is developed within ICTerra. It is used for communicating and socializing throughout the company and employees. Moreover, in the past months, ICTerra have received a "Domestic Goods Certificate" for this product. Likewise, CEP is another domestic product example of ICTerra. The employees' working hours are controlled and counted in the system. Also, getting permission from work, and other administrative affairs could be done via CEP. Lastly, MyIT is another example of company's products. It is a digital workplace that can be used for again some administrative affairs, reporting technical problems, requests about purchasing software, so on.

As a research example, the company's lately performed test in MMS extent can be given. A target signal's location was determined approximately utilizing hybrid technology by using benefits of national opportunities only.

Energy group, which is the group that I was working with, was engaged in a project which will be delivered to a leading company in defense sector in Turkey. Simulators & Advanced Remote Terminal Unit (RTU) of the project is handled by the Energy Group. It is about software devices for the embedded devices that are designed for measuring and analyzing electrical parameters of low voltage distribution network with high integration capabilities for real-time

visibility. Technologies that are used are MISRA C, IBM Rational Rhapsody, FCGI, Lighttpd, Qt, and MQTT. Some parts of the project is written with Java, but they also use C languages while working with embedded systems.

Generally, I worked individually because of the limited internship time. However, our group manager Mert Aydın gave a project that is relevant with Energy group's project. I used MQTT as a messaging protocol as they do, I also reported a message containing a temperature, but the difference is I created the temperature randomly. Hence, my individual project has common features with the real/on-going project, actually, it is just a small part of the whole project. In other words, it can be said that the project that I have done is a software prototype of their project.

Energy group has been working in groups, and they use project management software methodology named Agile. Every day, they do Scrum, which is the most popular Agile development framework, for 10-15 minutes. They discuss the previous day's achievements as well as the expectations for the following one. It can be accepted that Agile's most important and core values are expressed as individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan. This method is so beneficial that it increases the quality of product, the revenue gained, and also it ensures higher customer satisfaction.

MQTT Client – Server Application Using Spring Boot & MQTT – DDS Comparison

i. Problem Statement

Approximately two decades ago, lack of sources to transfer data efficiently was a serious issue in IoT. Also, solutions were not easy enough. In order to provide these main features, a protocol should have been created. Expectations for the future protocol specified by engineers has several requirements.

Constraints in find a solution:

- Simple implementation
- Lightweight and bandwidth efficiency
- Quality of Service data delivery
- Message continuity

MQTT

The first solution is MQTT protocol. The MQTT protocol was invented in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link). It is a very light weight and binary protocol, and due to its minimal packet overhead, MQTT excels when transferring data

over the wire in comparison to protocols like HTTP. Another important aspect of the protocol is that MQTT is extremely easy to implement on the client side. Ease of use was a key concern in the development of MQTT and makes it a perfect fit for constrained devices with limited resources today.

It consists of three components which are publishers, broker and subscribers which are seen in Figure 1. It is based on publish/subscribe architecture. Publishers are like sensors or IoT devices which send their data or change in some information to the broker as per topic. Subscribers are like applications which subscribed with broker to receive change in parameters of a certain topic or sensory data.

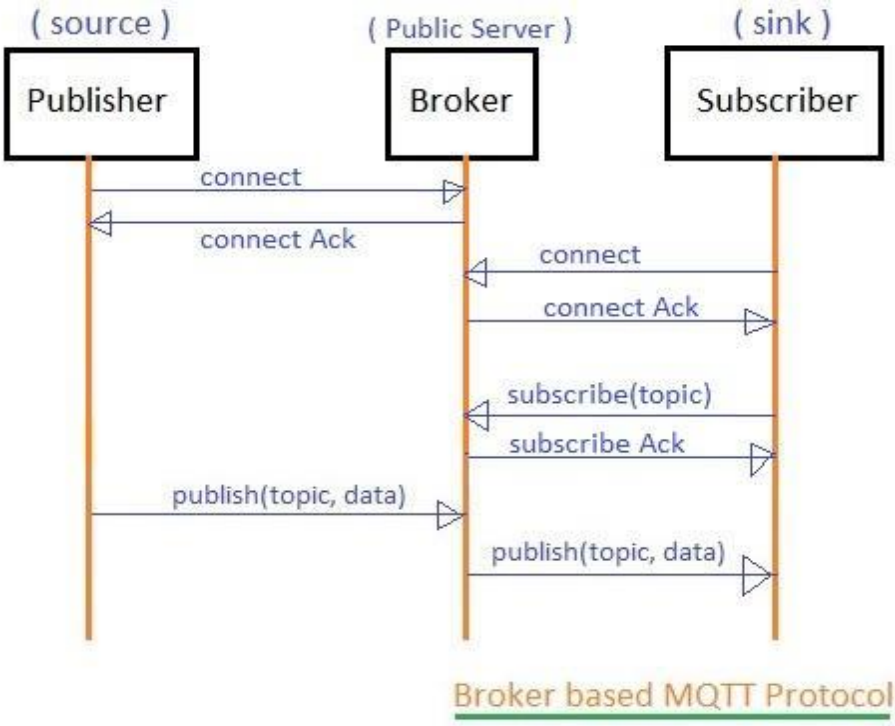


Figure 1

MQTT is designed to provide very efficient communication across the wide area network portion of the network. HTTP can be used for this purpose but it has some limitations.

However, MQTT also has several disadvantages. Devices on today's Internet of Things communicate primarily with centralized servers like MQTT does. The lack of protocol is a direct obstacle to the IoT. If data will be trapped within centralized silos, it would remain more difficult to share; and more security and privacy concerns would be raised. It would have to travel farther and might be subject to congestion at hubs, slowing down services.

DDS

The second solution for these problems is the Data Distribution Service for Real-Time Systems (DDS). DDS is an Object Management Group (OMG) machine-to-machine middleware "m2m"

standard that aims to enable scalable, real-time, dependable, high-performance and interoperable data exchanges between publishers and subscribers. The Data Distribution Service (DDS) most directly addresses the development of intelligent distributed machines.
It is broker less protocol used mainly for M2M and IoT applications. It is session layer protocol. Instead of broker, it uses data writers and data readers. Like MQTT, it is also publish / subscribe protocol without broker. It has 23 QoS levels which include security, priority, reliability, urgency, durability, and so on.
Comparison between MQTT and DDS protocols are stated in Table 1.

Features	**MQTT**	**DDS**
Full Form	Message Queue Telemetry Transport	Data Distribution Service
Architecture	Centralized, all communications route through the broker	Decentralized, peer to peer communication
Transport layer protocol	TCP	UDP
Admin needed	Yes	No
Message rate	Few messages/second per device	Up to 10000s of messages/second per device
Service interruptions	They are tolerable in MQTT network	They are potentially calamitous in DDS architecture

Table 1

There are numerous protocols for message queuing like AMQP, MQTT, STOMP and among these, big players seem to have chosen a winner: MQTT due to its network bandwidth friendly nature and low power consumption. Although DDS is an advantageous protocol in terms of architecture, transport layer protocol, and so on, MQTT is definitely a more commonly used protocol than DDS. It is because DDS has several cons that, unlike MQTT, it has high costs, and it is not as practical as MQTT. MQTT has ease of use in every area.

ii. Tools and Techniques Used

MQTT client libraries are available for a huge variety of programming languages. For example, Android, Arduino, C, C++, C#, Java, JavaScript, so on. However, I chose to write the project with Java, because Java is the most comfortable language for me to implement applications. It is an advantage that I am used to write programs with Java. Additionally, according to what I have heard, C languages mostly used when memory, and other efficiency features taken into consideration as a priority. For instance, it is used for projects that are integrated with embedded systems. However, in this project, I gave the priority to research, then a successful

implementation and achievements that I gained. That is why I used Java instead of other languages.

IntelliJ was used as an IDE. I used Spring Boot to implement the project as it should be. It is nothing but existing Spring Framework + some embedded HTTP servers (Tomcat, Jetty, so on) – XML or annotations configurations. That means, it is not needed to write any XML configuration and few annotations only. Thanks to Spring Boot, it became an easier Java application to develop. Also, it provides embedded HTTP servers like Tomcat to develop and test the web applications easily. In addition to that, it ensures lots of plugins to develop and test Spring Boot Applications very easily using build tools like Maven. I also used Maven in order to specify the necessary dependencies in the project. In addition to that, I used Mosquitto as a MQTT broker. Also, Eclipse Paho Library is used to obtain essential MQTT classes, MQTTLens as a client, and Postman as a REST API.

This was not a group project, so the whole project was under my responsibility. I prepared a plan first, started to do research, tried to write the codes from easier to harder and complex, estimated tasks to perform, if it ended up with an error, debugged the program, eventually implemented it, and got feedback from the group manager who gave me the project.

iii. Detailed Explanation

At first, I tried to come up with a project which does not include Spring Boot, then I adapted the project to Spring Boot.

The first thing should be done is downloading Apache Maven and opening a Maven project in IntelliJ. After downloading Apache Maven, \bin directory must be added to the Path from Environment Variables in order to make project run. By opening a Maven project, a file which is called pom.xml is going to be created. This file contains project properties, dependencies, so on.

Next step is to download Eclipse Paho Library from Maven Central. After that, it should be imported to the project from Project Structure, also the following dependency have to be added to pom.xml.

```
<dependency>
  <groupId>org.eclipse.paho</groupId>
  <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
  <version>1.2.0</version>
</dependency>
```

Next thing that is going to be done is creating MQTTClientTest class. In that class, an instance of the MqttClient need to be created, and then it will be connected to the server via its serverURI parameter. Another parameter MqttClient object has is publisherID or clientID, which can be created randomly, because it is not used in this context. Moreover, there is a

class called MqttConnectOptions, which has all significant options that MqttClient should have. For example, setAutomaticReconnect(boolean) ensures MqttClient to connect automatically, setCleanSession(boolean) is about connection persistency. If setCleanSession(true) is called, that means it is a non-persistent connection. With a non-persistent connection, the broker doesn't store any subscription information or undelivered messages for the client. This mode is ideal when the client only publishes messages.

Now, MqttClient is ready to connect to the server. It is done by its connect() method. If the connection is demanded with particular options, connection should be established with parameter as connect(MqttConnectOptions).

After connecting to server, messages need a specific topic to be sent. This can be initiated basically as a String.

Next thing to do is to send messages via MqttClient. In order to do that, Eclipse Paho Library has another class called MqttMessage. I arranged this with another class named EngineTemperatureSensor. It is shown in the following code.

```
package mqttwspring;

import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttMessage;

import java.util.Random;
import java.util.concurrent.Callable;

public class EngineTemperatureSensor implements Callable<Void> {
    private MqttClient client;
    public String TOPIC;
    private Random rnd = new Random();

    public EngineTemperatureSensor(MqttClient client) {
        this.client = client;
    }

    @Override
    public Void call() throws Exception {
        if (!client.isConnected())
            return null;
        MqttMessage msg = readEngineTemp();
        msg.setQos(0);
        msg.setRetained(true);
        client.publish(TOPIC, msg);
        return null;
    }

    public MqttMessage readEngineTemp() {
        double temp = 80 + rnd.nextDouble() * 20.0;
        byte[] payload = String.format("T:%04.2f", temp).getBytes();
        return new MqttMessage(payload);
    }
}
```

As can be seen, this class includes readEngineTemp() method which generates a random temperature and returns it as a MqttMessage.

After implementing this class, it should be used as new MqttMessage instance by calling the readEngineTemp() method. Then, this message can be published with publish() method to the specific topic. Also there are multiple setter & getter methods of MqttMessage that can be used such as setQos(int), which can take 3 values in MQTT implementation, setPayload(byte[]) which takes a byte[] –or a string turned into byte array with getBytes() method- and sets the MqttMessage as it. QoS is a key feature of the MQTT protocol. It gives the client the power to choose a level of service that matches its network reliability and application logic. 3 integer values that QoS can have are 0, 1, and 2. 0 means at most once and guarantees a best-effort delivery. It is often called “fire and forget”. 1 means at least once meaning that the message is delivered at least one time to the receiver. 2 means exactly once, and it is the highest level of service. This level guarantees that each message is received only once by intended recipients. In addition to that, it is the safest and slowest quality of service level. With optional and configurable QoS approach, MQTT provides ensured message delivery which is most probably the most important aspect since people are looking for real-time, complete, time-tagged data.

In order to receive messages from the MQTT broker, a callback handler to process received messages is required. In this class, CountDownLatch is used as a synchronization mechanism between our callback and the main execution thread, decrementing it every time a new message arrives. Essentially, by using a CountDownLatch, it can be caused a thread to block until other threads have completed a given task. In other words, CountDownLatch is used to make sure that a task waits for other threads before it starts.

The subscribe() method used below takes an MqttMessageListener instance as an argument. In this case, a simple lambda function that processes the payload and decrements a counter. If not enough messages arrive in the specified time -here, it is 1 minute-, the await() method will throw an exception.

```
CountDownLatch receivedSignal = new CountDownLatch(10);
mqttClient.subscribe(topicName, (topic, msg) -> {
    byte[] payload = msg.getPayload();
    String m = new String(payload);
    System.out.println(m);
    receivedSignal.countDown();

});
try {
    receivedSignal.await(1, TimeUnit.MINUTES);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

Finally, the MQTT client – server application can be tested. In order to subscribe topics or publish messages, MQTTLens should be downloaded. It is available on the Google Chrome as an extension. First step after downloading it is connecting to the server. Then, entering the particular topic is enough to reach the published messages from the server. Output in the MQTTLens and IntelliJ should be like the Figure 2.

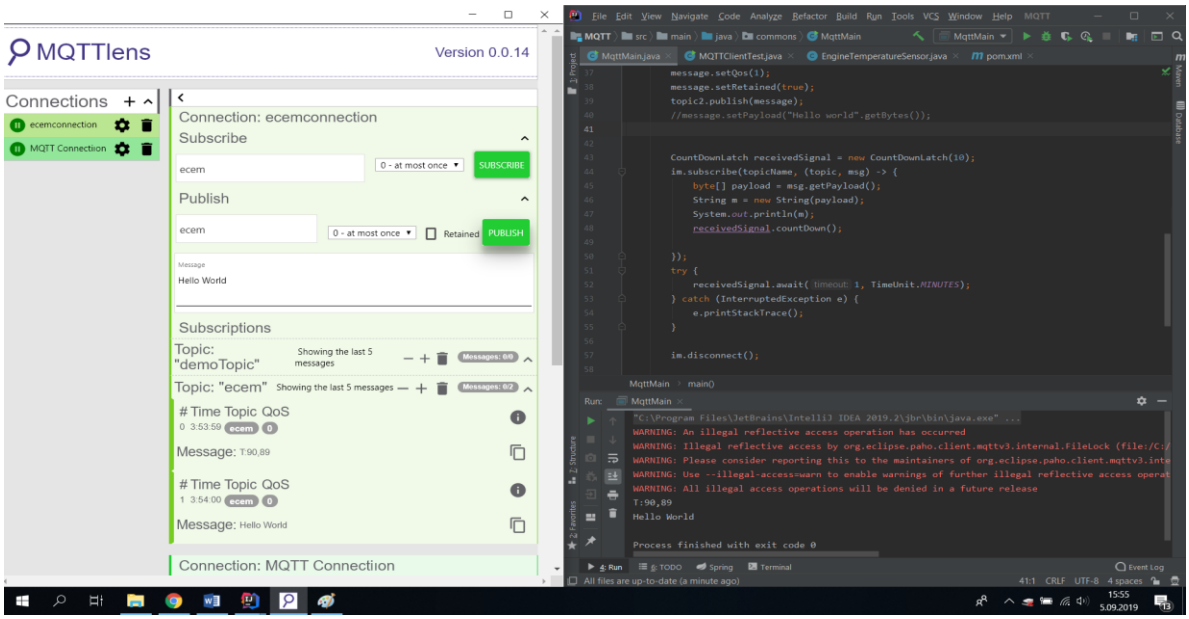


Figure 2

Now, it is time to turn the project into a Spring Boot application. In order to do that, first thing should be done is opening again a Maven project and adding Spring Boot framework to this project. By doing that, all dependencies that Spring Boot requires is going to be imported to the pom.xml file automatically. Other than that, Spring Boot CLI should be downloaded and \bin directory should be added to the Path.

After that, first class of the new project which is MqttConfig can be created. In this class, necessary variables such as broker address, port number, QoS, TCP should be defined. Also, this class includes an abstract method config(), which makes the class abstract as well. Second thing should be done is creating the MQTTPublisherBase interface. This interface includes publishMessage(String, String) and disconnect() methods. Next, MQTTPublisher class, which extends MQTTConfig and implements MqttCallback, MQTTPublisherBase should be initiated. In this point, Spring Boot became involved. Because the class includes defined objects, @Component annotation should be written before the class definition. In this class, some constants such as colon and clientID exists. Other than that, brokerURL, MqttClient, MqttConnectionOptions, EngineTemperatureSensor and MemoryPersistence is defined as null. Additionally, class has a default constructor. Inside the constructor, config() is called. In config() method, brokerURL, persistence, options, and mqttClient is initialized. Same

connection options are valid for this MqttClient as well. mqttClient should be connected with these options. Also, the EngineTemperatureSensor is initiated by sending it mqttClient as a parameter. The difference from the implementation without Spring Boot is that this includes setCallback() method.

Then, in order to see the random temperature as a message, MqttMessage should be created by calling the readEngineTemp() method again. Other steps are simple with the setter methods of MqttMessage. Since the class implements MqttCallback interface, it should override connectionLost(Throwable), messageArrived(String,MqttMessage), and deliveryComplete(IMqttDeliveryToken) methods. Also, because the class implements MQTTPublisherBase, publishMessage(String, String) must be overridden. This method takes two strings which are the topic and the message as a parameter. Inside the method, new MqttMessage should be created by turning the String message into MqttMessage. After that, the message can be published to the topic specified with the publish(String, MqttMessage) method. Lastly, disconnect() method should be included.

Next, MQTTSubscriberBase interface should be created with methods subscribeMessage(String) and disconnect() methods. Then, MQTTSubscriber class which extends MQTTConfig and implements MqttCallback, MQTTSubscriberBase must be initiated. Also, this class need to be started with the annotation @Component too. In this class, some constants such as colon and clientID exists. Other than that, brokerURL, MqttClient, MqttConnectionOptions and MemoryPersistence is defined as null. Additionally, class has a default constructor. Inside the constructor, config() is called. In config() method, brokerURL, persistence, options, and mqttClient is initiated. Same connection options are valid for this MqttClient as well. mqttClient should be connected with these options. This config() method includes setCallback() method, too. Since this class implements MqttCallback too, connectionLost(Throwable), messageArrived(String,MqttMessage), deliveryComplete(IMqttDeliveryToken) must be overridden, again. Inside messageArrived(String, MqttMessage) method, there are certain Strings that are reporting the message, when the message came, by whom the message was sent in the console. Also, subscribeMessage(String) and disconnect() methods are overriden. In subscribeMessage(String) method, subscribe(String, int) is called with specific topic and a particular QoS.

The subscriber needs to subscribe messages continuously. It can be achieved by implementing Spring Boot's CommandLineRunner. A bean of TaskExecutor must be created, which is in this case a SimpleAsyncTaskExecutor().

```
package mqttwspring;

import org.springframework.context.annotation.Bean;
import org.springframework.core.task.SimpleAsyncTaskExecutor;
import org.springframework.core.task.TaskExecutor;
import org.springframework.stereotype.Component;

@Component
public class AppConfig {
    @Bean
    public TaskExecutor taskExecutor(){
        return new SimpleAsyncTaskExecutor();
    }
}
```

In MessageListener class, subscriber is created. Since this class implements Runnable interface, run() method need to be overriden, and subscribeMessage(String) method is called with that subscriber inside this method.

```
package mqttwspring;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MessageListener implements Runnable{

    @Autowired
    MQTTSubscriberBase subscriber;

    @Override
    public void run() {
            subscriber.subscribeMessage("demoTopic");

    }
}
```

Next step is to create the Application class which has the main method. Before initializing the class, @SpringBootApplication annotation is used in order to specify that this application will be run as a Spring Boot application. In the class, a MessageListener object is created. Finally, CommandLineRunner can be created to schedule the task. It will be executed after all beans have been initialized. Process will be executed by using MessageListener.

```
package mqttwspring;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import
org.springframework.boot.web.servlet.support.SpringBootServletInitializer;
import org.springframework.context.annotation.Bean;
import org.springframework.core.task.TaskExecutor;
```

```java
@SpringBootApplication
public class Application extends SpringBootServletInitializer {

    @Autowired
    Runnable MessageListener;

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder
application){
        return application.sources(Application.class);
    }
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
    @Bean
    public CommandLineRunner schedulingRunner(@Qualifier("taskExecutor")
TaskExecutor ex){
        return new CommandLineRunner() {
            @Override
            public void run(String... args) throws Exception {
                ex.execute(MessageListener);
            }
        };
    }
}
```

In addition to all these classes, another class called DemoRestController is needed to receive data from REST client –Postman- and publish to the broker at the specific topic. Finally REST client will get a response from server as “Message sent to Broker”. To do that, firstly the class should be started with @RestController annotation. Secondly, @RequestMapping should be done outside the index(String) method. By doing that, the address value in which the messages are sent, also the request method is specified.

```java
package mqttwspring;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class DemoRestController {

    @Autowired
    MQTTPublisherBase publisher;

    @RequestMapping(value="/mqtt/send", method = RequestMethod.POST)
    public String index(@RequestBody String data){
        publisher.publishMessage("demoTopic", data);
        return "Message sent to Broker";
    }
}
```

Now, the application is ready to be tested with MQTTLens, and also using the REST client Postman.

Although DDS implementation is not in the project content, besides implementing MQTT, I tried to implement DDS, but it didn't finish. Two classes named HelloPublisher and HelloSubscriber are explained below in detail.

First of all, RTI Connext should be downloaded and installed in order to define objects and entities such as Domain, DomainParticipant, Publisher and DataWriter, Subscriber and DataReader, Topic. Next, the environment variables must be set. Then, essential libraries should be added to the project externally. Now, the project is ready to be created.
First, Publisher class named HelloPublisher was initiated. Then, a DomainParticipant has to be created. It defines in which Domain an application belongs, and will be used to create all other needed entities like the Publisher, DataWriter, and Topic. The create_participant() method takes four arguments named domainID, QoS, listener, and mask. Definition is below.

```
DomainParticipant participant =
DomainParticipantFactory.get_instance().create_participant(
        0,
        DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
        null,
        StatusKind.STATUS_MASK_NONE);
```

Now, the DomainParticipant can be used to create a Topic with name "Hello, World", and the built-in String data type. The create_topic() method takes five parameters called topic_name, type_name, QoS, listener, mask.

```
Topic helloWorldTopic = participant.create_topic(
        "Hello, World",
        StringTypeSupport.get_type_name(),
        DomainParticipant.TOPIC_QOS_DEFAULT,
        null,
        StatusKind.STATUS_MASK_NONE);
```

The DomainParticipant can now instatiate a DataWriter. This object will be used to write the messages to be sent to subscribers. The create_datawriter() method takes four arguments which are topic, QoS, listener, and, mask.

```
StringDataWriter dataWriter = (StringDataWriter)
participant.create_datawriter(
        helloWorldTopic,
        Publisher.DATAWRITER_QOS_DEFAULT,
         null,
        StatusKind.STATUS_MASK_NONE);
```

Finally, DataWriter can be used to write messages, and automatically publish through the default Publisher. The write() method takes two arguments.

```
BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
try
{
    while (true) {
        System.out.print("Please type a message> ");
        String toWrite = reader.readLine();
        dataWriter.write(toWrite, InstanceHandle_t.HANDLE_NIL);

        if (toWrite.equals("")) break;
    }
}
catch (IOException e)
{
    e.printStackTrace();
} catch (RETCODE_ERROR e) {
    // This exception can be thrown from DDS write operation
    e.printStackTrace();
}
```

Before terminating the program, all entities of the DomainParticipant and the DomainParticipant itself should be deleted.

```
System.out.println("Exiting...");

participant.delete_contained_entities();

DomainParticipantFactory.get_instance().delete_participant(participant);
```

Next step is to create the Subscriber class. It is very similar to the Publisher. The difference is that instead of DataWriter, there will be a DataReader. A listener will be implemented to automatically run when new messages are available. For the Subscriber to be able to receive messages sent by the HelloPublisher, the DomainParticipant must be in the same Domain and the Topic must have the same name and data type.

After the DomainParticipant and Topic, the DataReader will be created. The listener is, in this case, an instance of HelloSubscriber. That is also the reason why the HelloSubscriber class extends from the DataReaderAdapter class and overrides the on_data_available(DataReader reader) method which is the method that will be triggered once data becomes available. The mask indicates that this listener should be trigger as soon as new messages/data is available.

```
StringDataReader dataReader = (StringDataReader)
participant.create_datareader(
        topic,
```

```
        Subscriber.DATAREADER_QOS_DEFAULT,
        new HelloSubscriber(),
        StatusKind.DATA_AVAILABLE_STATUS);
```

Finally in the on_data_available() method, the logic to read data is implemented. Every received message is actually composed of two parts. First the message itself, and second a metafile of type SampleInfo which contains additional information about that message.
The created stringReader then tries to read the next sample with the take_next_sample() method into which it passes the info file. If the sample contains actual data, the valid_data property of the info object will equal true and the sample will be printed onto the console.

```
public void on_data_available(DataReader reader) {

    StringDataReader stringReader = (StringDataReader) reader;
    SampleInfo info = new SampleInfo();

    for (; ; ) {
        try {
            String sample = stringReader.take_next_sample(info);
            if (info.valid_data) {
                System.out.println(sample);

                if (sample.equals("")) {
                    shutdown_flag = true;
                }
            }
        } catch (RETCODE_NO_DATA noData) {
            // No more data to read
            break;
        } catch (RETCODE_ERROR e) {
            e.printStackTrace();
        }
    }
}
```

iv. Results

Project was implemented successfully. When the message which includes randomly generated temperature is sent from the application, the subscriber receives it, and prints it to the console. At the same time, MQTTLens receives the message, too. Also, a message can be published from MQTTLens, and it can be seen in the console. Moreover, a data can be sent via Postman, and it can be also seen in the console and in MQTTLens. Test visualization and the application model are given in the next pages.

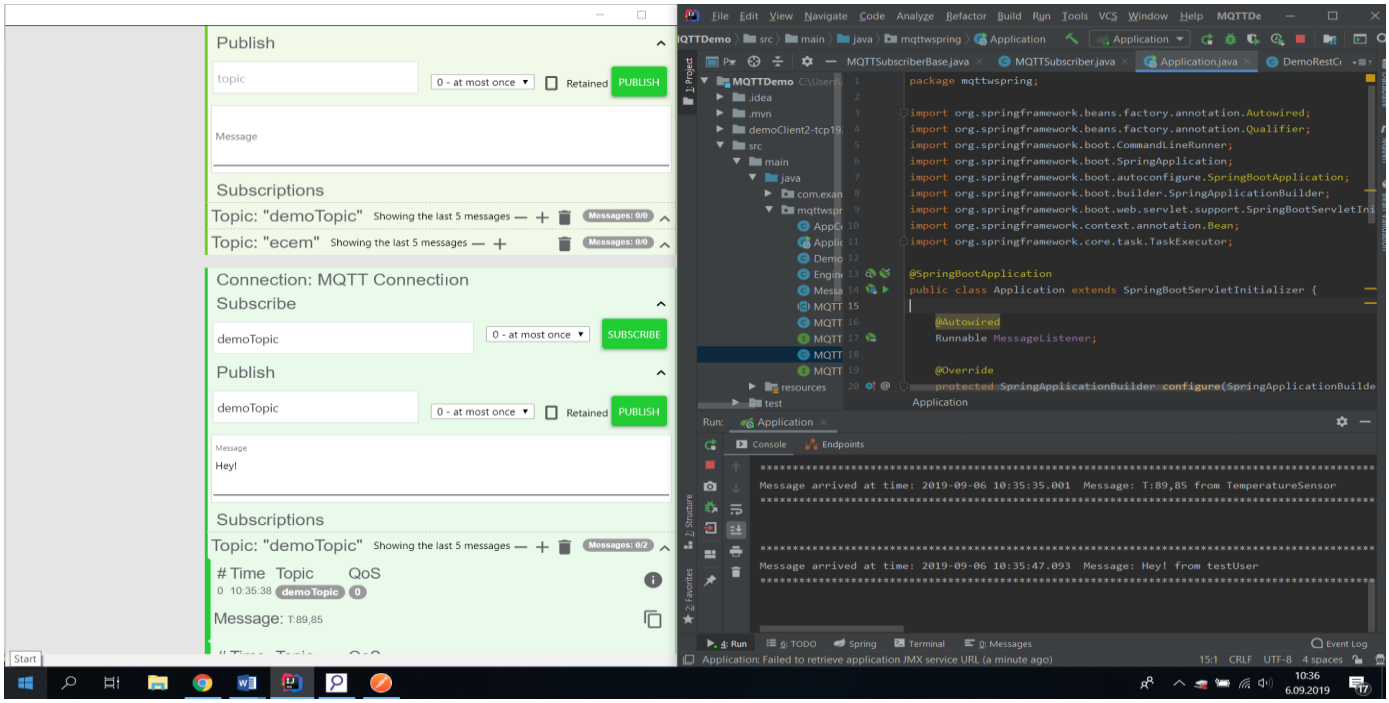


Figure 3

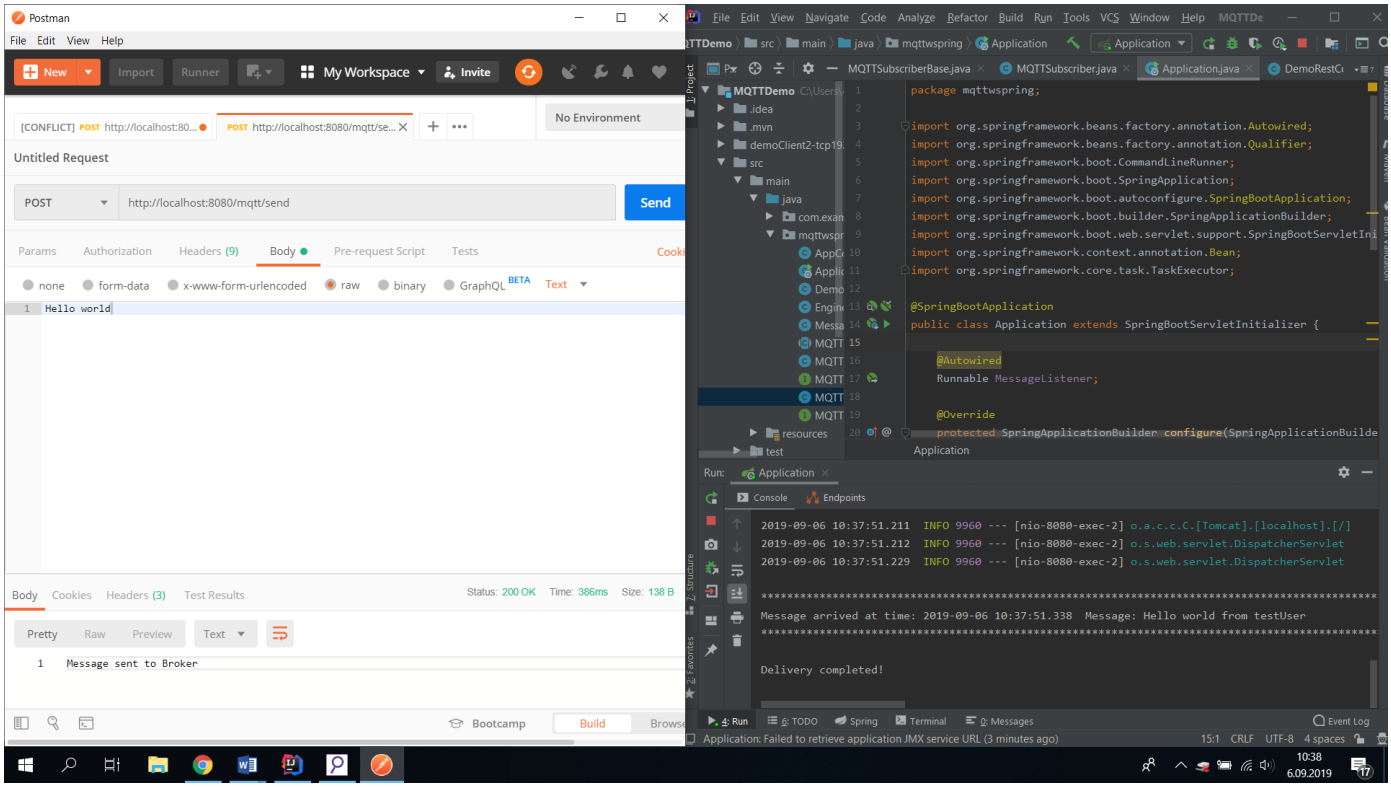
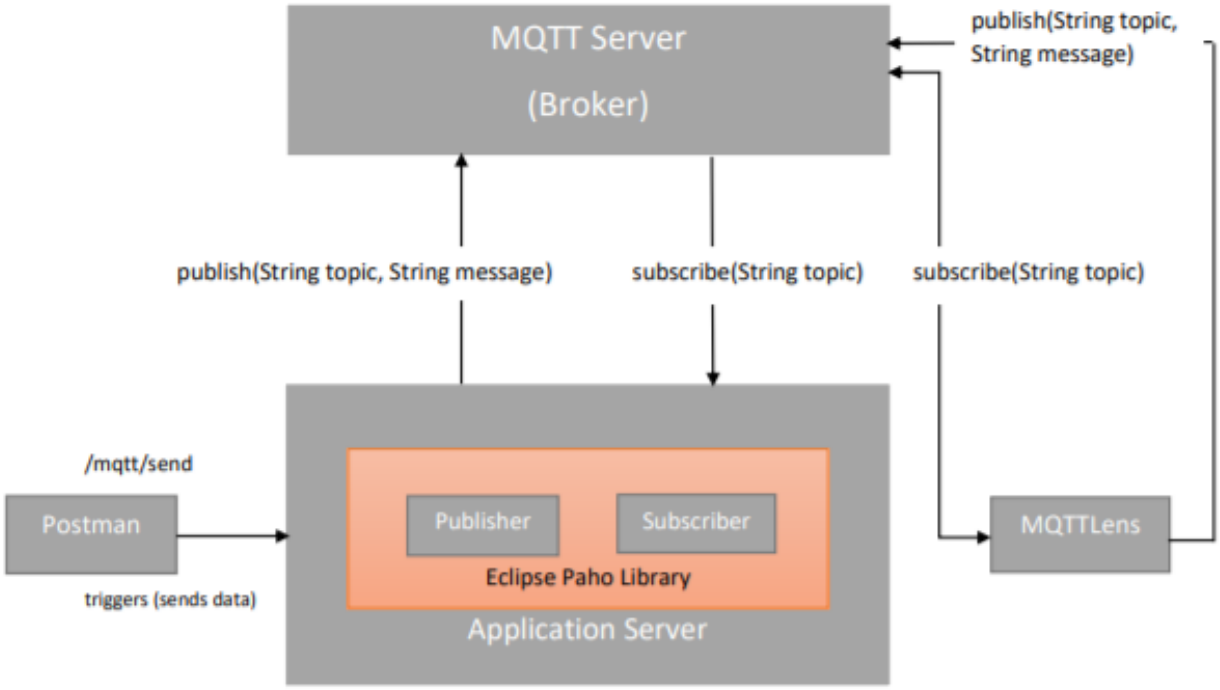


Figure 4

APPLICATION MODEL



Application Flow

Part 1 (Postman)

1. Postman sends data via REST API.
2. The server receives data and sends to broker with the help of Eclipse Paho's publisher.
3. The subscriber receives data from broker and prints it to the console.
4. MQTTLens also receives same data from the broker

Part 2 (MQTTLens)

***Also, MQTTLens can send messages to the MQTT server.

1. MQTTLens sends a message to MQTT server.
2. The subscriber receives the message from broker and prints it to the console.

***Additionally, MQTTLens receives a message coming from the Application.

1. A message is published to a specific topic from the Application server.
2. The subscriber receives the message from broker and prints it to the console.
3. Also, MQTTLens receives the message from broker.

However, the DDS implementation was not finished, because of an error. Although I imported the particular library to the Path, I struggled with the error informing that “The library nddsjavad.dll could not be loaded by Windows. Make sure that the library is in your Path environment variable.” The runtime errors’ screenshot is below.

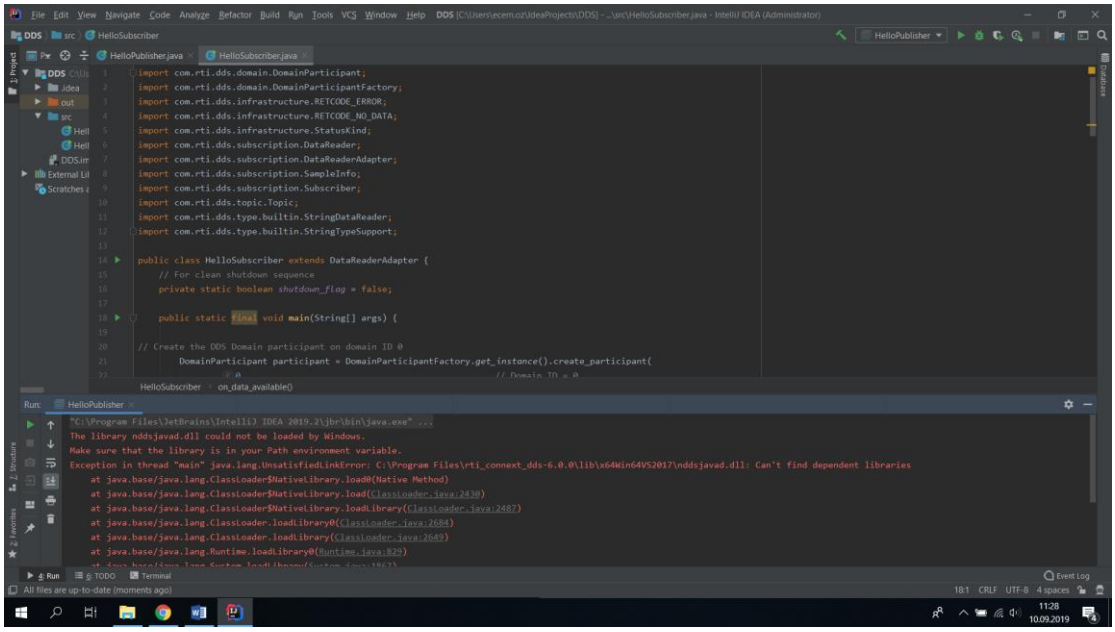


Figure 5

Conclusions

In this project, I have learned how can MQTT protocol used in Java applications using the library provided by the Eclipse Paho project and Spring Boot framework. The competence I have in Java simplified the work that is done and reduced the effort expended to finish the project. This Java knowledge basically comes from CS101 and CS102 lectures. In CS101 (Introduction to Programming), developing and debugging computer programs to solve engineering problems, understanding the basics of project management and software engineering, being able to implement programs that have sequential, iterative/looping, and switching flows, demonstrating ability to develop both console based and graphical user interface (GUI)-based applications, being able to reuse and repurpose classes and libraries developed by other people were explained in detail. In addition to that, CS102 (Object Oriented Programming) was a lecture that includes implementing an object’s necessary interface, using static and non-static methods appropriately, using main collection objects, writing programs that make use of polymorphism and writing programs that throws exceptions, in other words exception handling. CS101 assignments were mostly logic based. They are solved by only one class which includes main method and optionally other methods. On the other hand, CS102 assignments consists of multiple classes and their relation between. In this project that I have done for the intern, there are numerous classes that are related with each other. It can be accepted that in this point of view, the project with CS102 assignments

are common. However, the difference is that it does not include such logical operations as seen in CS101.

This intern will be helpful for understanding the CS447 lectures which is about the terminology and concepts of computer networks, learning applications of computer networks, learning the data communication and share, understanding the network layers and IP addressing and subnetting. In the sense of writing codes more quickly and more efficiently, this intern will be beneficial for my education life, and eventually for my career. Additionally, this project made me understand the concept of class relations, how they connect to each other, so on.

The skills and qualifications I have acquired in this internship are researching, programming with Java and improvement in Java knowledge, Maven & Spring Boot, using different libraries like Eclipse Paho, object-oriented design –abstraction, inheritance, polymorphism, so on-, software debugging, testing, problem solving, written communication, agile development processes and principles, strategic planning, web API, analysis. My future may be affected positively, because these qualifications and proficiencies are top needed skills of a software engineer. During intern, I have decided to work in software area, again. I made my decision according to the working conditions, office environment, teamwork, research area, so on.

My personal views about the ICTerra is positive. Although it is a new company which was established in 2013, it has well organized, qualified, knowledgeable, helpful workers. In addition to that, social environment exists, and I think it is so beneficial in order to motivate the workers. For example, once a month, Happy Hour is held. People eat and drink together, socialize, have good time, and become more motivated for work. They organize a lot of different activities like Happy Hour. Thus, a firm like ICTerra is absolutely a workable place.

Appendix

Turkey's Expert Company in Software Engineering: ICterra

27 Nisan 2017

ICterra's story started when the Turkish R&D team of Siemens decided to continue their work as a separate and national company. Today, ICterra stands out as an expert software engineering company in the defence and aerospace industry as well as in the civilian and public sectors.

Mehmet Arif SEZGİN: ICterra has a history going back over a quarter century. We started our journey as the R&D department of the multinational company Siemens in the year 1991. At the time, the company was developing software to support Siemens' various telecommunications products in the international market.

When Vedat Uslu, our CEO, took over the company's management and shares in 2013, we adopted the name ICterra, as well as identity of an indigenous software engineering company. Since then, we intensified our efforts to provide solutions and services for the national and international markets with our capabilities.

We work in a field where we can offer a wide range of benefits to our users. Today, information technologies have a vital role in many areas from the basic applications we use in our daily lives to the processes that help us manage and continue our lives, including the production and delivery of the milk on our tables, and even national defence.

We conduct activities abroad, especially towards the EU market. The software development projects in the field of telecommunications constitute a part of our activities. We continue our course by adding new projects to our existing ones in telecommunication. Furthermore, in recent times, we started a long term R&D collaboration with the Japanese company Wacom. We are involved in the development of their mobile applications in various products.

We are following our growth target nationally. Currently, we are focused on defence and aerospace projects, public projects and the corporate market. This year, we strengthened our activities towards the corporate market by opening our Istanbul branch. On the other hand, we have also made great advances by adding new projects to the ones we are currently conducting in the public sector.

We also pay special attention and particularly focus on the defence and aerospace industry. Having considered our technical competences and corporate experience, we decided that we could take responsibility in this field and we started working on it. Turkey carries out very important indigenous defence projects, and is continuing to add new ones on top of its existing projects. We are involved in some of those projects by providing software engineering services to prime contractors. Software in the defence and aerospace industries differs greatly from the software-related activities in other areas. That is why we first started with some infrastructural investments and preparations. The past experiences and corporate memory of ICterra helped us adapt very rapidly to the defence and aerospace industry.

ICterra: Global IT and Consulting Company

As a global information technology (IT) services and consulting company, ICterra has been providing software R&D services to different sectors for 25 years through its extensive knowledge and expertise in its domain, and developing innovative and customer-oriented products. The company offers IT solutions with high added value and high-quality consulting services based on special customer needs through the strong partnerships it established with the world‘s leading IT companies.

The areas in which ICterra offers solutions are as follows:

- Software Development,
- Software Maintenance and Support,
- Software Consulting,
- Software Testing,
- IT Service Management and
- Information Security

The company conducts activities in these areas through business models such as; turn-key projects, the provision of engineering resources, R&D partnerships, and strategic business partnerships.

References

Sandra Bruno 2012, RTI Connext DDS : Setup and HelloWorld Example (Windows/Eclipse/Java), accessed 7 November 2012, <https://blog.zhaw.ch/icclab/rti-connext-dds-setup-and-helloworld-example-windowseclipsejava/>

The HiveMQ Team 2015, Introducing the MQTT Protocol – MQTT Essentials: Part 1, accessed 12 January 2015, <https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/>

The HiveMQ Team 2015, Quality of Service 0,1 & 2 – MQTT Essentials: Part 6, accessed 16 February 2015, <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>

MSI Turkish Defense Review (TDR) 2017, accessed 27 April 2017, <http://www.milscint.com/en/turkeys-expert-company-in-software-engineering-icterra-2/>

Monir 2017, Eclipse Paho Java Client (MQTT Client) integration with Spring Boot REST API, accessed 22 November 2017, <http://www.monirthought.com/2017/11/eclipse-paho-java-client-mqtt-client.html>

Baeldung 2019, MQTT Client in Java, accessed 6 September 2019, <https://www.baeldung.com/java-mqtt-client>

RF Wireless World n.d., MQTT vs DDS in IoT | Difference between MQTT and DDS<https://www.rfwireless-world.com/Terminology/MQTT-vs-DDS.html>

Nurettin Mert Aydın-Group Manager (Energy), From Communication Protocols to Message Queueing: Evolution of Communication in Energy & Utilities, accessed 30 April 2019, <https://blog.icterra.com/from-communication-protocols-to-message-queuing-evolution-of-communication-in-energy-utilities/>