Risiko:
Zeitnahe Implementierung einer Message Oriented Middleware, mit der noch sehr wenig Erfahrung besteht. Dadurch kann keine Nachrichtenbasierte Kommunikation stattfinden.

Fail-Kriterium:
Sollte der Android Client oder der Server nicht fähig sein Channel's zu abonnieren und dadurch Nachrichten zu erhalten, oder nicht der Lage sein in einen Channel Nachrichten publizieren zu können gilt der PoC als gescheitert und es wird auf die Fallback-Option zurückgegriffen.

Exit-Kriterium:
Der PoC gilt als erfolgreich, wenn der Android Client und der Server eine Nachricht in einen Channel publizieren können, und der Server und der Android Client die diesen Channel subscribten diese Nachricht erhalten. Dies muss bis zum Freitag den 4.12.2015 funktionieren.

Fallback-Option: Sollte das angestrebte Exit-Kriterium bis zum angegeben Datum nicht erreicht werden, wird auf den externen Dienst Pub/Nub zurück gegriffen. Dieser ist bereits Implementiert und erfüllt das Exit-Kriterium.

Beschreibung:
Der Server und die Android Clients müssen in der Lage sein über Rabbit MQ Nachrichten auszutauschen. Dazu müssen beide in einen Channel publishen können und einen Channel auch subscriben können.

Durchführung:

Die Schwierigkeit in der Durchführung dieses PoC bestand in der Verbindung zwischen dem Android Client und dem Server. Das Problem hierbei lag beim Portmapping zwischen dem Android Gerät, und dem Computer auf dem der Server gehostet wurde.
Um dies zu lösen musste der localhost des Computer über den der Server erreichbar ist über Google Chrome und dessen Möglichkeit zum Portmapping dem Android Client bekannt gemacht werden.

Hier ist der Javascript Code, um dem Server Nachrichtenbasierte Kommunikation zu ermöglichen.

Send.js:

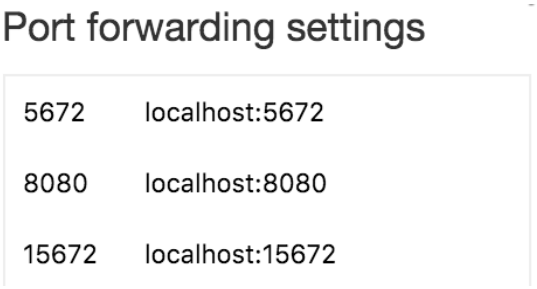
```
#!/usr/bin/env node

var amqp = require('amqplib/callback_api');

////Zum RabbitMQ Server der auf dem localhost läuft verbinden
amqp.connect('amqp://localhost', function(err, conn) {
  conn.createChannel(function(err, ch) {
    var name = 'chat';
    //Zusammenbauen der Message
    var msg = process.argv.slice(2).join(' ') || 'Hello World!';

    ch.assertExchange(name, 'fanout', {durable: false});
    //Nachricht an den Exchange mit namen Chat schicken.
    ch.publish(name, '', new Buffer(msg));
    console.log(" [x] Sent %s", msg);
  });

  setTimeout(function() { conn.close(); process.exit(0) }, 500);
});
```

Reciever.js:

```
#!/usr/bin/env node

var amqp = require('amqplib/callback_api');

//Zum RabbitMQ Server der auf dem localhost läuft verbinden
amqp.connect('amqp://localhost', function(err, conn) {
  conn.createChannel(function(err, ch) {
    var name = 'chat';

    // Create ein Exchange vo Typ Fanout mit dem namen "chat"
    ch.assertExchange(name, 'fanout', {durable: false});

    //Parameter 1 = Leer : Der Server weißt einen zufälligen Namen zu
    ch.assertQueue('chat', {exclusive: true}, function(err, q) {
      console.log(" [*] Waiting for messages in %s. To exit press CTRL+C", q.queue);
      //Binding zwischen der queue und dem exchange herstellen
      ch.bindQueue(q.queue, name, '');

      ch.consume(q.queue, function(msg) {
        console.log(" [x] %s", msg.content.toString());
      }, {noAck: true});
    });
  });
});
```

Der Android Client

Der Code auf dem Android Gerät ist im Gegensatz zum Server in Java geschrieben. Die Schwierigkeit den Localhost des Computers über das Android Gerät zu erreichen wurde mittels Portforwarding durch Google Chrome gelöst.

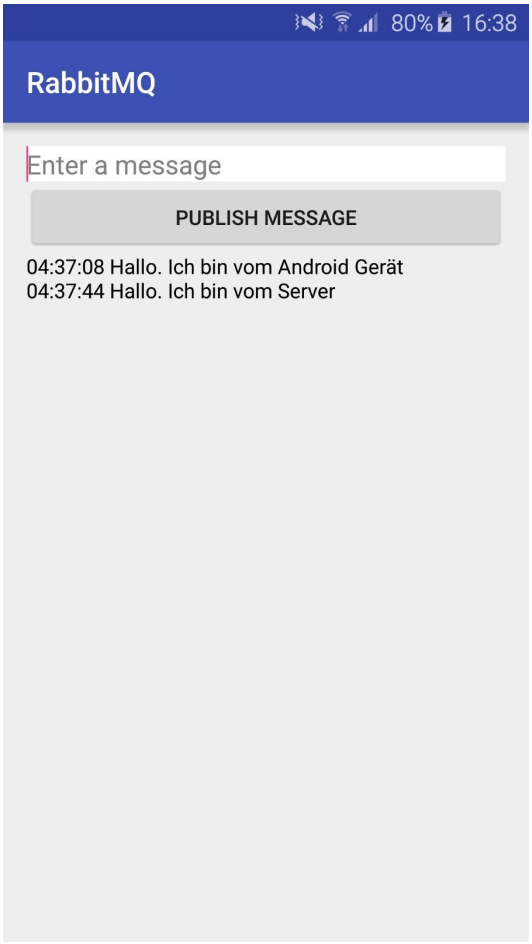


Die Relevanten Code Ausschnitte vom Android Client:

```
ConnectionFactory factory = new ConnectionFactory();
private void setupConnectionFactory() {
    factory.setHost("localhost");
}

void subscribe(final Handler handler)
{
    subscribeThread = new Thread(new Runnable() {
        @Override
        public void run() {
            while(true) {
                try {
                    Connection connection = factory.newConnection();
                    Channel channel = connection.createChannel();
                    channel.basicQos(1);

                    channel.exchangeDeclare("chat", "fanout");
                    String queueName = channel.queueDeclare().getQueue();
                    channel.queueBind(queueName, "chat", "");

                    QueueingConsumer consumer = new QueueingConsumer(channel);

                    channel.basicConsume(queueName, true, consumer);

                    // Process deliveries
                    while (true) {

                        QueueingConsumer.Delivery delivery = consumer.nextDelivery();

                        String message = new String(delivery.getBody());
                        Log.d("","[r] " + message);

                        Message msg = handler.obtainMessage();
                        Bundle bundle = new Bundle();

                        bundle.putString("msg", message);
                        msg.setData(bundle);
                        handler.sendMessage(msg);
```

```
public void publishToAMQP()
{
    publishThread = new Thread(new Runnable() {
        @Override
        public void run() {
            while(true) {
                try {
                    Connection connection = factory.newConnection();
                    Channel ch = connection.createChannel();
                    ch.confirmSelect();

                    while (true) {
                        String message = queue.takeFirst();
                        try{
                            ch.basicPublish("chat", "", null, message.getBytes());
                            Log.d("", "[s] " + message);
                            ch.waitForConfirmsOrDie();
                        } catch (Exception e){
                            Log.d("","[f] " + message);
                            queue.putFirst(message);
                            throw e;
                        }
                    }
                } catch (InterruptedException e) {
                    break;
                } catch (Exception e) {
                    Log.d("", "Connection broken: " + e.getClass().getName());
                    try {
                        Thread.sleep(5000); //sleep and then try again
                    } catch (InterruptedException e1) {
                        break;
                    }
                }
            }
        }
    });
    publishThread.start();
}
```

Die Ausgabe auf dem Android Client:



Die Ausgabe des Server:

```
[*] Waiting for messages in chat. To exit press CTRL+C
[x] Hallo. Ich bin vom Android Gerät
[x] Hallo. Ich bin vom Server
```