Implementation eines Publish-Subscribe -Nachrichtenservice mittels “RabbitMQ “

Es soll ein Publish-Subscribe-Nachrichtenservice mit RabbitMQ realisiert werden. RabbitMQ ist eine Message-Queue, die es ermöglicht durch verschiedene Clients (Java, Javascript, Node-JS, PHP, etc..) Nachrichten auf den RabbitMQ Server zu senden und durch das “subscriben” einer Queue Nachrichten zu empfangen.

Das Proove of Concepts soll zeigen, ob Nachrichten über ein bestimmten Topic vom Node-JS-Server an RabbitMQ gesendet werden können und anschließend von der Message-Queue an ein Android- und Node-JS-”Consumer” weitergeleitet werden können. Ebenso soll der Android-Client ebenfalls in der Lage sein Nachrichten an den RabbitMQ-Server zu schicken. Die “Consumer”-Clients sollen eine Präsentationslogik haben, die es uns gestattet die gesendeten Nachrichten aus der Message-Queue zu sehen. Ein Textfeld oder eine Konsolenausgabe soll in diesem Fall genügen.

Bezug zum Projekt

Dieses “Proove-of-Concept” ist für die Entwicklung des Projekts entscheidend. Die Ärzte müssen Verordnungen durch das System stellen können und das Pflegepersonal sowie die betroffenen Patienten müssen vom System benachrichtigt werden. Dazu ist eine Message-Queue notwendig, die diese Nachrichten an die jeweiligen Clients mittels Publish-Subscribe-Prinzip verteilt. Das Alleinstellungsmerkmal, dass der Patient mit in den Medikationsprozess einbezogen wird, ist daher ebenso in diesem POC adressiert.

Exit-Kriterium

Das “Proove-of-Conecept” gilt als erfolgreich, wenn alle Nachrichten des “Producers” an die “Consumer” weitergeleitet werden, die das entsprechende Topic abonniert haben. Die Nachrichten sollen auf den jeweiligen “Consumer”-Clients als Textfeld oder Konsolenausgabe aufgelistet sein. Dabei soll der “Producer” zuerst ein Node-JS-Client sein und dann ein Android-Client.

Fail-Kriterium

Das Proove-of-Conecepts gilt als gescheitert, wenn keine oder nicht alle Nachrichten des “Producers” an die “Consumer” weitergeleitet werden, die das entsprechende Topic abonniert haben. Das heißt, dass die Nachrichten nicht auf

den jeweiligen “Consumer”-Clients als Textfeld oder Konsolenausgabe aufgelistet sind.

Fallback

Als Fallback muss eine andere Message-Queue implementiert werden. ZeroMQ ist eine mögliche alternative, da die Entwickler ebenfalls offizielle Java- und Node-JS-Clients zur Verfügung stellen. Jedoch ist die Implementation dieser Message-Queue ebenso aufwendig wie das Implementieren von RabbitMQ. Aus zeitlichen Gründen könnte das vorhandene Wissen über Faye mit Node-JS herangezogen werden. Dabei gilt es allerdings zu beachten, dass eine Anbindung an einen Android-Client nicht in der offiziellen Library enthalten ist.

Prozessdarlegung

Zunächst müssen alle Komponenten, Programmiersprachen, SDKs, Librarys und Entwicklungsumgebungen installiert und die Umgebungsvariablen definiert werden (siehe Anhang Installation). RabbitMQ ist nach der Installation sofort nutzbar. Dies kann getestet werden, indem man über den Browser den “localhost:15672” aufruft. Über diese Adresse ist der standardmäßig mitgelieferte Manager von RabbiMQ erreichbar. Der Manager bietet einen Überblick über sämtlichen Datenaustausch und Verbindungen des RabbitMQ-Servers.

Da RabbitMQ nun lauffähig ist, müssen die Clients programmiert werden. Die Anbindung in Node-JS wurde durch das Package “”amqp” realiesiert. “amqp” ermöglicht uns durch eine knappe aber verständliche API eine zügige Implementation für einen Node-JS-“Producer” und -”Consumer”. Für das POC wurden diese beiden Komponenten getrennt programmiert.
Der “Producer” hat die Aufgabe, eine Nachricht mit dem routingkey=’verordnung’ auf den Exchange “amq.topic” zu senden. Der routingkey ist in diesem Fall unser Topic und der Exchange der Übermittler an die Queues. Exchanges können aus den Clients heraus generiert werden und befinden sich dann auf dem RabbitMQ-Server. In diesem Fall ist “amq.topic” ein Exchange, der standardmäßig auf dem RabbitMQ-Server zur Verfügung steht und für unser POC verwendet wird. Der “Konsument” soll nun die Nachrichten des “Producers” empfangen. Dazu muss eine Queue entweder auf dem Client generiert werden und mit dem routingkey=’verordnung’ an den Exchange ‘amq.topic’ gebunden oder entsprechend über den RabbitMQ-Manager erstellt und gebunden werden. Die Verbindungen sind nach dem Start der Node-JS Komponenten direkt im Manager ersichtlich.
In dem Server des “Producers” wurde eine Route auf den localhost/ definiert. Über diese Route ist eine HTML-Datei erreichbar, in der ein Eingabefeld definiert

ist. Dadurch soll das Verfassen einer Nachricht ermöglicht werden. Die Nachricht wird dann per http-post an den Server übermittelt, der die Nachricht dann an den RabbitMQ-Server übermittelt.
Die gesendeten Nachrichten waren in dem Konsolen-Log des Konsumenten ersichtlich.

Da der erste Schritt erfolgreich war, kann nun der Android-Client programmiert werden. Die Java-API für die RabbitMQ-Library ist deutlich umfangreicher und stellt zu Beginn ein Problem dar. Es mussten gründliche Recherchen durchgeführt werden, bevor zur Implementation übergegangen werden konnte.
Zunächst wurde eine einfache Activity mit einem Eingabefeld und einem Ausgabefeld erstellt. Dies genügt bereits den Anforderungen zur Erfüllung des POC.
Das Erstellen einer Verbindung zum dem RabbitMQ-Server via Android stellte scheiterte vorerst. Für die Verbindung musste ein Thread programmiert werden und geworfene Exceptions mussten abgefangen werden. Daraufhin wurde ein weiterer User “test” im RabbitMQ-Manager angelegt, da weitere Recherchen zeigten, dass eine Verbindung mit dem Standard-User “guest” nur über den “localhost” des Rechners möglich ist, auf dem der RabbitMQ-Server läuft. Zusätzlich war es notwendig der Applikation im Android-Manifest die Erlaubnis zur Nutzung der Netzwerkverbindung zu erteilen.

Nach diesen Einstellungen konnte eine Verbindung zum RabbitMQ-Server hergestellt werden. Nun musste eine Queue erstellt und mit dem routingkey=’verordnung’ an den Exchange gebunden werden. Nach einigen Versuchen und Fehlerbehebungen konnten Nachrichten vom Android-Client verschickt und auch von dem Node-JS-”Producer” empfangen werden.

Auffallend war allerdings, dass die Queues, die vom Client erstellt wurden nur temporär auf dem RabbitMQ-Server angelegt waren. Dies hat den Nachteil, dass Nachrichten auf dem Topic nicht an die Clients weitergeleitet werden können, wenn diese keine Verbindung mehr zum RabbitMQ-Server haben und sich anschließend wieder verbinden. Nach weiteren Tests und Recherchen wurde ersichtlich, dass das Anlegen von Queues durch verschiedene Methoden möglich ist. So konnten Attribute wie “non-autodelete” und “durable” der Methodenschnittstelle mitgegeben werden. Die Nachrichten auf den Topics wurden dann wie gewünscht auf der Queue solange abgelegt, bis der jeweilige Client wieder vom RabbitMQ-Server erreichbar war.

Status

Die programmierten Komponenten wurden problemlos nach den Exit-Kriterien getestet. Der “Proove-of-Conecept” ist demnach erfolgreich abgeschlossen und die getesteten Technologien können für die weitere Entwicklung genutzt werden.

Ausblick

Für die weitere Entwicklung muss nun eine geeignete Technik gefunden werden, wie die für das Projekt benötigten Topics und Sub-Topics in RabbitMQ realisiert werden sollen. Außerdem muss das Verhalten der Exchanges und Queues weiter analysiert werden, damit die Anforderungen aus dem Projekt erfüllt werden können und die Kommunikation zwischen den Benutzern über das System einwandfrei funktioniert. Des weiteren soll der in diesem POC entwickelte Android-Client als Grundlage für den nächsten POC dienen, wodurch die “gepublishten” Nachrichten als Benachrichtigung angezeigt werden.

Anhang:

Installationen:

1. Erlang
2. RabbitMQ-Server
3. Node-JS
    a. express
    b. amqp
    c. bodyparser
4. Eclipse
    a. ADT-Plugin
5. Android SDK herunterladen
6. RabbitMQ Java Client herunterladen