Chat Application

# Requirements

Below are the basic requirements for implementing a chat server and client based out of Java.

## Server Application requirements

1. Ability to host multiple chat rooms

a. SysOp launches server app and configures chat rooms via a configuration file.

b. Each room continues to live if the server application is running, even if no users are in the room.

2. Users login with no authentication

3. Each user can participate in one room only

4. Maximum of 16 users per room

5. Maximum of 16 rooms

6. Log every conversation in the room in a file

7. Certain words (offensive) may be configured to ban. For example, if SysOp configures (via a config file) words say "fool", "religion" to be banned. Then such messages that contain this word will be replaced with "XXXX".

## Client application requirements

1. Ability to join a room with a given user handle

2. Display all participants in the room when user joins

3. Ability to send message to the room

4. Ability to receive messages

# Design Approach

The basic design involves using an available Messaging framework like RabbitMQ to enable communication between the chat server and chat client. Following are the key motivation and advantages of using this approach,

1. Message framework is readily available to use with well-defined clients.
2. Chat application includes chatrooms where multiple users joins a same chatroom, send / receive message which is analogous to the publish/subscribe messaging mechanism in the Rabbit MQ.
3. Built-in support for performance tuning, high availability and redundancy if required.

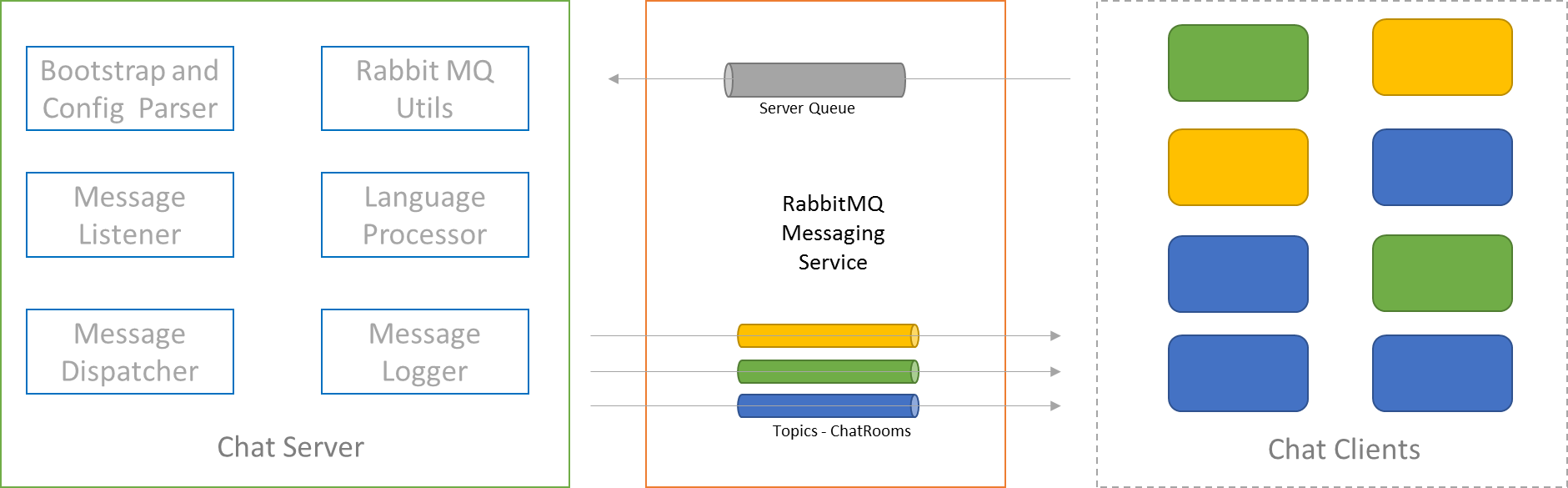
## Other Design Options

1. Creating a socket based communication interface and custom messaging infrastructure for the client-server communication
2. Event driven messaging frameworks natively available in Java.

## High level Architecture

## Chat Server

Basically, the all the chat clients in all the chat rooms will send the chat messages to the server along with the user information and the chat room details. The server validates the message, logs into a file, performs a language processing and publishes the message to the specific chatrooms by means of a Topic. Each chatroom has a corresponding topic created during the server startup.

The Chat Server consists of the following components

|  |  |  |
| --- | --- | --- |
| # | Component | Description |
| 1 | Bootstrap and Config Parser | Responsible for initializing the RabbitMQ listeners and parse the configuration files containing the details of Chat Rooms and Abuse Words.  It then spawns a thread and wait for incoming messages from the clients. |
| 2 | Message Listener | This module will listen to the incoming messages from the chat clients in a queue called “server\_queue”. Once the message is received, it invokes the language processing APIs and the sends to the Message Dispatching modules. |
| 3 | Message Dispatcher | This module will post the processed message back to a RabbitMQ Topic, so that its sent to the chat clients subscribed to that specific topic |
| 4 | Rabbit MQ Utils | This is a wrapper class to abstract all the RabbitMQ specific functionalities. |
| 5 | Language Processor | This is to replace the pre-defined abuse words with the appropriate markers configured in the configuration file. |
| 6 | Message Logger | This is to all the messages to a file |

## Chat Client

The Chat client is a small Java swing based application performing the following functionalities,

1. To begin with, provide options to select the available chatrooms that the user can select.
2. Once the user selects the chatrooms, it also fetches the other users already logged in to that specific chat room.
3. Listen to the topic corresponding to the chatroom, once the user confirms the chatroom to join.
4. When the user types in a message, the same is sent to server along with the chatroom and user details.
5. When the user closes the client application, notify the server and log out.

# Deployment Options

## Docker

The plan is to deploy the server in a docker container and RabbitMQ in a separate docker container (already available in the docker hub)

The chat client must manually be downloaded for now or can be made available to download via running a webserver. Manually downloading the client is preferable in the short term.

The docker-compose will be used to bring up the chat server and Rabbit MQ container.

## Creating JAR files for Server and Client using Maven

Have the maven configuration to create the jar files along with the required dependencies.

## Preferred approach

Docker based approach is preferred considering the ease of deployment and not requiring the user to install the dependencies

# Sample configuration Files

## Chatroom Configuration

<chatrooms>

<chatroom>

<name>baseball</name>

<type>sports</type>

<maxusers>16</maxusers>

</chatroom>

<chatroom>

<name>football</name>

<type>sports</type>

<maxusers>16</maxusers>

</chatroom>

<chatroom>

<name>Tourist Attraction in SFO</name>

<type>Travel</type>

<maxusers>10</maxusers>

</chatroom>

<chatroom>

<name>Restaurants</name>

<type>Food and Living</type>

<maxusers>16</maxusers>

</chatroom>

</chatrooms>

## Language Configuration

<LanguageProcessor>

<AbuseWord>

<name>fool</name>

<type>Personal</type>

<replacement>XXXX</replacement>

</AbuseWord>

<AbuseWord>

<name>stupid</name>

<type>Personal</type>

<replacement>YYYY</replacement>

</AbuseWord>

<AbuseWord>

<name>Religion</name>

<type>Religion</type>

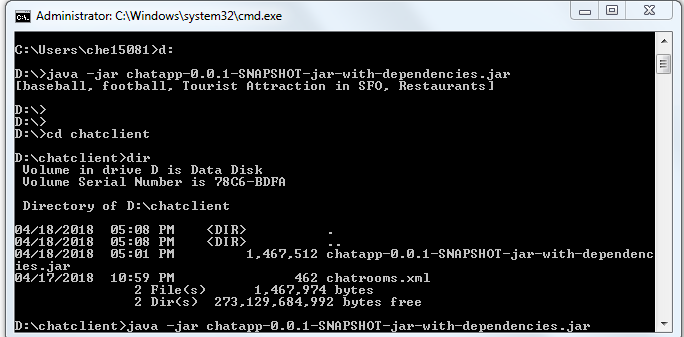
<replacement>XXXX</replacement>

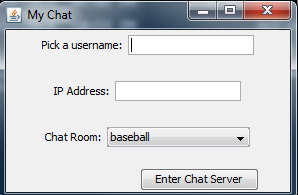
</AbuseWord>

</LanguageProcessor>

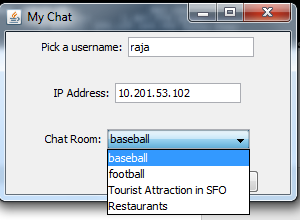
## How to start Chat Client

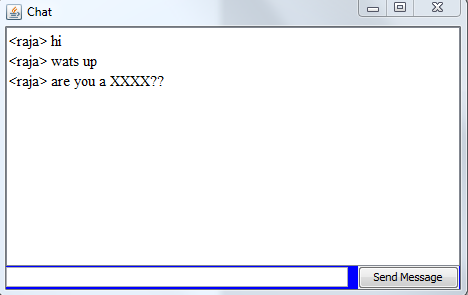
1. Download the Client jar file provided in the repository
2. Place the ***chatrooms.xml*** in the same folder hierarchy as that of the jar file
3. Execute the command “java -jar chatapp-0.0.1-SNAPSHOT-jar-with-dependencies.jar”





Provide the username , IP address of the host where the Chat Server Docker container is running and chatroom details above.





The abuse word fool is masked with XXXX in the chat above.

**Multiple chat clients can be launched each with different set of rooms and verified.**

**Logging**

Logs specific to the chat room is logged in a separate file within the server docker container.

Open Item

1. The list of chatrooms has to be passed as input as an xml file to client as well as server. The ideal approach should have the client get the list of rooms during startup. Did not have time to fix that and for now its configuration.