AffinitasChat Solution Architecture

# Overview

This document covers software solution architecture for AffinitasChat. It makes sure that all functional and non-functional requirements are achieved.

I was thinking to provide UML class, state and sequence diagrams. I did not do that since I though it may be redundant, but if you think this will add a value I will be more than happy to do so.

# Requirements

## Functional requirements

 Super simple chat service and a very basic front end using it.

## Non-Functional Requirements

1. Requirements are changing a lot and the system design should be flexible to adapt to these changes.
2. System should be scalable.

# Scalability and Performance

## Overview

Since requirements did not specify the scalability and performance goals the system must achieve, this document will only discuss general goals and how to achieve them.

Looking from an out of the box view, this system is a set of functionalities that can be distributed on a set of servers and database. Based on that this section will discuss databases, data retrieval and notifications, functionality decomposition and servers interactions.

## Database

The assumption here is data integrity is not an issue in this system and thus relational database can be put aside and replaced by high performance nosql databases.

## Functionality Decomposition

It looks like the functionality of this system can be divided to Notification functionality, Messaging functionality, and Users management’s band devices functionality. Every one of these can have its own web application and can be scaled alone depending on the need. Right now, the code reflects that by having three main packages: Messages, Users and Notifications. Every one of these packages can have its own maven module and thus its own web application. A data module will be common by all of these modules, which can be collected from all the three packages. Notice that this common module will have User Message, Device and etc, but it will never have any of the repository classes.

Every one of these system can call the other system rest interface, but it will not know about its services classes. In the code I am using services classes directly but in the reall system this need to be changed to use rest end points. TODO comments are in the code to notify about that.

Cloud methodology can be used and system can be auto scale.

## Asynchronous

Another point about improving performance and scalability is by using asyn calls. For example doing notification can be async call and no need to hold the call until the notification completed, especially notification can be time consuming.

Notification management is needed and missing in this application, like what will happen if the device is offline, when to retry, and etc.

In the code you can find TODO talking about this, but this is not implemented in this example.

## Push and Pull

Using pull mechanism to get user chat messages will increase traffic on the system without and need. Push notification is recommended in this chat system and got implemented in this system through notifications module.

In the code you will find both pull and push implemented just to demonstrate the idea.

# Logging and Monitoring

With scalable system looking and monitoring become essentials to be able to debug, evaluate, and monitor and potential defects or improper behaviors.

I am using spring AOP, but Logger class code is not good and I used very old logger that I had long time ago. But this can be easily replaced.