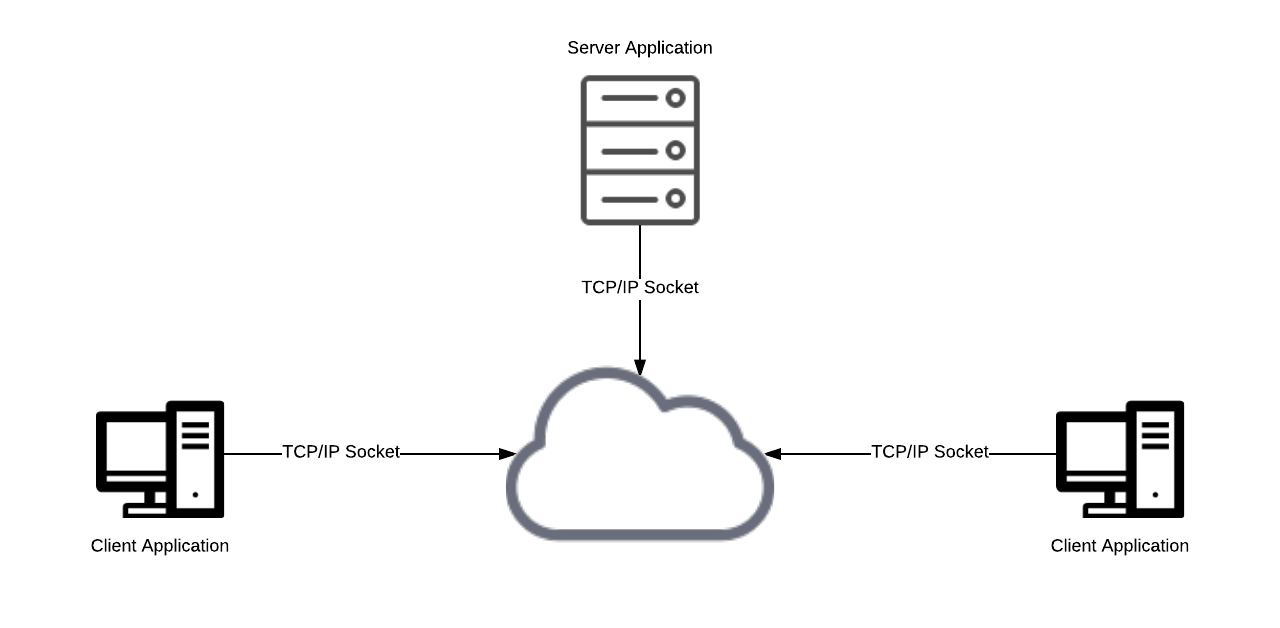
Candidate Architectures

# Component Topology

The Messaging service is fundamentally an IP dependent system and the use and exploitation of Internet Technologies. The topology arrangement of Components are implemented as a Client Server Model. This model is a typical Model used in many communication protocols, and will support the operational function and behaviour of the Messenger Service. The core goal of the Service topology is to provide a Middleware-oriented Service for messaging through non-heterogeneous networks.The Middleware-oriented Structure will justify the support for the critical NFR.

  
Figure 1

# Core Communication Protocol

The Messaging Service requires a common communication framework to meet the requirements of the project. XMPP is a long standing and popular communication protocol for messaging and presence services. Development on the protocol starting in 1999, it has long term support and is used on significant Platforms such and Microsoft Lync and Atlassian HipChat. The Platform also has a large base of matured Open Source projects.

|  |  |  |
| --- | --- | --- |
| **Key Mechanisms** | **Requirements** | **Priority** |
| IEEE International Standard, Open Source Platform | NFR 3.3 | Medium |
| Middleware-oriented Architecture | NFR 4.2, 4.3, 3.4 | High |
| P2P Encryption, PKI, SASL and TLS | NFR 4.1 | Medium |
| Asynchronous Messaging Relaying. | NFR 3.2 | High |
| User Presence Relaying. | NFR 2.7 | High |
| Centralised Database | NFR 4.4 | Low |
| Architectural Neutral | NFR 3.3 | Medium |

# Table 2

# 

# XMPP Server Implementation

XMPP Server should use industry standard database systems for storage of the chat data. This requirement will ease the transition to more performant infrastructure in the event that demands on the system outpace the initial capacity. (**Business Requirement 12**)

The server will be located in a data centre (or platform as a service) with a minimum guaranteed up time of 99.9% (i.e. less than 45 minutes offline per month).(**Business Requirement 13**)

XMPP implementations support the use of directory services. This means it is possible for users to have single sign on authentication to use the service. As with corporate email, Pinnacle will make it clear to users that in the event of an investigation, user’s chat data may be accessed. This will be possible because all chat communications will be stored on the database which is controlled by Pinnacle (**Business Requirement 14**).

|  |  |  |
| --- | --- | --- |
| **Key Mechanisms** | **Requirements** | **Priority** |
| Central Administration | NFR 4.2, 4.3 | Medium |
| SSL/TLS support | NFR 4.1 | High |
| Native Data Storage | NFR 1.3 | High |
| Platform Independent | NFR 3.3 | Medium |
| Can support large user base >10,000 | NFR 1.1 | High |
| Archiving and offline message retrieval | NFR 1.2 | High |
| Multi User Chat | NFR 1.1 | High |
| User Administration | NFR 1.4, 1.5, 1.6, 1.7, 1.8 | High |

Table 3

# XMPP Client Implementation

# Client API

* 1. **Assumptions**
     1. **Java:**  
        We have decided to use Java for the client side, as we all know java quite well. Java is also quite high level and therefore fast and easy to work in (**NFR 3.5**), tried and tested, and secure (**NFR 3.6**). Java being strictly object orientated will be helpful as we are designing and modelling this system in an object oriented way. Finally Java’s cross platform capabilities will make it easy for us to have our client side application be compatible with windows, os x, and linux (**NFR 2.2**). Therefore only Java API's will be considered.
     2. **XMPP:**  
        We have decided to use the XMPP protocol, as justified in **section 2.1**.  
        Therefore only XMPP API’s will be considered.
  2. **Candidate API’s**  
     1. **Smack**-Opensource

-Pure Java

-JVM 1.7 or higher

-gradle or maven highly recommended

-provides higher level constructs to allow programmer to work at a higher level than XMPP protocol.

-well tried and tested

-has its origins with Openfire server

-Dont need to be familiar with XML or XMPP XML format

-last commit 7 days ago

-Very good documentation

* + 1. **Babbler**  
       -Quite young

-Might change

-Less well tried and tested

-last commit 2017

* + 1. **Emite GWT**  
       -Bosh technique

-GWT

-Pure Java

-Maven

-Well tested (JUnit)

-Stable

-last commit 4 years ago

* 1. **Choice:**  
     **Smack**  
     Since we are likely going to use the Openfire server, this makes Smack a good choice for the client side API, as Smack has its origins alongside Openfire, and therefore they fit together very well.  
     Smack is open source, allowing for free use and easy extension.   
     It is still maintained, with the last commit being only a week ago.  
     It has been around for a long time, and therefore is well tried and tested, as well as having very good documentation available.  
     It is pure Java, which fits well with our philosophy.  
     It provides high level constructs to make programming easier, and allow a higher level focus. Knowledge of XML or the XMPP XML format is not even necessary.

|  |  |  |
| --- | --- | --- |
| **Key Mechanisms** | **Requirements** | **Priority** |
| Architecture Neutral Pure Java Implementation | NFR 3.5 | Low |
| Opensource | NFR 2.6 & 3.1 | High |
| Object Oriented | NFR 3.1 & 3.5 |  |
| Abstraction from XMPP protocol, XML, and XMPP XML format. | NFR 3.5 | Medium |
| Great documentation | NFR 3.5 | High |
| Works well with openfire server |  | High |
| Gradle/Maven support |  | Medium |
| Well tried and tested (Stable) | NFR 3.5 & 3.6 | High |
| Multi-user chat | FR 2.1 | High |
| Presence of other users | FR 4.5 | High |
| Contact Lists |  | Medium |
| Instant Messaging | NFR 3.2 | High |
| Multiplatform | NFR 2.2 | High |
|  |  | High |
|  |  | High |