System Integration: ChatX

Anders Munkgaard, Jackie Engberg Christensen, Michael Dolatko, Patrick Petersen
December 17, 2015

University College Nordjylland
Teknologi og Business
Sofiendalsvej 60
Phone: +45 7269 8000
Fax: +45 7269 8001
http://www.ucn.dk/

Theme:
System Integration: ChatX
Project period:
Spring 2014

Author(s):

Names

Supervisor: **Printed Copies:**
Total Pages:
Appendix:
Completed December 17, 2015

Synopsis:

This rapport covers the development of a online multiplayer game taking place in the real world by using a smart phone. We will talk about other Alternative reality or virtual reality games taking place in the real world, describe some general ideas behind the games and our own game. We will also talk about how the phones hardwares could be used in such games, and which implementations we have included and which we could have included to have made the game better. The game is implemented in Java using the Android API and Google play services. The server side is also made in java but runs on a computer and uses socket to communicate with the clients playing the game.

Contents

1 Overview

1.1 Introduction

This report has been worked on as part of the system integration course, with the purpose of learning to develop an application that makes use of multiple technologies taught in the course. The application is known as ChatX, an instant messaging(IM) system that provides client-side RSA encryption for its users and will make it difficult for unwanted third party listeners to understand the messages. This report will describe how the process was planned, how the application was designed, the implementations of our solutions and lastly conclude on the project, alongside with thoughts and considerations of what could be improved on in the application and the process.

2 Process

This chapter will cover the process used whilst developing ChatX. An agile method was adopted but a few practices that were outside of the chosen method also have seen some use, such as Test First and Pair Programming.

2.1 Kanban

Due to time pressure and considering how agile Kanban is, it was deemed the best methodology to follow for ChatX. Kanban has few rules and encourages changing flow of process if it helps a development team to deliver more value to the software, it also puts emphasis on how important visualization of work flow is therefore a virtual Kanban board (figure 2.1) was made.

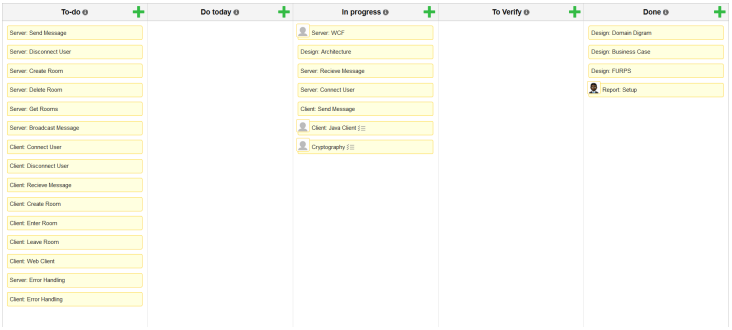

Figure 2.1: Kanban board

This Kanban Board has five columns. To-do, Do Today, In progress, To verify and Done. The To-do column on the Kanban Board is used to keep track of the tasks that are left to be completed in order to complete the entire project. The next column is Do Today, tasks from To-do are picked and then put over to Today. This helps keep track of how fast the work flow is in the team and whether it can handle more tasks each day or needs to take a step back and do fewer. In progress column shows which assignments are being

currently worked on, it helps to show which tasks are already taken so that code work won't be duplicated and not wasting working hours. Once a task is completed it is then being put over to the To Verify column where other team members have to take a look at the results of the task and verify it's been made properly. Once a task has been verified it is then moved to the final column which is named Done, it shows all assignments that have been completed.

2.2 Reflection

Unfortunately, the methodology rules that were set have barely been followed. Due to time pressure and size of the project the kanban was not used, and instead assignments and ideas were communicated through direct communication. It was also difficult to follow the methodology since the design of the system changed a lot and frequently, the final draft of the design ended up being very different from the original one. In retrospect it would have been better if the methodology was decided on earlier, and more effort into trying to follow it, even if the project isn't that big.

3 System Design

3.1 Business case

ChatX is a chatting system which will allow users to communicate with each other from where ever they are and without fear of the messages being read by unwanted users, as long as they use a platform that supports the application and have a Internet connection.

3.1.1 FURPS

FURPS is a architectural checklist. It helps describe what areas needs to be focused on, and an overall view of what ChatX's purpose is.

3.1.1.1 Functionality

Functionality deals with requirements the customer has set for the project. To help the customer determine the requirements, a question to ask is: What problem is the system going to solve? And with ChatX, users that are in need of communication with other users from a long distance on a secure connection, making sure that messages are private, and only seen by the intended users.

3.1.1.2 Usability

Usability describes how accessible ChatX is for the users. Is it easy to use? Is the GUI design intuitive? A way to check if the users like the aesthetics (and more importantly, can see how to use it) would be to prepare usability tests and alpha/beta tests as they can give crucial feedback on how user friendly a system is.

3.1.1.3 Reliability

Reliability describes how reliable the system should be: How much downtime can it handle? Are failures predictable? How is the system going to recover in case of failure?

Due to ChatX's architecture it is possible to setup several servers at once and if one fails, then the other servers can be used instead. In worst case scenario if all servers go down, then it would not take too long to install the server software on a different one and ChatX would continue working.

3.1.1.4 Performance

Performance describes how fast an application must be, what the highest allowed response time is and how much memory is needed.

ChatX is a chatting system, meaning that messages has to be sent and received as quickly as possible, however we can only assure a fast reliable service at the server end with optimal coding.

3.1.1.5 Supportability

Supportability, the last of the FURPS principles, describes how maintainable an application is after it is deployment. It also describes how easy it is to install and configure.

For ChatX two installations are required, the "server" installation has to be put up on a server, and a client has to install the java "client" on their computers. ChatX has been developed with expansion in mind, as earlier mentioned in case of server side errors, another one can easily take its place. The same principle applies here, if for whatever a reason someone chooses to shut down the server, it would be easy to simple install the "server" software on a different machine.

3.2 Architecture

ChatX's architecture will be a client-server architecture. The client will call the server whenever a user requests to use the system and the server will keep track of how many users are online and where each user messages is suppose to be sent to.

3.2.1 Domain Model

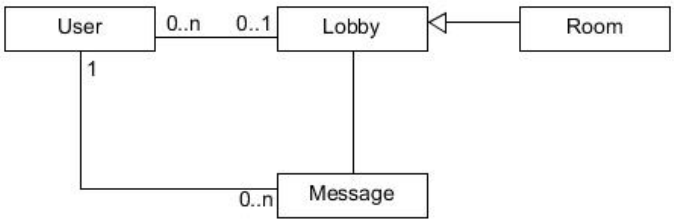

Figure 3.1: Domain Model over ChatX

The domain model consists of four classes. The user class is where users are identified and if they are regular users or administrative users, a user can only be in one lobby at a time but a lobby can have multiple users. A user can also send none to many messages. The message class is used for sending messages between users, a user can have none to many messages, whilst a message can only belong to one user. The lobby is where all users start. From the lobby a user can create a room and choose which users are to be included in it, Or join a room. The room class inherits the functionality of the lobby class.

4 Implementation

The implementation section below will explain what technologies were used and how they were implemented into ChatX.

4.1 Code Standards

Since this project makes use of two different programming languages it would be unwise to not set a code standard. So a code standard was set for each programming language.

4.1.1 Java

Class names must start with a capital letter. Variables must start with lowercase letters, private ones should be refereed by using ".this". Curly brackets are to be in their own lines at all times. Method names must start with a lowercase letter.

4.1.2 C#

Class names must start a capital letter. Variables must start with lowercase letters, private ones should be refereed by using ".this". Curly brackets are to be in their own lines at all times. Method names must start with a capital letter.

4.2 Client

The "Client" is what the person at home will use to use the chat system. There will be two different clients to cater to as many as possible.

4.2.1 Java Client

The java client has been written as the first client and will be used as the test client as it is possible to have a console application.

4.2.2 Web Client

The web client will be what caters to most people since it's a thing that can be used on all platforms, such as phones, pc's, and even consoles. It will be the most difficult to make as well since it requires AJAX (Update a website without reloading).

4.3 Cryptography

To make the communication secure the clients implements an RSA encryption to encrypt all messages sent. RSA is a Public- Private key (asymmetric) encryption algorithm. RSA encryption is a standard for encrypting and is practically impossible to decode due to prime factorization. The general idea is that it is easy to find the product of two large primes, but it is hard to factor a large product and find the primes.

RSA was originally made by Ronald Linn Rivest along with Adi Shamir and Len Adlemanand published in their paper A Method for Obtaining Digital Signatures and Public-Key Cryptosystems[1].

To be able to apply RSA encryption, 6 variables are needed; d, e, n, p, q and φ(phi). d, p, q and φ(phi) should be kept secret at all time to keep this encryption secure. Let p and q be a prime number, lets for simplicity say $p = 7$ and $q = 13$. In practice these 2 numbers would be larger depending on the numbers of bits used for encryption. n is defined as in equation 4.1.

$$n = p \times q \tag{4.1}$$

In this case $n = 91$. Additional n have the bit-length dependent on security of the encryption. If the encryption should be 1024 bit, when n would be a 1024 bit number. φ or $\varphi(n)$ is found by using equation 4.2.

$$\varphi = (p - 1)(q - 1) \tag{4.2}$$

For this example $\varphi = 72$ and it is now possible to find e which is the public exponent. e is a random prime and have two requirements which must be met; e must be a relative prime of φ, ie. e and φ have no common factors and e must be an integer such that $1 < e < \varphi$. For this example let $e = 7$. The public key is now available as n and e is known and can be given out.

Lastly we calculate d which is the private exponent using extended euclidean algorithm. Equation 4.3 is used for finding d.

$$d = e^{-1} \bmod \varphi \tag{4.3}$$

From equation 4.3 $d = 31$ and it is possible to further check if this is correct by using equation 4.4 which in this case it is, and the private key is now defined by n and d.

$$e \times d \bmod n = 1 \tag{4.4}$$

Now where both the public and private keys are assigned, they can be used to encrypt and decrypt messages. If the message "Hi" were to be decrypted it would first need to be translated into a number, eg. as 8 and 9. It is now possible to encrypt both letter individually by using equation 4.5 which gives the output cipher text C, and as well as decrypt it again with equation 4.6.

$$C = M^e \bmod n \quad \text{Where M} < \text{n} \tag{4.5}$$

$$M = C^d \bmod n \tag{4.6}$$

In the computer world when encrypting data, a padding is often used. Padding is often used to fill out byte blocks. For instance, if the message "Hi!" would had the byte block [48 69 21]. This block size is only of 3 bytes, but the encryption algorithm might read a block of 8 at a time which would mean the block needs 5 more bytes. There is multiple way of padding a message, such as zero-padding [48 69 21 00 00 00 00 00], padding with the same value as the number of bytes [48 69 21 05 05 05 05 05][2]. When the padding is done a block can be sent to the encryption algorithm to produce the cipher text.

4.4 Server

4.5 Webservice

4.6 RabbitMQ

5 Test

6 Conclusion

ChatX have been a very design heavy system where a lot of thoughts went into how the system should be designed.

6.1 Future Work

There were a larger amount of features that did not get implemented due to the scale of the project, but it is possible to extend the system with these features in the future a minimal changes and additions to the current system.

A client solution that was suggested was a web client which would be accessible by a web browser and would also be the client which would require the largest changes as a web client can not yet fully use sockets in the way the system currently works. Instead there the web client could run in a loop requesting the webservice for new messages. Additional SSL would need to be implemented to make the solution secure and encrypted.

Voice over IP(VoIP) and video chat was an other technology that was considered to be implemented. A few changes could make this possible on the current clients, but for a web client either a plug-in and/or a cloud services would be necessary.

A though was also at the client side to be able to embed, for instance medias, such as videos and images directly into the chat and make it possible for to view without opening external tools or programs.

Integration of other systems

Own Encryption

Bibliography

[1] A. Shamir R.L. Rivest and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems, April 4, 1977.

[2] www.di mgt.com.au. Using padding in encryption. http://www.di-mgt.com.au/cryptopad.html.

Apendix