CHAT APPLICATION USING SOCKET AND MULTITHREADING

**High Level Design & Low Level Design**

The purpose of this document is to provide  a template for documenting both HLD & LLD.

**Document Control :**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Project Revision History** | | | | | | | | |
|  |  |  | |  |  |  |  |  |
|  |  |  | |  |  |  |  |  |
| **Date** | **Version** | **Author** | **Brief Description of Changes** | | | | **Approver Signature** | |
| 16/11/2022 | Version 1.0 | Group 7 |  | | | | Trainer | |
|  |  |  |  | | | |  | |
|  |  |  |  | | | |  | |
|  |  |  |  | | | |  | |

[**1. INTRODUCTION**](#_heading=h.nj8zbtwhd4pt) **5**

[1.1 Intended Audience](#_heading=h.2et92p0) 6

[1.2 Acronyms/Abbreviations](#_heading=h.6lba4laccon7) 6

[1.3 Project Purpose](#_heading=h.3dy6vkm) 6

[1.4 Key Project Objectives](#_heading=h.1t3h5sf) 6

[1.5 Project Scope and Limitation](#_heading=h.k098myssv8y9) 6

[1.5.1 In Scope](#_heading=h.97vis8isnen6) 6

1.5.2 Out scope 6

[1.6 Functional Overview](#_heading=h.lgync04pdrgx) 6

[1.7 Assumptions, Dependencies & Constraints](#_heading=h.3rig67t6kwi) 7

[1.8 Risks](#_heading=h.z3nty3qqbd6h) 7

[**2. DESIGN OVERVIEW**](#_heading=h.lm41yhoph0kr) **8**

[2.1 Design Objectives](#_heading=h.vb0glt6gpp15) 9

[2.1.1 Recommended Architecture](#_heading=h.44sinio) *9*

[2.2 Architectural Strategies](#_heading=h.2jxsxqh) 9

[2.2.1 Design Alternative](#_heading=h.z337ya) 9

[2.2.2 Reuse of Existing Common Services/Utilities](#_heading=h.3j2qqm3)

[2.2.3 Creation of New Common Services/Utilities](#_heading=h.1y810tw) 9

[2.2.4 User Interface Paradigms](#_heading=h.4i7ojhp) 9

[2.2.5 System Interface Paradigms](#_heading=h.2xcytpi) 9

[2.2.6 Error Detection / Exceptional Handling](#_heading=h.1ci93xb) 9

[2.2.7 Memory Management](#_heading=h.3whwml4) 10

[2.2.8 Performance](#_heading=h.2bn6wsx) 10

[2.2.9 Security](#_heading=h.qsh70q) 10

[2.2.10 Concurrency and Synchronization](#_heading=h.3as4poj) *10*

[**3. SYSTEM ARCHITECTURE**](#_heading=h.ynyjvlpxz1vb) **11**

[3.1 System Architecture Diagram](#_heading=h.44px9x2z276u) 11

[3.2 System Use-Cases](#_heading=h.gapa3au4l3hm) 12

[**4. DETAILED SYSTEM DESIGN**](#_heading=h.9jotnomypsl6) **13**

[4.1 Key Entities](#_heading=h.41mghml) 13

[4.2 Detailed-Level Database Design](#_heading=h.2grqrue) 13

[4.2.1 Data Mapping Information](#_heading=h.vx1227) 13

[4.2.2 Data Conversion](#_heading=h.3fwokq0) 13

[Converting the IP address(IPv4 format) from binary to standard text format.](#_heading=h.c0456sdibaze) 13

[4.3 Archival and retention requirements](#_heading=h.1v1yuxt) 13

[4.4 Disaster and Failure Recovery](#_heading=h.4f1mdlm) 13

[4.5 Business Process workflow](#_heading=h.2u6wntf) 14

[4.6 Business Process Modeling and Management](#_heading=h.19c6y18) 14

[4.7 Business Logic](#_heading=h.3tbugp1) 14

[4.8 Variables](#_heading=h.28h4qwu) 14

[4.9 Activity / Class Diagrams](#_heading=h.uc7g1lrl186u) 14

[4.10 Data Migration](#_heading=h.37m2jsg) 14

[4.10.1 Architectural Representation](#_heading=h.1mrcu09) 15

[4.10.2 Architectural Goals and Constraints](#_heading=h.46r0co2) 15

[4.10.3 Logical View](#_heading=h.2lwamvv) 15

[4.10.4 Architecturally Significant Design Packages](#_heading=h.111kx3o) 15

[4.10.5 Data model](#_heading=h.3l18frh) 15

[4.10.6 Deployment View](#_heading=h.1egqt2p) 15

[**5. ENVIRONMENT DESCRIPTION**](#_heading=h.7t1aw0k78i5r) **15**

[5.1 Time Zone Support](#_heading=h.2dlolyb) 16

[5.2 Language Support](#_heading=h.sqyw64) 16

[5.3 User Desktop Requirements](#_heading=h.3cqmetx) 16

[5.4 Server-Side Requirements](#_heading=h.1rvwp1q) 16

[5.4.1 Deployment Considerations](#_heading=h.4bvk7pj) 16

[5.4.2 Application Server Disk Space](#_heading=h.2r0uhxc) 16

[5.4.3 Database Server Disk Space](#_heading=h.1664s55) 16

[5.4.4 Integration Requirements](#_heading=h.3q5sasy) 16

[5.4.5 Jobs](#_heading=h.25b2l0r) 16

[5.4.6 Network](#_heading=h.kgcv8k) 16

[5.5 Configuration](#_heading=h.1jlao46) 16

[5.5.1 Operating System](#_heading=h.43ky6rz) 16

[5.5.2 Database](#_heading=h.2iq8gzs) 16

[5.5.3 Network](#_heading=h.xvir7l) 17

[5.5.4 Desktop](#_heading=h.nfy8mkhlluzj) 17

[**6. REFERENCES**](#_heading=h.wu91c7yvas2x) **17**

[**7. APPENDIX**](#_heading=h.ln9hmyhzy5py) **17**

1. **INTRODUCTION**

The main objective of the chat room project is to create a chat application which helps different clients or users to communicate with each other through a server connected.The main objective of the Simple Chat Room project is to create a chat application which helps different users to communicate with each other through a server connected. This is a simple chat program with a server and can have many clients.

The server needs to be started first and clients can be connected later. Simple Chat Room provides a bidirectional communication between server and clients. It enables users to seamlessly communicate with each other. The user has an option to login to the chat room. The user can chat using this chat application. The server will display the user names who are active in the chat room and make those users visible. If the user at the other end is active then they can start a chat session.

**1.1 Intended Audience**

|  |  |
| --- | --- |
| BU Authority |  |
|  |  |

**1.2 Acronyms/Abbreviations**

|  |  |
| --- | --- |
| TCP | Transmission Control Protocol |
| IP | Internet Protocol |
| IPv4 | Internet Protocol version 4 |
| API | Application Programming Interface |
| VM | Virtual Machine |

**1.3 Project Purpose**

The main purpose of this project is to communicate with multiple clients

with a server.

**1.4 Key Project Objectives**

The objective is to create a communication between multiple clients and a server.

**1.5 Project Scope and Limitation**

**Scope:**

The scope of this project includes that communication can be formed in

between client to client and all in server .

**Limitation:**

Multiple clients can connect to only one server.

**1.5.1 In Scope**

* Clients/servers should provide a valid file name.
* When the client is sending,the server should respond to read the file.

**1.5.2 Out Scope**

* Client/server can transfer different files

**1.6 Functional Overview**

* **Client:**

In the client terminal the client will login. After successful login, the client will communicate to the other clients.

* **Server:**

In the server terminal the server receives the data from multiple clients.

**1.7 Assumptions, Dependencies & Constraints**

OPERATING SYSTEMS:

Operating environment systems are:

* Client/server system
* Operating system: Linux
* Platform: Intel X86

**1.8 Risks**

No Risks.

**2. DESIGN OVERVIEW**

1. Start

This is the start block which indicates the start of the program,where socket is created and consists of server and multiple clients.where server accepts the clients credentials like username.Then the server will allow the client and give access for further communication.

1. Client Login

This is the module used for client login where the client can connect to the server by their username and then join to the server to communicate between multiple clients.

1. Login Credentials

In this module the credentials entered by the client are then validated by the system. If the client enters a valid user name then it will move to the further step.

1. Enter your domain name

Once the server has validated the credentials, it will now be connected to the server’s port number. Once connected to the server’s socket the client will access.

1. Catching the domain name

Once the server validates the credentials, the server socket will be created and will be binded to the server’s port no and now the server will remain listening on the port waiting for any client connection. Once a connection is established between the client, the server will catch the domain name entered by the client in this module.

1. Server provides respective IP address

When a domain name corresponding to the client entered domain name is found, the server will give the respective IP address to the client.

1. End

This ensures that the program has terminated.

**2.1 Design Objectives**

Create login credential page for both server and client

Take domain name as an input from the client after successful login

Server will check whether domain names exist or not.

If exists, server will send IP address to client

**2.1.1 Recommended Architecture**

Socket

**2.2 Architectural Strategies**

* Header files
* Structures
* Macros

**2.2.1 Design Alternative**

NA

**2.2.2 Reuse of Existing Common Services/Utilities**

NA

**2.2.3 Creation of New Common Services/Utilities**

NA

**2.2.4 User Interface Paradigms**

Command Line Interface: Terminal

**2.2.5 System Interface Paradigms**

Command Line Interface: Terminal

**2.2.6 Error Detection / Exceptional Handling**

**Error detection:**

1. Invalid Domain name.
2. IP address does not exist.
3. Errors will be handled by perror.
4. If the socket is not created.

**2.2.7 Memory Management**

NA

**2.2.8 Performance**

NA

**2.2.9 Security**

For security purposes the system asks for login credentials from server and

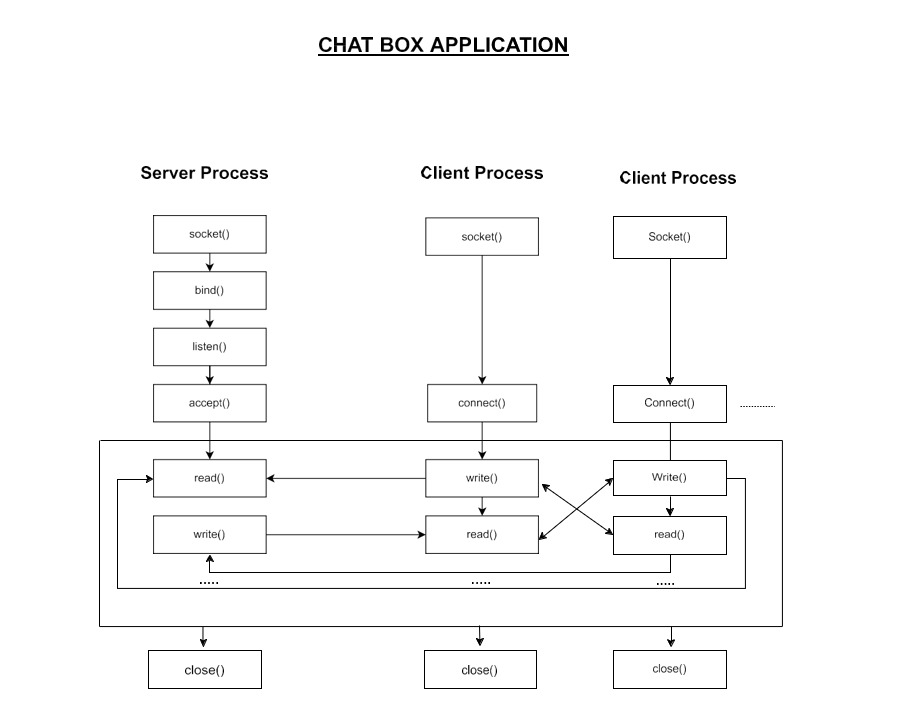
client.

**2.2.10 Concurrency and Synchronization**

NA

**3. SYSTEM ARCHITECTURE**

**3.1 System Architecture Diagram**



**3.2 System Use-Cases**

**

**4. DETAILED SYSTEM DESIGN**

The code starts by declaring the struct sockaddr\_in and hostent. After that client socket will be created. Using connect() the client establishes connection with the server. After a successful connection client will connect to the server.

The message will be sent to the server.

The server will first do binding after that it will be listening on a particular port which will be the same for server and client.

Accept() will be called which will accept the connection from the client. The received domain name will be accepted by the server and will search for the equivalent domain name from the connected Clients.

After fetching the IP address server will send the IP address to the client.

**4.1 Key Entities**

* Valid domain/host name
* IP Address

**4.2 Detailed-Level Database Design**

NA

**4.2.1 Data Mapping Information**

Mapping the IP address from the server side is done.

**4.2.2 Data Conversion**

Converting the IP address(IPv4 format) from binary to standard text format.

**4.3 Archival and retention requirements**

NA

**4.4 Disaster and Failure Recovery**

* We don’t have any control over the system. In case of failure, source code is safe.
* Use of Git.

**4.5 Business Process workflow**

NA

**4.6 Business Process Modeling and Management**

(as applicable)

NA

**4.7 Business Logic**

NA

**4.8 Variables**

NA

**4.9 Activity / Class Diagrams**

**Pseudocode for Server Side:**

Create a server socket

Bind socket to specific port where client will connect with the server

Listen for connections on the socket

Loop

Accept new connection

Read and Write data to communicate with the Clients

Client Connect with Server

Close AcceptRetStatus

End Loop

Close

**Pseudocode for Client Side:**

Create a client socket

Establish connection with server by calling connect()

Read and Write data in ConnectRetStatus to communicate with server

Close

**4.10 Data Migration**

NA

**4.10.1 Architectural Representation**

NA

**4.10.2 Architectural Goals and Constraints**

The project is just for communication purposes.

**4.10.3 Logical View**

NA

**4.10.4 Architecturally Significant Design Packages**

NA

**4.10.5 Data model**

NA

Legacy system data model

Proposed system data model

Interface data model

**4.10.6 Deployment View**

NA

**5. ENVIRONMENT DESCRIPTION**

**GCC**: In Linux, the GCC stands for GNU Compiler Collection. It is a compiler system for the various programming languages. It is mainly used to compile the C and C++ programs.

Socket Programming: Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

**UBUNTU:** Ubuntu is an open-source operating system (OS) based on the Debian GNU/Linux distribution. Ubuntu incorporates all the features of a Unix OS with an added customizable GUI, which makes it popular in universities and research organizations. Ubuntu is primarily designed to be used on personal computers, although a server edition does also exist.

**GITHUB:** GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere. This tutorial teaches you GitHub essentials like repositories, branches, commits, and pull requests.

**5.1 Time Zone Support**

NA

**5.2 Language Support**

English

**5.3 User Desktop Requirements**

Linux, Ubuntu

**5.4 Server-Side Requirements**

None

**5.4.1 Deployment Considerations**

NA

**5.4.2 Application Server Disk Space**

NA

**5.4.3 Database Server Disk Space**

NA

**5.4.4 Integration Requirements**

NA

**5.4.5 Jobs**

NA

**5.4.6 Network**

Ethernet network.

**5.4.7 Others**

NA

**5.5 Configuration**

NA

**5.5.1 Operating System**

Linux desktop editions with 8 GB RAM- A GUI-based LINUX system must be used

**5.5.2 Database**

NA

**5.5.3 Network**

Ethernet network

**5.5.4 Desktop**

* CPU : Intel i3/i5/i7 generation 3 and later
* RAM: 4GB or greater - For optimal performance, 6GB or 8GB are recommended if you will be running multiple browser tabs and/or multiple applications at the same time
* Internal memory:476 GB SSD/HDD.

**6. REFERENCES**

**7. APPENDIX**

Change Log

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **QMS Template Version Control (Maintained by QA)** | | | | | |
|  |  |  |  |  |  |
| **Date** | **Version** | **Author** | | **Description** | |
| 16/11/22 | Version 1.0 | Group 7 | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |