**CALL FORWARDING SYSTEM SIMULATOR**

**CFSS\_DesignDocument-v0.2**

**Document Control:**

| **Project Revision History** | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | |  |  |  |  |  |
| **Date** | **Version** | **Author** | **Brief Description of Changes** | | | | **Approver Signature** | |
| 09/30/2022 | 0.1 | Prateek, Alok, Abhishek, Rahul, Jinse | Detailed Design of CFSS | | | |  | |
| 10/11/2022 | 0.2 | Alok, Jinse | Changes in diagrams and changes under database header in 5.5.2 | | | |  | |
| 10/13/2022 | 0.2 | Prateek | Corrected Indentation, Activity Class Diagram | | | |  | |

| **Team Members** |
| --- |

| **Employee ID:** | **Name** |
| --- | --- |
| 46264036 | Prateek Sharma |
| 46264677 | Abhishek Gupta |
| 46264042 | Alok Kumar |
| 46264105 | Jinse Thomas |
| 46264081 | Rahul Dey |

[**1. Introduction 6**](#_heading=h.3znysh7)

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

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

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

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

[1.5. Project Scope and Limitation 7](#_heading=h.4d34og8)

[1.5.1. In Scope 7](#_heading=h.2s8eyo1)

[1.5.2. Out of scope 7](#_heading=h.17dp8vu)

[1.6. Functional Overview 7](#_heading=h.3rdcrjn)

[1.7. Assumptions, Dependencies & Constraints 7](#_heading=h.26in1rg)

[1.8. Risks](#_heading=h.lnxbz9) 8

[**2. Design Overview**](#_heading=h.35nkun2) **8**

[2.1. Design Objectives 8](#_heading=h.1ksv4uv)

[2.1.1. Recommended Architecture 8](#_heading=h.44sinio)

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

[2.2.1. Design Alternative 9](#_heading=h.3j2qqm3)

[2.2.2. Reuse of Existing Common Services/Utilities 9](#_heading=h.1y810tw)

[2.2.3. Creation of New Common Services/Utilities 9](#_heading=h.4i7ojhp)

[2.2.4. User Interface Paradigms](#_heading=h.2xcytpi) 9

[2.2.5. System Interface Paradigms 9](#_heading=h.1ci93xb)

[2.2.6. Error Detection / Exceptional Handling](#_heading=h.3whwml4) 10

[2.2.7. Memory Management](#_heading=h.2bn6wsx) 10

[2.2.8. Performance 1](#_heading=h.qsh70q)0

[2.2.9. Security 1](#_heading=h.3as4poj)0

[2.2.10. Concurrency and Synchronization 1](#_heading=h.49x2ik5)0

[2.2.11. Housekeeping and Maintenance 1](#_heading=h.2p2csry)0

[**3. System Architecture**](#_heading=h.147n2zr) **11**

[3.1. System Architecture Diagram. (Not Necessary) 1](#_heading=h.23ckvvd)1

[3.2. System Use-Cases 1](#_heading=h.ihv636)1

[3.3. Subsystem Architecture](#_heading=h.32hioqz) 12

[3.4. System Interfaces](#_heading=h.1hmsyys) 13

[3.4.1. Internal Interfaces](#_heading=h.41mghml) 13

[3.4.2. External Interfaces 1](#_heading=h.2grqrue)3

[**4. Detailed System Design**](#_heading=h.vx1227) **14**

[4.1. Key Entities](#_heading=h.3fwokq0) 14

[4.2. Detailed-Level Database Design](#_heading=h.4f1mdlm) 15

[4.2.1. Data Mapping Information 1](#_heading=h.2u6wntf)8

[4.2.2. Data Conversion 1](#_heading=h.19c6y18)8

[4.3. Archival and retention requirements 1](#_heading=h.xvir7l)9

[4.4. Disaster and Failure Recovery](#_heading=h.3hv69ve) 20

[4.5. Business Process workflow](#_heading=h.28h4qwu) 21

[4.6. Business Process Modeling and Management (as applicable)](#_heading=h.nmf14n) 22

[4.7. Business Logic](#_heading=h.37m2jsg) 23

[4.8. Variables](#_heading=h.1mrcu09) 25

[4.9. Activity / Class Diagrams (as applicable)](#_heading=h.2lwamvv) 26

[**5. Environment Description**](#_heading=h.3l18frh) 30

[5.1. Time Zone Support](#_heading=h.206ipza) 30

[5.2. Language Support](#_heading=h.4k668n3) 30

[5.3. User Desktop Requirements](#_heading=h.2zbgiuw) 30

[5.4. Server-Side Requirements](#_heading=h.1egqt2p) 30

[5.4.1. Deployment Considerations](#_heading=h.3ygebqi) 30

[5.4.2. Application Server Disk Space](#_heading=h.2dlolyb) 31

[5.4.3. Database Server Disk Space](#_heading=h.sqyw64) 31

[5.4.4. Integration Requirements](#_heading=h.1rvwp1q) 31

[5.4.5. Jobs](#_heading=h.4bvk7pj) 31

[5.4.6. Network](#_heading=h.2r0uhxc) 31

[5.4.7. Others](#_heading=h.1664s55) 31

[5.5. Configuration](#_heading=h.3q5sasy) 31

[5.5.1. Operating System](#_heading=h.25b2l0r) 31

[5.5.2. Database](#_heading=h.kgcv8k) 32

[5.5.3. Network](#_heading=h.34g0dwd) 32

[5.5.4. Desktop](#_heading=h.1jlao46) 32

[**6. References**](#_heading=h.43ky6rz) 32

[**7. Appendix**](#_heading=h.2iq8gzs) 32

# 1. INTRODUCTION

The Call forwarding System Simulator should be able to make a call from one client to another client and depending on the status of the service selected by the client the respective call forwarding takes place

## Intended Audience

This project is a prototype for everyone who wants to make a call and connect or forward the call depending on the status selected by the client. This document is intended to be read by developers, testers, project managers and customers. This is a technical document, and the terms should be understood by all of them.

## Acronyms/Abbreviations

| CFSS | Call forwarding system simulator |
| --- | --- |
| TCP | Transmission Control Protocol |

## Project Purpose

Purpose of this project is to provide a call forwarding system to the client. It aims to make a call from one client to the other and depending on the service selected by the receiver the call forwarding will take place.

## Key Project Objectives

* Establishing connection between client and client via server
* Giving option to client to choose from the types of call forwarding service.
* To forward the call to other client

## 1.5 Project Scope and Limitation

The scope and limitation of the project are listed,

### 1.5.1 In Scope

In this project the call forwarding simulator will allow the client to connect to the

server and enable the call forwarding services after registering so that when another

client calls the call is forwarded depending upon the 3 types of call forwarding

services selected. If the client is authenticated from the database in server and the

service is enabled the call gets forwarded, in other cases a normal call is placed and

all calls are client requests and forward calls are logged with timestamp. Admin has

the right to add/delete/update the entries of the database.

### 1.5.2 Out of scope

NA

## 1.6 Functional Overview

In this service, we will have servers and multiple clients. The call forwarding service

is provided by the server.

First, the client will request for a connection which will be validated, and if the client

has enabled call forwarding service, then it establishes connection between the

requesting client and intended forwarding client. In the service we provide three types

of call forwarding service unconditional, no reply and busy. Our server will be

maintaining a database for call forwarding details. Server will allow the client to

register or unregister for a call initially before enabling or disabling the service.

On a call connected from a client, the server shall check for activation of forwarding

service using its database and if service is activated, then shall forward accordingly

depending on the call forwarding type.

Once unregistered, no forwarding service shall be provided. Clients can unregister

with servers if services are no longer required. If the service has been deactivated by

the client the call received will be normal else all the incoming calls will be handled as

per the forwarding type.

CFSS should work on Linux as well as on Windows.

## 1.7 Assumptions, Dependencies & Constraints

The connection is established between server and the client.

## 1.8 Risks

NA

# DESIGN OVERVIEW

The system consists of three entities, the sender, the receiver and the server.

## Design Objectives

The call forwarding simulator will allow the client to connect to the server and enable the call forwarding services after registering so that when another client calls the call is forwarded depending upon the 3 types of call forwarding services selected.

* Unconditional
* No reply
* Busy

The receiver can enable or disable the service whenever they want.

### Recommended Architecture

The recommended system architecture is as follows.

Server-side architecture:

* 1GB RAM
* 500MHz Processor
* 120GB HDD CPU
* Terminal

Client-side hardware interface:

* Desktop or Linux machine
* Terminal

## Architectural Strategies

The architectural strategy used in this project is a client-server strategy. This strategy consists of two entities, the client, and the server. The client requests services from the server and the server provides services to clients. The server continues to listen to the clients’ requests until the client wants to discontinue with the service or the connection is not closed.

### 2.2.1 Design Alternative

The project uses the TCP protocol to establish a connection between the server and client. An alternative to TCP is the UDP protocol. TCP stands for Transmission Control Protocol, a communications standard that enables application programs and computing devices to exchange messages over a network. We have used TCP because TCP is a connection-oriented protocol, it is more reliable as it provides error checking support and guarantees delivery of data to the destination router. .

### 2.2.2 Reuse of Existing Common Services/Utilities

The system requires the use of existing common services and utilities which are: TCP (Transmission Control Protocol) & other System Calls.

### 2.2.3 Creation of New Common Services/Utilities

The project does not create or use any new common services or utilities.

### 2.2.4 User Interface Paradigms

* Command Line Interface (CLI).

### 2.2.5 System Interface Paradigms

* Operating system – Linux/windows.

### 2.2.6 Error Detection / Exceptional Handling

* Error detection in all phases of client connection to the server will be provided.
* Four levels of debug log messages will be included like ERROR, INFO, WARNING & DEBUG.
* Appropriate error messages for file handling will also be included.

### 2.2.7 Memory Management

NA.

### 2.2.8 Performance

Taking the essential speediness into consideration we will be using TCP designed as TCP is more reliable. Performance of TCP allows the server and client to deliver the best throughput and latency for an individual connection.

**2.2.9 Security**

We have made use of getpass() function to secure the entered password which hides the entered password

### 2.2.10 Concurrency and Synchronization

Client and server will be in synchronization in receiving and sending the messages.

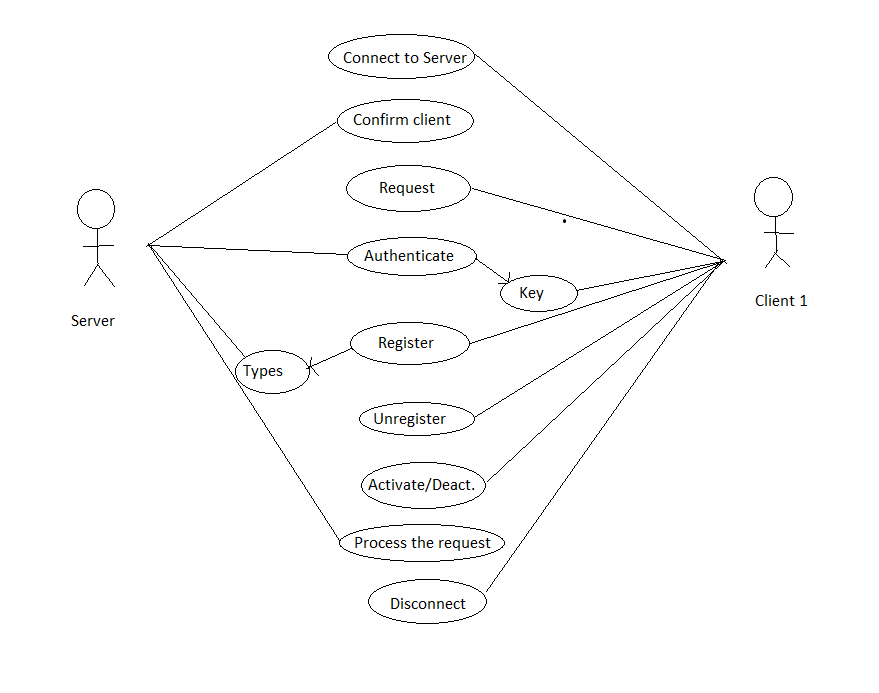
### 2.2.11 Housekeeping and Maintenance

* Clearing the memory buffers from the system.
* Flushing the contents of the screen when the client starts for better User Experience.

# SYSTEM ARCHITECTURE

Call forwarding system services employ specialized data structures that are optimized for receiving the message and forward it as per our requirement. Because it ensures the successful delivery of data and messages over networks.

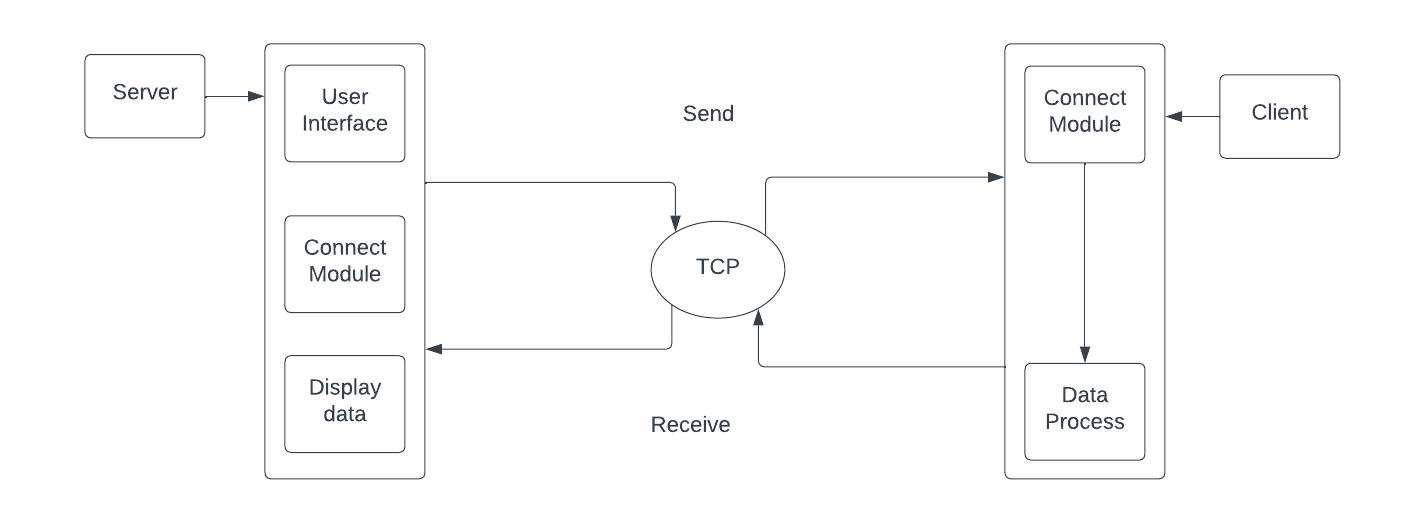
## System Use Case Diagram



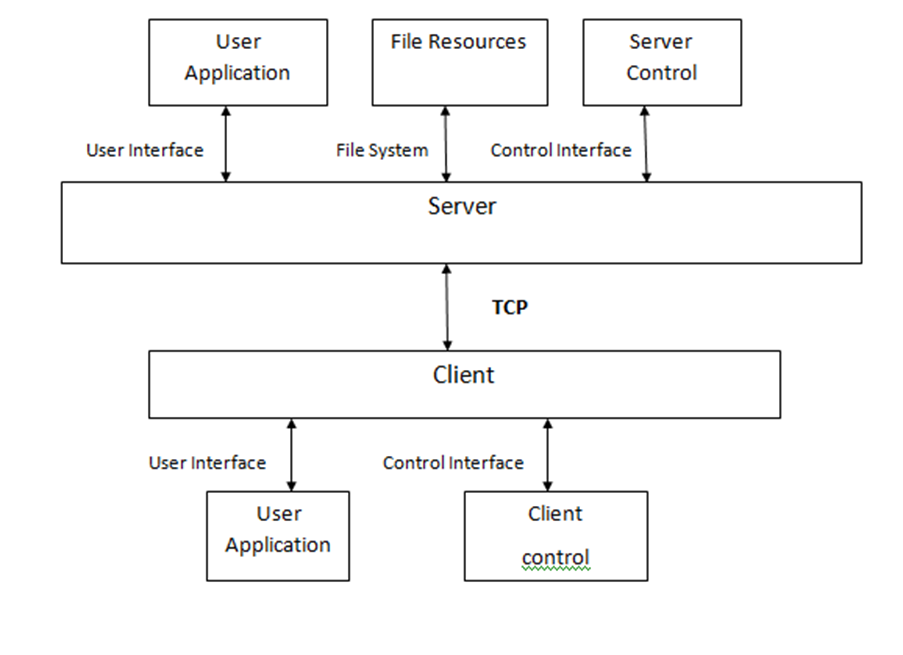
## System Use-Cases

* The request to make a connection to the receiver will be done.
* Once the connection is successful, then displaying “Your call has been forwarded” on successful call forwarding, then the server will check if the receiver has enabled any forwarding service or not.
* If yes, then depending on the type of forwarding service enabled the message will be forwarded. If not then we are directly sending the message to the receiver and storing it in logs with date and time stamps.
* If the client unregisters then disable all the services for the client.

## Subsystem Architecture



## System Interfaces



### 3.4.1 Internal Interfaces

The internal interfaces comprise interfaces through which the system interacts with the clients through which it provides them services.

### 3.4.2 External Interfaces

The external interface comprises interfaces through which the users interact with the system.

* Sender’s desktop or Linux Machine
* Receiver desktop or Linux machine.

# DETAILED SYSTEM DESIGN

The Call Forwarding System Simulator (CFSS) contains server and multiple client connections. The connection between the server and clients is established using TCP protocol.

The client can: -

1. Register/ Unregister to the server database
2. Login to the server to Enable/ Disable Service
3. Forward call if in the busy, unconditional, or no-reply state

If the Client is registered in the server and service is activated, the call should be forwarded to the designation.

## Key Entities

The key entities associated with the system are:

Server

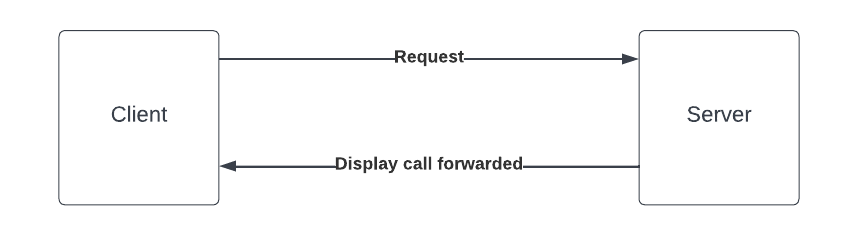
* The server is a remote entity that stores the file which contains a database of existing users.
* It verifies the IP address of the client making the connection.
* Provides service on client requests.
* It verifies the unique number of the receiver and the type of forwarding service enabled.
* Provides service on the client request.
* Client requests the server to register it to the database, enable/ disable the service, or forward the call to desired destination
* Logs are maintained as per user request.

Client

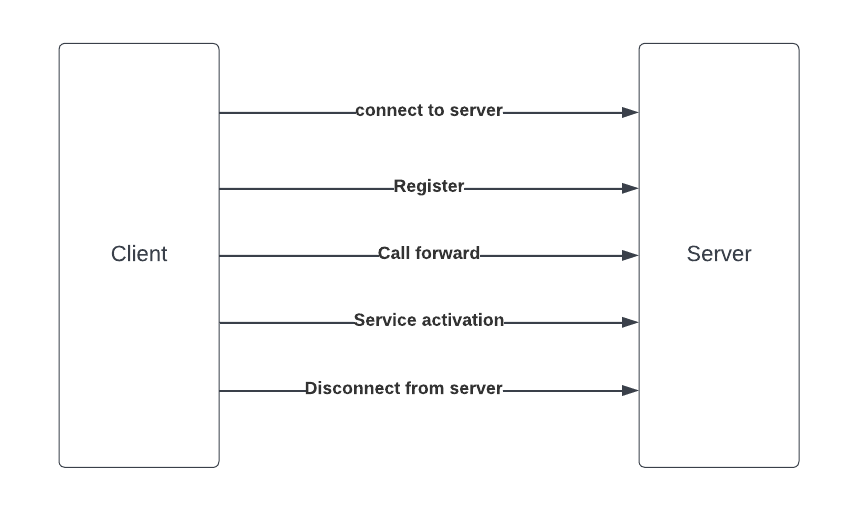
* The client is an entity that is run by the user on their system.
* It requests to connect with the server.
* Client is calling another client and depending on the choice of the call forwarding system the call will be forwarded.
* Client will exit after all the operations.

## Detailed-Level Database Design

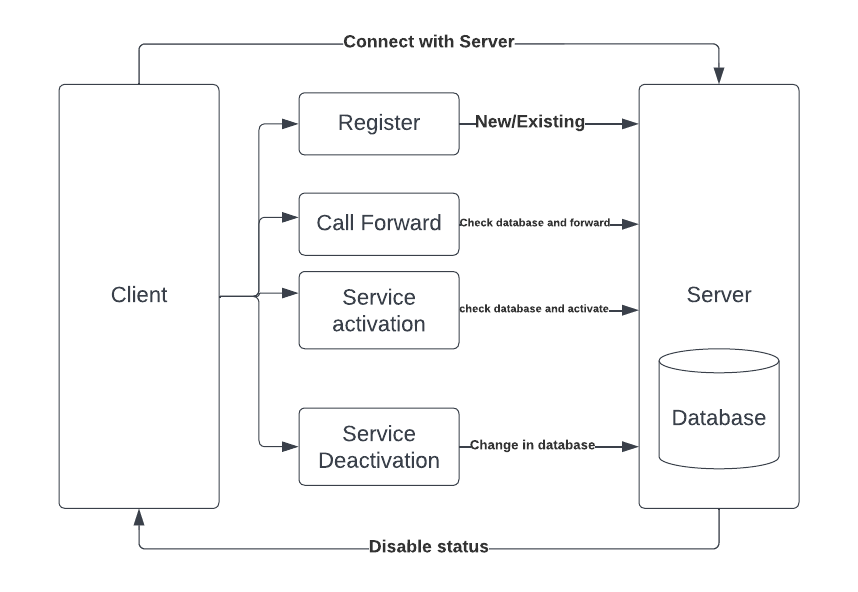
**4.2.1 Level - 0 Diagram**

****

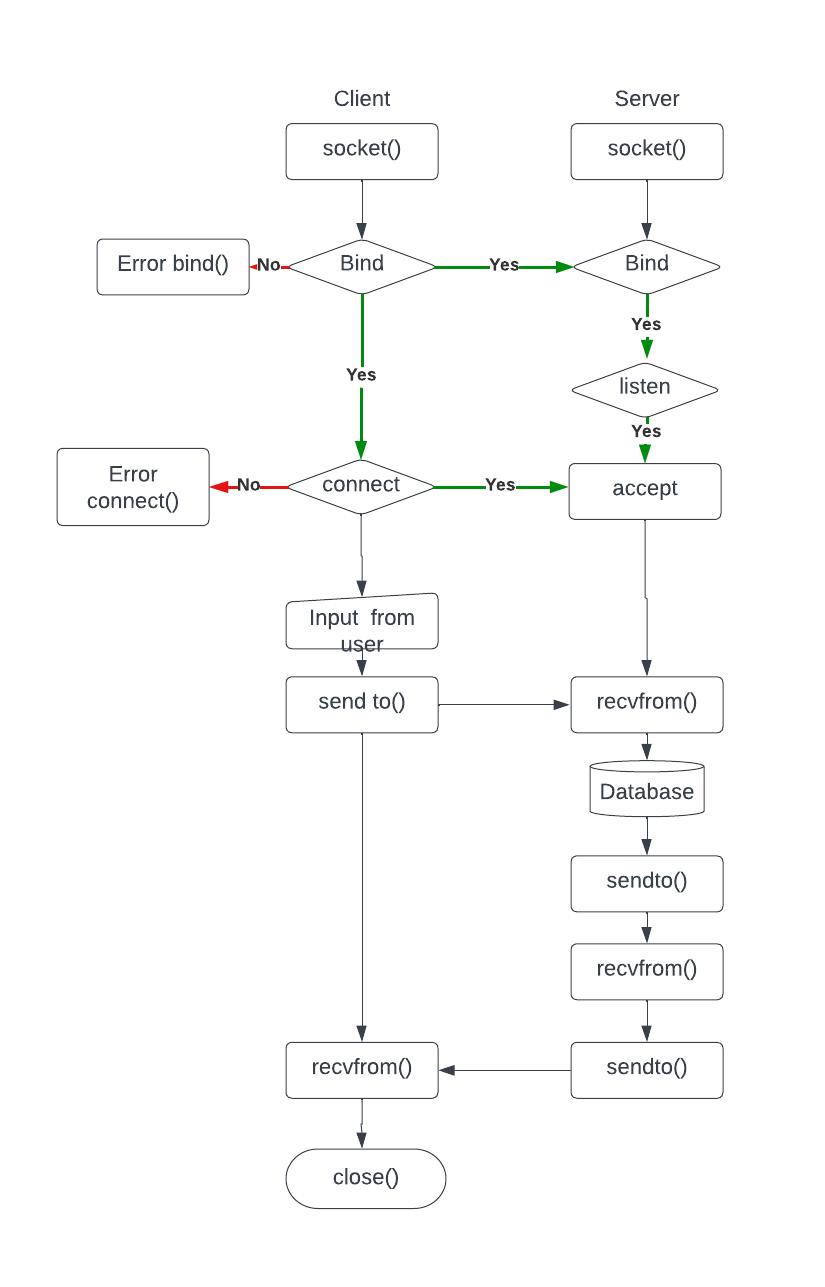
**4.2.2 Level - 1 Diagram**

****

**4.2.3 Level - 2 Diagram**

****

### 4.2.4 Data Mapping Information



### 4.2.5 Data Conversion

NA.

**4.2.6 Requirement Specifications**

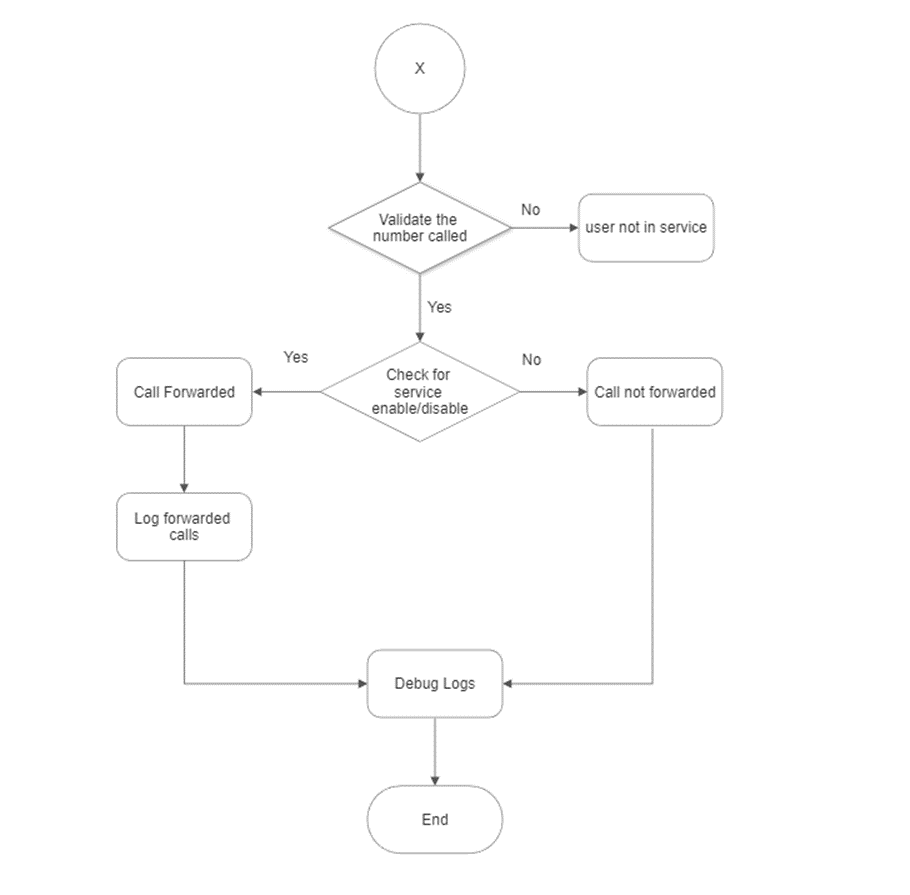
| Functional Requirements | Requirement Description | Priority |
| --- | --- | --- |
| CFSS\_SR\_01 | Server should maintain a database for the call forwarding details which include user authentication, type, source and destination etc. | Mandatory |
| CFSS\_SR\_02 | Server shall allow clients to register or unregister for call forwarding service initially before enabling or disabling the service.  Clients register/unregister requests shall be authenticated using a database. | Mandatory |
| CFSS\_SR\_03 | Register/Unregister request shall be encrypted | Optional |
| CFSS\_SR\_04 | The Server should support 3 types of call forwarding services i.e. unconditional or No reply or as Busy. | Mandatory |
| CFSS\_SR\_05 | Shall allow authenticated and registered users to activate or deactivate the service as per their requirement | Mandatory |
| CFSS\_SR\_06 | Activation and Deactivation of Call forwarding request shall require encrypted authentication | Optional |
| CFSS\_SR\_07 | On a call connected from a client, the server shall check for activation of forwarding service using its database and if service is activated, then shall forward accordingly depending on the call forwarding type. | Mandatory |
| CFSS\_SR\_08 | Once unregistered, no forwarding service shall be provided | Mandatory |
| CFSS\_SR\_09 | Admin shall allow add/delete/update of database entries | Mandatory |
| CFSS\_SR\_10 | Shall log all client requests and forwarded calls with date timestamp. | Mandatory |
| CFSS\_CL\_01 | Shall register with server to request for call forwarding service | Mandatory |
| CFSS\_CL\_02 | Registered and Authenticated Client can enable or disable the Call forwarding service anytime.  No restriction on the number of enable/disable of Call forwarding service | Mandatory |
| CFSS\_CL\_03 | Shall unregister with server if services are no longer required | Mandatory |
| CFSS\_CL\_04 | Shall receive normal calls addressed to the client if service is deactivated else all incoming call to be handled as per call forwarding service type | Mandatory |
| CFSS\_CL\_05 | Shall unregister with server | Mandatory |
| CFSS\_CL\_06 | Should include debug log messages with at least 4 levels (FATA, INFO, WARNING, DEBUG) | Mandatory |
| Non-Functional Requirements |  |  |
| CFSS\_N\_01 | The application should run (after compilation) on LINUX as well as windows | Desirable |

**4.3 Disaster and Failure Recovery**

NA.

.

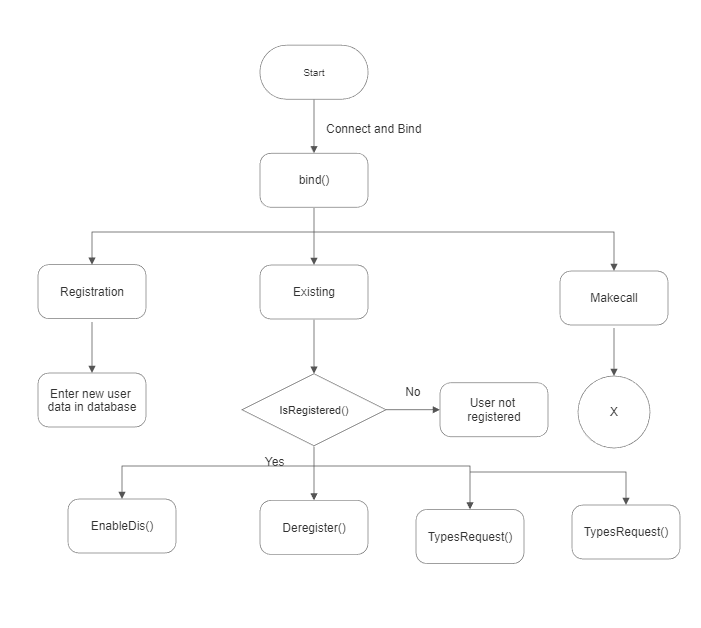
## 4.4. Business Process workflow

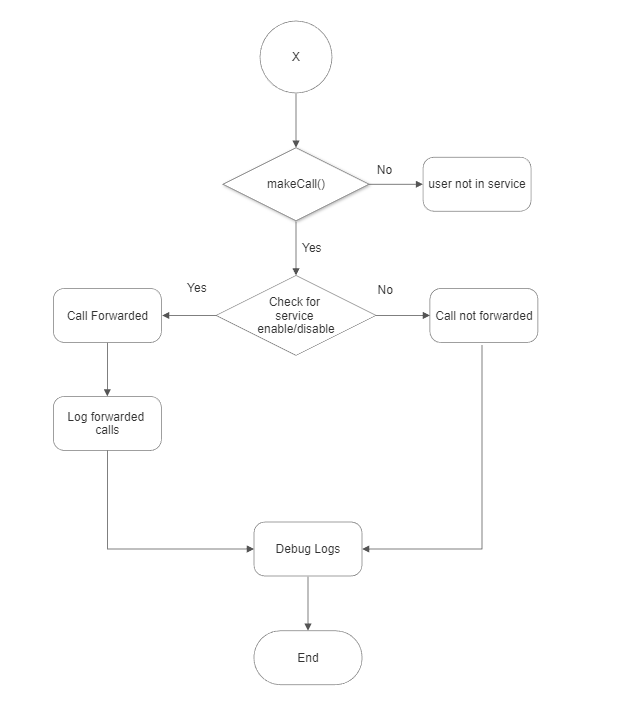


## 4.5. Business Process Modeling and Management (as applicable)

NA

## 4.6 Business Logic





## 4.7. Variables

int portNo

int sfd

int csfd

struct sockaddr\_in server\_addr

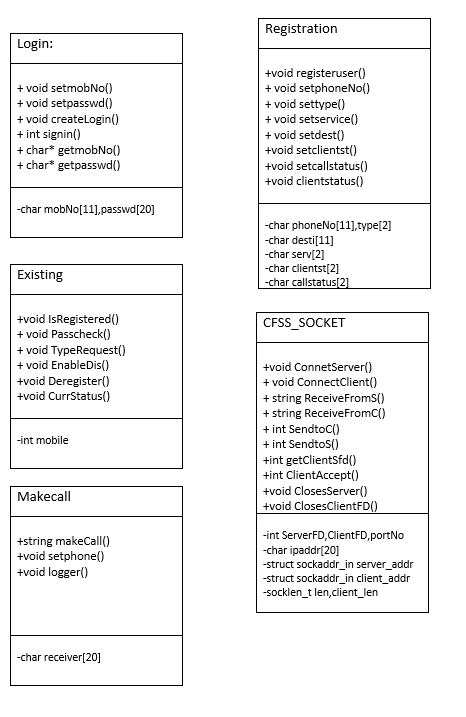
struct sockaddr\_in client\_addr

int choice

string mobileNo

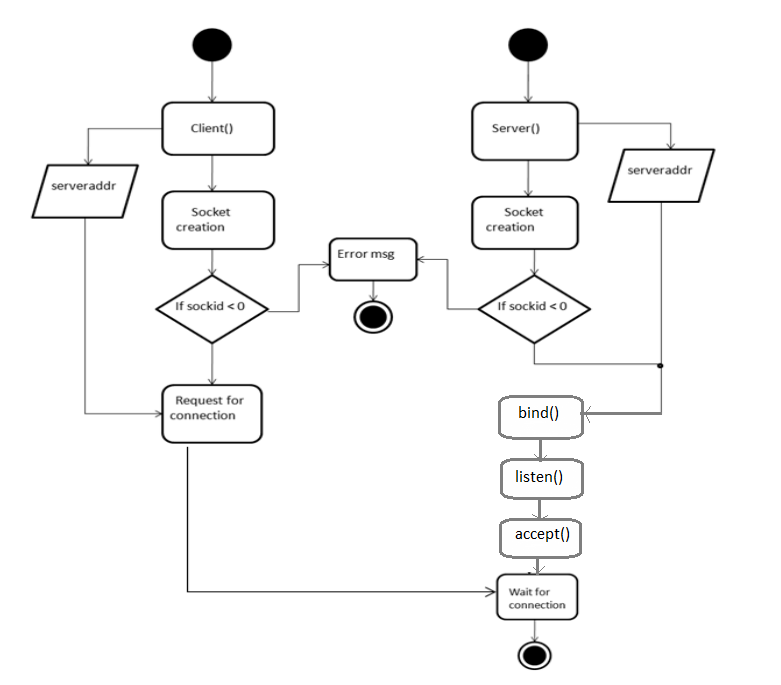
string Filename

## 4.8. Activity/ Class Diagrams (as applicable)

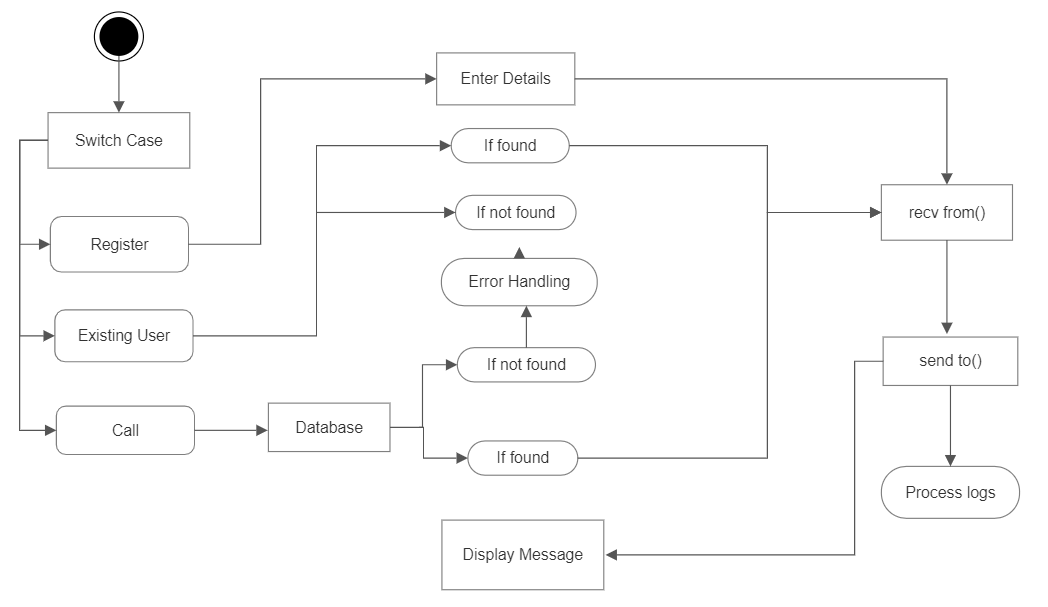


#### 4.8.2 Activity Diagram

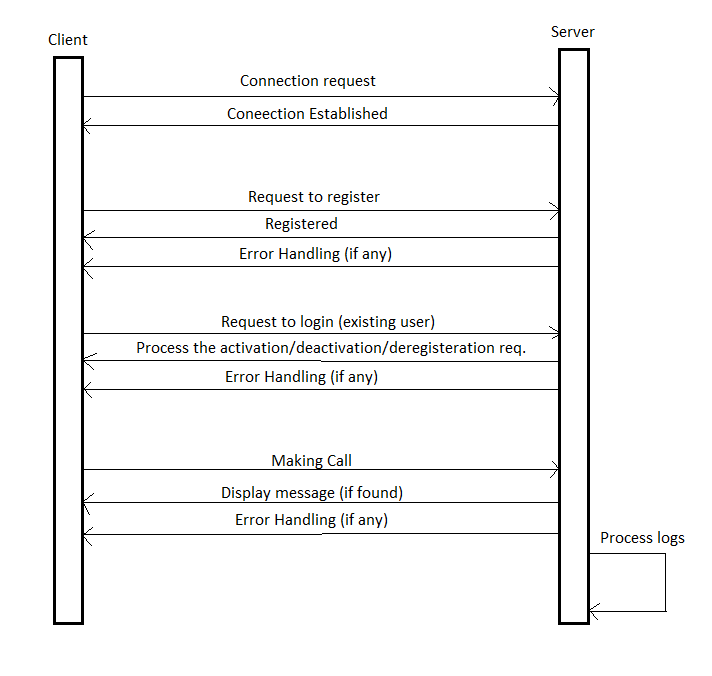
#### 4.8.2.1 Socket creation diagram:



#### 4.8.2.2 Socket end operation diagram:



#### 4.8.3. Sequence diagram:



## 4.9. Data Migration

Data is migrated between the client and the server.

# ENVIRONMENT DESCRIPTION

The Environment description allows the client to connect to the server and then the server will check the status of the receiver and do call forwarding accordingly if the service has been enabled by the receiver.

## Time Zone Support

It will support time zones as per Indian standard time (IST) in (GMT +5:30) and UST standard.

## Language Support

C++ language and compilation using g++. The Linux commands helps us to do that task we can specify the commands.

## User Desktop Requirements

User desktop requires a Linux environment or windows environment.

## Server-Side Requirements

On the server side,

● Disk space – Minimum 150GB

● Monitor long-running jobs, to reduce the server load.

### 5.4.1Deployment Considerations

Deployment considerations are

● 500Mhz Processor

● 120GB HDD CPU

● 4GB RAM

### 5.4.2 Application Server Disk Space

Minimum 150GB.

### 5.4.3 Database Server Disk Space

NA.

### 5.4.4 Integration Requirements

The pwd linux command displays the current working directory on the server for the logged-in user.

### 5.4.5 Jobs

NA.

### 5.4.6 Network

NA

### 5.4.7 Others

NA.

## 5.5 Configuration

Call forwarding system simulator works in a client-server model. The server provides the call forwarding system for the client, using system calls and TCP protocol for the transfer of data. Call forwarding system simulators shall work in all standard Operating systems. The client can enable the call forwarding servo

### 5.5.1 Operating System

* Operating system – Unix.
* RAM - 8GB.
* Processor - i3/i5/i7.

### 5.5.2 Database

Fat file system is used.

### 5.5.3 Network

The following are the network details regarding the project:

* The client and server communicate over a TCP/IP protocol.
* The IP address used can be either IPv4 or IPv6.

### 5.5.4 Desktop

A Unix-like environment is required.

# 6. REFERENCES

[https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading](https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading/) To handle multi-client without multi threading

System Requirements Specification Document

[https://www.geeksforgeeks.org/vector-in-cpp-stl](https://www.geeksforgeeks.org/vector-in-cpp-stl/) Vector in STL

# 7. APPENDIX

NA.

**Change Log**

| **QMS Template Version Control (Maintained by QA)** | | | | | |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| **Date** | **Version** | **Author** | | **Description** | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |
|  |  |  | |  | |