Command Line Chat Application

Grzegorz Sosnowski

Szczecin, 17.09.16

Table of Contents

1. Introduction 3

1.1 Purpose 3

1.2 Scope 3

1.3 Definitions 3

2. Definition of requirements 4

2.1 Client functional requirements 4

2.2 Server functional requirements 5

3. Use case diagram 5

4. Client request protocol 6

4.1 Introduction 6

4.2 Protocol design 6

1. **INTRODUCTION**

**1.1** **PURPOSE**

This document is starting point for design of “Command line chat application”. Given document contains design, work diagrams but its main purpose is to present the process of creating application.

**1.2** **SCOPE**

This project can be divided in two main modules:

* Client application
* Server

Which means that that project is based on Client/Server architecture, client sends request to the server and the server responses by handling given instruction and sending proper answer. This brings us to main functionalities of chat server. In order to be used server has to occupy port which also serves as its address. First and foremost server has to establish and uphold connection with client.

**1.3 DEFINITIONS**

The following is a list of definitions for this document:

|  |  |
| --- | --- |
| Server | Server based on TCP connection |
| User | Client connected to Server |
| Rx.y | Requirement ID, x and y stand for module and specific requirement identification |
|  |  |
|  |  |

**2.** **DEFINITION OF REQUIREMENTS**

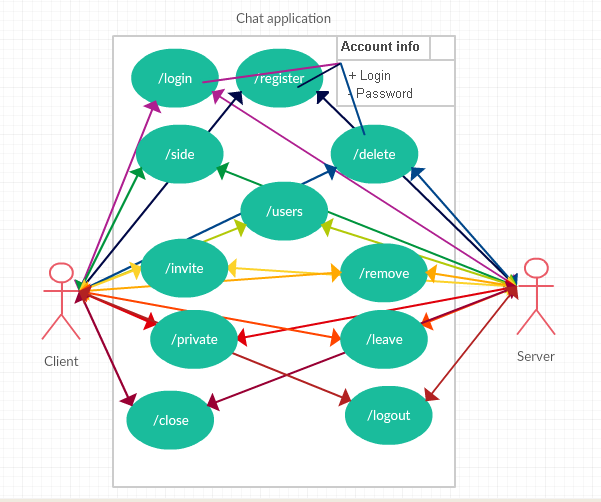
**2.1** **CLIENT FUNCTIONAL REQUIREMENTS**

|  |  |
| --- | --- |
| **Requirement ID** | **Definition of requirement** |
| **R1.1** | User connects to server via TCP connection |
| **R1.2** | User can register new account using login and password |
| **R1.3** | User can remove existing account using login and password |
| **R1.4** | User can log into system by using his login and password |
| **R1.5** | After logging user joins main conversation channel |
| **R1.6** | After joining main channel user can send messages to side channels by using built-in commands |
| **R1.7** | User can display list of connected users by using specific command |
| **R1.8** | User can create side channel by using specific command. User receives channel ID from Server   |  |  | | --- | --- | | **R1.8a** | User can send messages to other participants | | **R1.8b** | User can invite(add) users to his channel | | **R1.8c** | User can remove users from his channel | | **R1.8d** | User can delete his channel | |
| **R1.9** | User can join side channel when invited by side channel owner   |  |  | | --- | --- | | **R1.9a** | Send messages in this channel | | **R1.9b** | Invite(Add) other users to this channel | | **R1.9c** | Leave channel(joins main conversation channel) | |
| **R1.10** | User can log out from system. This action terminates connection to Server |

**2.2** **SERVER FUNCTIONAL REQUIREMENTS**

|  |  |
| --- | --- |
| **Requirement ID** | **Definition of requirement** |
| **R2.1** | Server starts after determining port which it will occupy |
| **R2.2** | Server can add users to system after receiving specific command |
| **R2.3** | Server can remove users from system after receiving specific command |
| **R2.4** | Server can register side conversation channel after receiving specific command. Servers sends ID of new channel to user which requested it |
| **R2.5** | Server receives messages and sends them proper responses |

**3.** **USE CASE DIAGRAM**



**4.** **CLIENT REQUEST PROTOCOL**

**4.1. INTRODUCTION**

In order to handle specific command from user, system must be designed in way that allows recognition of certain keywords and then process user's input in a proper manner. Table below lists those keywords, their definitions and possible server response.

**4.2. PROTOCOL DESIGN**

|  |  |  |  |
| --- | --- | --- | --- |
| Command | Data to process | Definition | Server Response |
| /login | login<string> password<string> | User inputs his login and password assigned to it. In case when given data is correct user acquires flag “Logged” which allows using all functionalities of system. | Logged(“”)  ERROR(“USER DOESN’T EXIST; WRONG PASSWORD”)  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /register | login<string>  password<string> | User sends his login and password which are saved and then server logs user in | Logged(“”)  ERROR(“USER EXISTS”)  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /delete | login<string>  password<string> | User sends his login and password which are removed from the system then communication with closes | Loggedout(“”)  ERROR(“USER DOESN’T EXIST; WRONG PASSWORD”)  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /chat | message<string> | User sends message to all other users connected and logged into system | None  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /side | none | Server creates private channel, generate ID, send it to user and grant flag that allow to operate it | Created(ID)  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /users | none | Displays list of logged users with their ID’s | DIsplayed(ID)  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /invite | ID<int>  CID<int> | User sends user ID and channel ID to server in order to invite another user to channel | ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /remove | ID<int>  CID<int> | User removes another user form his private channel by sending his ID to server which removes channel flag | Removed(ID)  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /private | CID<int>  message<string> | User sends message to channel based on channel idr | None  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /leave | CID<int> | User leaves private channel | Left(CID)  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /close | CID<int> | User closes private channel | Closed(ID)  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |
| /logout | none | User logs out of system then communication with server closes | Loggedout(“”)  ERROR(“UNHANDLED EXCEPTION CHECK LOG”) |