Protocol Documentation

Module: Internet Technology
Student Name: Bako Asaad
Student Number: 510944
Delivery Date: 23-3-2024

Table of Contents

Introduction

Welcome to the Saxion chat application protocol documentation of the year 2024. This document serves as a guide to understanding how clients and the server interact with each other. The protocols outlined here cover various scenarios, including welcoming messages, user login/logoff, displaying online users, sending private messages, broadcasting messages, transferring files, guessing game, and handling unknown commands.

Protocols

1. Welcome Protocol

Use Cases

1. User logs in successfully	
Server to Client (S ---> C)	**WELCOME** {"msg" : "<Welcome to the Saxion chat application of 2024! Type (help) to view the Commands!>"}

2. Login Protocol

Use Cases

1. User logs in successfully	
Client to Server (C ---> S)	**LOGIN** {"username": "<username>"}
Server to Client (S ---> C)	**LOGIN_RESP** {"status":"OK"}
Server to Others (S ---> Others)	**JOINED** {"others":"<username>"}
2. User attempts to login again twice	
Client to Server (C ---> S)	**LOGIN** {"username": "<username>"}
Server to Client (S ---> C)	**LOGIN_RESP** {"status":"ERROR", "code":"<5002>"}
3. New user attempts to login with existing username	
Client to Server (C ---> S)	**LOGIN** {"username": "<username>"}
Server to Client (S ---> C)	**LOGIN_RESP** {"status":"ERROR", "code":"<5000>"}

4. User attempts to login with invalid username format or length	
Client to Server (C ---> S)	**LOGIN** {“username”: “<username>”}
Server to Client (S ---> C)	**LOGIN_RESP:** {“status”:”ERROR”, “code”:”<5001>”}

Error Descriptions

Error code	Description
5000	User already logged in
5001	Username has an invalid format or length
5002	User cannot log in twice

3. Ping Pong Protocol

Use Cases

1. Server sends Ping to client and client responds with Pong	
Server to Client (S ---> C)	**PING** {}
Client to Server (C ---> S)	**PONG** {}
2. Server receives unexpected Pong	
Client to Server (C ---> S)	**PONG** {}
Server to Client (S ---> C)	**PONG_ERROR:** {“code”:”<8000>”}
3. Server sends Ping to client and client does not respond within three seconds	
Server to Client (S ---> C)	**PING** {}
Server to Client (S ---> C)	**DSCN:** {“reason”:”<7000>”}

Error Descriptions

Error code	Description
7000	Pong timeout
8000	Pong without ping

4. Broadcast Message Protocol

Use Cases

1. User sends a message successfully	
Client to Server (C ---> S)	**BROADCAST_REQ** {“message”:”<message>”}
Server to Client (S ---> C)	**BROADCAST_RESP** {“status”:”OK”}
Server to Others (S ---> Others)	**BROADCAST** {“username”:”<username>”, “message”:”<message>”}
2. User attempts to send a message while not logged in	
Client to Server (C ---> S)	**BROADCAST_REQ** {“message”:”<message>”}
Server to Client (S ---> C)	**BROADCAST_RESP** {“status”:”ERROR”, “code”:”<6000>”}
3. User attempts to send an empty message	
Client to Server (C ---> S)	**BROADCAST_REQ** {“message”:”< >”}
Server to Client (S ---> C)	**BROADCAST_RESP** {“status”:”ERROR”, “code”:”<6000>”}

Error Descriptions

Error code	Description
6000	User is not logged in
6001	Cannot send an empty message

5. Private Message Protocol

Use Cases

1. User sends a private message successfully	
Client-1 to Server (C1 ---> S)	**PRIVATE_REQ** {“receiver”:”<C2 username>”, “message”:”<message>”}
Server to Client-1 (S ---> C1)	**PRIVATE_RESP** {“status”:”OK”}
Server to Client-2 (S ---> C2)	**PRIVATE** {“sender”:”<C1 username>”, “message”:”<message>”}
2. User sends a private message to themselves	
Client-1 to Server (C1 ---> S)	**PRIVATE_REQ** {“receiver”:”<C1 username>”, “msg”:”<message>”}
Server to Client-1 (S ---> C1)	**PRIVATE_RESP** {“status”:”ERROR”, “code”:”<9001>”}
3. User sends an empty private message	
Client-1 to Server (C1 ---> S)	**PRIVATE_REQ** {“receiver”:”<C2 username>”, “message”:”<>”}
Server to Client-1 (S ---> C1)	**PRIVATE_RESP** {“status”:”ERROR”, “code”:”<6001>”}
4. User sends a private message while not logged in	
Client-1 to Server (C1 ---> S)	**PRIVATE_REQ** {“receiver”:”<C2 username>”, “message”:”<message>”}
Server to Client-1 (S ---> C1)	**PRIVATE_RESP** {“status”:”ERROR”, “code”:”<6000>”}
5. User sends a private message to a non-existent user	
Client-1 to Server (C1 ---> S)	**PRIVATE_REQ** {“receiver”:”<C2 username>”, “message”:”<message>”}
Server to Client-1 (S ---> C1)	**PRIVATE_RESP** {“status”:”ERROR”, “code”:”<9000>”}

Error Descriptions

Error code	Description
9000	User does not exist
9001	Cannot send a private message to yourself

6. Online Users Protocol

Use Cases

1. User requests to see online users successfully	
Client-1 to Server (C1 ---> S)	**ONLINE_USERS_REQ:** {“username”:”<C1 username>”}
Server to Client-1 (S ---> C1)	**ONLINE_USERS_RESP:** {“status”:”OK”}
Server to Client-1 (S ---> C1)	**ONLINE_USERS:** {“onlineUsers”: ["<user2>", "<user3>", ...]}
2. User requests to see online users while not logged in	
Client to Server (C ---> S)	**ONLINE_USERS_REQ:** {“username”:”<null >”}
Server to Client (S ---> C)	**ONLINE_USERS_RESP:** {” status”:”ERROR”, “code”:”<6000>”}
3. User requests to see online users but no other users are online	
Client to Server (C---> S)	**ONLINE_USERS_REQ:** {“username”:”<C1 username>”}
Server to Client (S ---> C)	**ONLINE_USERS_RESP:** {” status”:”ERROR”, “code”:”<10000>”}

Error Descriptions

Error code	Description
10000	There are no users online

7. Logoff Protocol

Use Cases

1. User logs off successfully	
Client to Server (C ---> S)	**BYE** {}
Server to Client (S ---> C)	**BYE_RESP** {"status":"OK"}
Server to Others (S ---> Others)	**LEFT** {"others":"<username>"}

8. Unknown Command Protocol

Use Cases

1. User sends an invalid message to server	
Client to Server (C ---> S)	**MSG** {"<Invalid Message>"}
Server to Client (S ---> C)	**UNKNOWN_COMMAND** {}

9. Guessing Game Protocol

Use Cases

1. User starts a guessing game successfully while two users are online	
• **Step-1:** User sends a guessing game invite	
Client-1 to Server (C1 ---> S)	**GUESSING_GAME_INVITE_REQ** {"requester":"<C1 username>"}
Server to Client-1 (S ---> C1)	**GUESSING_GAME_INVITE_RESP** {"status":"OK"}
Server to Online-Users (S ---> Online-Users)	**GUESSING_GAME_INVITE** {"requester":"<C1 username>"}
• **Step-2:** Users sends join request within ten second limit	
Client-1-2 to Server (C1-C2 ---> S)	**GUESSING_GAME_JOIN_REQ** {"player":"<username>"}

Server to Client-1-2 (S ---> C1-C2)	**GUESSING_GAME_JOIN_RESP** {“status”:”OK”}	
• **Step-3:** Users start playing the guessing game within a two minute time limit		
Client to Server (C ---> S)	**GUESS_REQ** {“player”:””<username>”, “guess”:”<guessedNumber>”}	
Server to Client (S ---> C)	OR	**GUESS_RESP** {“status”: “OK”}
	OR	**GUESS_RESP** {“status”:”ERROR”, “code”:”<11008>”}
	OR	**GUESS_RESP** {“status”:”ERROR”, “code”:”<11009>”}
	OR	**GUESS_RESP** {“status”:”ERROR”, “code”:”<11010>”}
• **Step-4:** Server sends game result to the users who have guessed correctly		
Server to Players (S ---> Players)	**GUESSING_GAME_RESULT** {“gameResults”:[“<position>, <player>, <winner>, <timeTaken>”]}	
2. User sends a guessing game invite while not enough users are online		
Client to Server (C ---> S)	**GUESSING_GAME_INVITE_REQ** {“requester”:”<username>”}	
Server to Client (S ---> C)	**GUESSING_GAME_INVITE_RESP** {“status”:”ERROR”, “code”:”<11000>”}	
3. User sends a guessing game invite and before ten seconds are over, user gets disconnected		
Client to Server (C ---> S)	**GUESSING_GAME_INVITE_REQ** {“requester”:”<username>”}	
Server to Client (S ---> C)	**GUESSING_GAME_INVITE_RESP** {“status”:”ERROR”, “code”:”<11001>”}	
4. User sends a guessing game invite while there is an active game or another user already requested a game		
Client to Server (C ---> S)	**GUESSING_GAME_INVITE_REQ** {“requester”:”<username>”}	
Server to Client (S ---> C)	**GUESSING_GAME_INVITE_RESP** {“status”:”ERROR”, “code”:”<11002>”}	
5. User attempts to join a guessing game after ten seconds has passed		
Client to Server (C ---> S)	**GUESSING_GAME_JOIN_REQ** {“player”:”<username>”}	
Server to Client (S ---> C)	**GUESSING_GAME_JOIN_RESP** {“status”:”ERROR”, “code”:”<11003>”}	

6. User attempts to join a guessing game without receiving an invitation	
Client to Server (C ---> S)	**GUESSING_GAME_JOIN_REQ** {“player”:”<username>”}
Server to Client (S ---> C)	**GUESSING_GAME_JOIN_RESP** {“status”:”ERROR”, “code”:“<11004>”}
7. User receives an invitation, and user joins and attempts to start guessing number before the ten seconds are passed	
Client to Server (C ---> S)	**GUESS_REQ** {“player”:””<username>”, “guess”:”< guessedNumber>”}
Server to Client (S ---> C)	**GAME_RESP** {“status”:”ERROR”, “code”:“<11005>”}
8. User receives an invitation and attempts to start guessing number without joining the game	
Client to Server (C ---> S)	**GUESS_REQ** {“player”:””<username>”, “guess”:”< guessedNumber>”}
Server to Client (S ---> C)	**GAME_RESP** {“status”:”ERROR”, “code”:“<11006>”}
9. User attempts to start guessing number without receiving an invitation	
Client to Server (C ---> S)	**GUESS_REQ** {“player”:””<username>”, “guess”:”< guessedNumber>”}
Server to Client (S ---> C)	**GAME_RESP** {“status”:”ERROR”, “code”:“<11004>”}
10. User already participated in a game and guessed correctly, and before the game is over and while waiting for the result, the user attempts to send another guess	
Client to Server (C ---> S)	**GUESS_REQ** {“player”:””<username>”, “guess”:”< guessedNumber>”}
Server to Client (S ---> C)	**GAME_RESP** {“status”:”ERROR”, “code”:“<11007>”}

Error Descriptions

Error code	Description
11000	Game cancelled, not enough users joined the game
11001	Game cancelled, user who initiated the game is not online anymore
11002	Cannot send a game request! There is an active guessing game now
11003	Ten seconds passed, you cannot join anymore
11004	An invitation is required
11005	Waiting for the rest of the players to join. Game can start in any second.
11006	User needs to join first
11007	User Already Guessed Correctly

11008	Your guess was too low
11009	Your guess was too high
11010	Invalid Guess, please guess a number between 1 and 50

10. File Transfer Protocol

Use Cases

1. User1 sends a file successfully after user2 accepts the file

- **Step-1:** User1 sends the filename

Client-1 to Server (C1 ---> S)	**FILE_TRANSFER_REQ** {“receiver”:”<C2 username>”, “filename”:”<filename>”}
Server to Client-1 (S ---> C1)	**FILE_TRANSFER_RESP** {“status”:”OK”}
Server to Client-2 (S ---> C2)	**FILE_TRANSFER** {“sender”:”<C1 username>”, “filename”:”<filename>”}

- **Step-2:** Users2 accepts the file

Client-2 to Server (C2 ---> S)	**ACCEPT_FILE_TRANSFER_REQ** {“sender”:”<C1 username>”, “filename”:”<filename>”}
Server to Client-2 (S ---> C2)	**ACCEPT_FILE_TRANSFER_RESP** {“status”:”OK”}
Server to Client-1 (S ---> C1)	**ACCEPT_FILE_TRANSFER** {“status”:”ACCEPTED”, “receiver”:<”C2 username”>, “filename”:<”filename”>}

2. User sends file to him/herself

Client to Server (C ---> S)	**FILE_TRANSFER_REQ** {“receiver”:”<C1 username>”, “filename”:”<filename>”}
Server to Client (S ---> C)	**FILE_TRANSFER_RESP** {“status”:”ERROR”, “code”:”<12000>”}

3. User sends file to non existing user

Client to Server (C ---> S)	**FILE_TRANSFER_REQ** {“receiver”:”<C2 username>”, “filename”:”<filename>”}
Server to Client (S ---> C)	**FILE_TRANSFER_RESP** {“status”:”ERROR”, “code”:”<9000>”}

4. User2 accepts file while user1 is no longer online

Client to Server (C ---> S)	**ACCEPT_FILE_TRANSFER_REQ** {“sender”:”<C1 username>”, “filename”:”<filename>”}
Server to Client (S ---> C)	**ACCEPT_FILE_TRANSFER_RESP** {“status”:”ERROR”, “code”:”<9000>”}
5. User2 accepts file while file is no longer available for transfer	
Client to Server (C ---> S)	**ACCEPT_FILE_TRANSFER_REQ** {“sender”:”<C1 username>”, “filename”:”<filename>”}
Server to Client (S ---> C)	**ACCEPT_FILE_TRANSFER_RESP** {“status”:”ERROR”, “code”:”<12001>”}
6. User1 sends file and user2 declines	
• **Step-1:** User1 sends the filename	
Client-1 to Server (C1 ---> S)	**FILE_TRANSFER_REQ** {“receiver”:”<C2 username>”, “filename”:”<filename>”}
Server to Client-1 (S ---> C1)	**FILE_TRANSFER_RESP** {“status”:”OK”}
Server to Client-2 (S ---> C2)	**FILE_TRANSFER** {“sender”:”<C1 username>”, “filename”:”<filename>”}
• **Step-2:** Users2 declines the file	
Client-2 to Server (C2 ---> S)	**DECLINE_FILE_TRANSFER_REQ** {“sender”:”<C1 username>”, “filename”:”<filename>”}
Server to Client-2 (S ---> C2)	**DECLINE_FILE_TRANSFER_RESP** {“status”:”OK”}
Server to Client-1 (S ---> C1)	**DECLINE_FILE_TRANSFER** {“status”:”DECLINED”, “receiver”:<”C2 username”>, “filename”:<”filename”>}
7. User sends file while not logged in	
Client to Server (C ---> S)	**FILE_TRANSFER_REQ** {“receiver”:”<C2 username>”, “filename”:”<filename>”}
Server to Client (S ---> C)	**FILE_TRANSFER_RESP** {“status”:”ERROR”, “code”:”<6000>”}
8. User sends a empty username or filename	
Client to Server (C ---> S)	**FILE_TRANSFER_REQ** {“receiver”:”<C2 username>”, “filename”:”<>”}
Server to Client	**FILE_TRANSFER_RESP** {“status”:”ERROR”, “code”:”<12002>”}

(S ---> C)	

Error Descriptions

Error code	Description
12000	Cannot send file to yourself
12001	File is no longer available for transfer
12002	Username or filename cannot be empty

11. Parse Error Protocol

Use Case

1. **For every feature on the chat application, when a client make a request to the server and the request has an invalid Json format, then a 'Parse Error' response will be sent back to the client.**

Client to Server (C ---> S)	**<Invalid Json Format>**
Server to Client (S ---> C)	**PARSE_ERROR** {}