Report Documentation

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

Contents

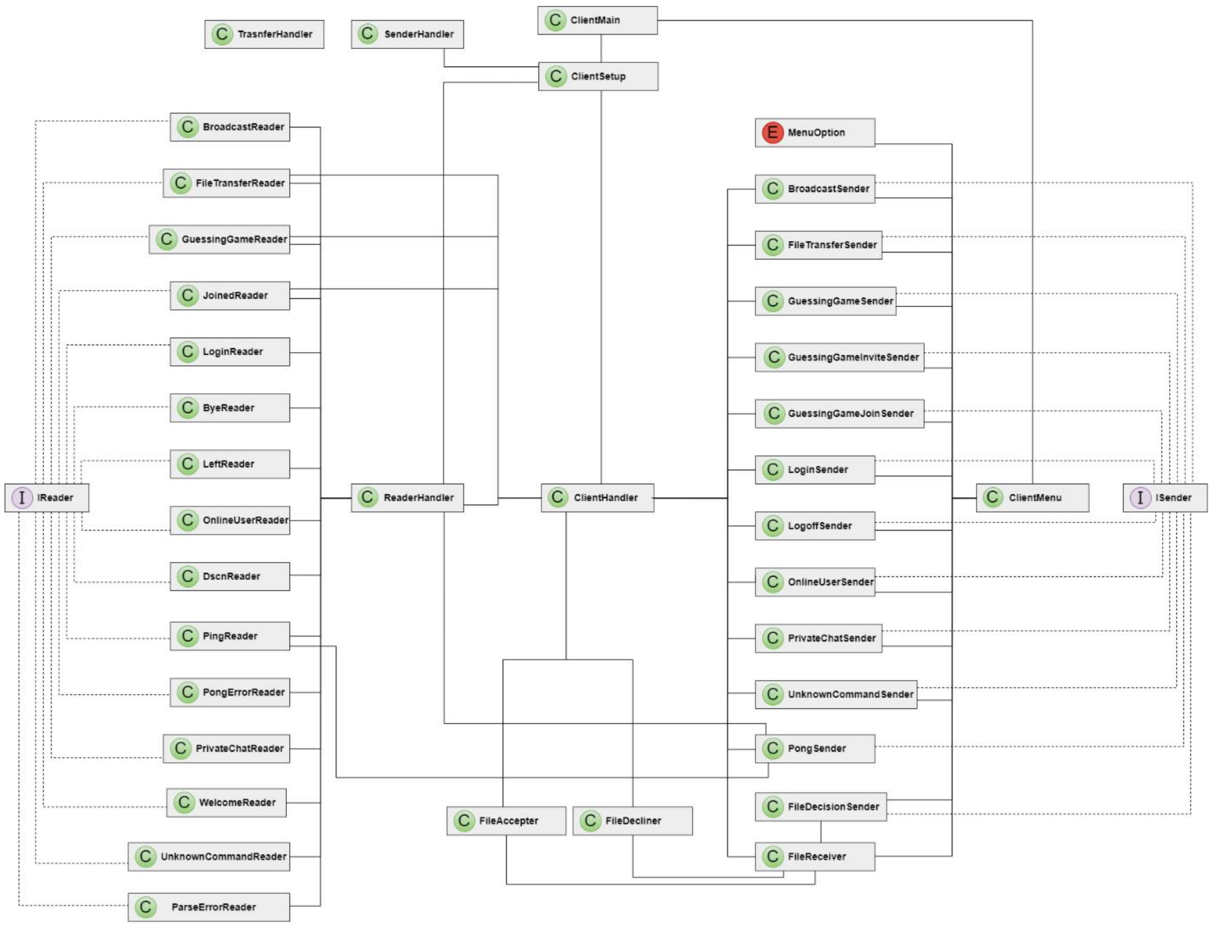
Architecture Overview and Class Diagrams

Client-side

- clients
  - **ClientMain:** Initializes the client setup, creates a client menu, start the application and connects it to the server.
  - **ClientSetup:** Sets up the connection between client and the server through three important classes 'SenderHandler', 'ReaderHandler', and 'ClientHandler'.
  - handlers
    - **ClientHandler:** Handles overall reading and sending messages between the client and server. Including managing user input, message sending, connection status, user status, and file handling.
    - **ReaderHandler:** Handle reading responses from the server and direct each response to a specific handler.
    - **SenderHandler:** Handles sending messages to the server.
    - broadcasts
      - **BroadcastSender:** Sends broadcast messages to the server.
      - **BroadcastReader:** Reads and processes broadcast messages from the server.
    - fileTransfers
      - **FileTransferSender:** Sends client request to send a file to another client.
      - **FileTransferReader:** Reads and processes various types of file transfer responses from the server.
      - **TransferHandler:** Responsible for the actual transmission of a file between two clients, handles the client socket connection, data streams, and file transfer process.
      - receivedFiles
        - **FileAccepter:** Handles sending acceptance response of file transfer to the server.
        - **FileDecliner:** Handles sending rejection response of file transfer to the server.
        - **FileReceiver:** Responsible for handling incoming file transfer requests from the server. It presents the client with a menu interface, allowing them to decide whether to accept or decline incoming files. This class uses instances of 'FileAccepter' and 'FileDecliner' classes to process user decisions.
        - **FileDecisionSender:** Sends the client decision to the server after receiving file requests.
    - guessingGames
      - **GuessingGameInviteSender:** Sends request to the server to initiate a guessing game.
      - **GuessingGameJoinSender:** Sends join request for the guessing game to the server.
      - **GuessingGameSender:** Presents a user interface so that clients can send their guesses to the server.

        - **GuessingGameReader:** Reads and process various types of guessing game response from the server, including invitation, join, user guess, and game result responses. It also handles a couple of Booleans to control the client's participation and entry requirements for the game based on the responses that comes from the server.
      - logins
        - **LoginSender:** Sends login request to the server.
        - **LoginReader:** Reads and process server responses regarding user login status.
        - **JoinedReader:** Reads and process server response regarding notifying other online users about new user logins.
      - logoffs
        - **LogoffSender:** Sends a request to the server indicating that the client is leaving the chat.
        - **ByeReader:** Reads the server's response to the client's logoff request and exits the program with a message.
        - **LeftReader:** Reads the server's response that notifies other online users that a user left the chat.
      - onlineUsers
        - **OnlineUserSender:** Sends a request to the server to retrieve the list of online users.
        - **OnlineUserReader:** Reads and processes responses from the server regarding online users.
      - parseErrors
        - **ParseErrorReader:** Reads parse error responses from the server.
      - pingPongs
        - **DscnReader:** Reads disconnected responses from the server.
        - **PingReader:** Reads ping responses from the server.
        - **PongErrorReader:** Reads pong error responses from the server.
        - **PongSender:** Sends pong message to the server after receiving Ping.
      - privateChats
        - **PrivateChatSender:** Sends a private chat message to a specific user through the server.
        - **PrivateChatReader:** Reads various types of private chat responses from the server.
      - unknownCommands
        - **UnknownCommandSender:** Handles sending unknown command to the server.
        - **UnknownCommandReader:** Reads server responses and notifies the user that its request was invalid.
      - welcomes
        - **WelcomeReader:** Reads server response and displays a welcome message when a client connects to the server.
    - menu
      - **ClientMenu:** This Class handles the main menu functionality, allowing users to interact with the server based on provided options.
      - **MenuOption:** This Enum defines options in the client menu.

    - **messageInterfaces**
        - **IReader:** Defines the ‘readResponse()’ method for reading server responses.
        - **ISender:** Defines the ‘sendRequest()’ method for sending messages to server.



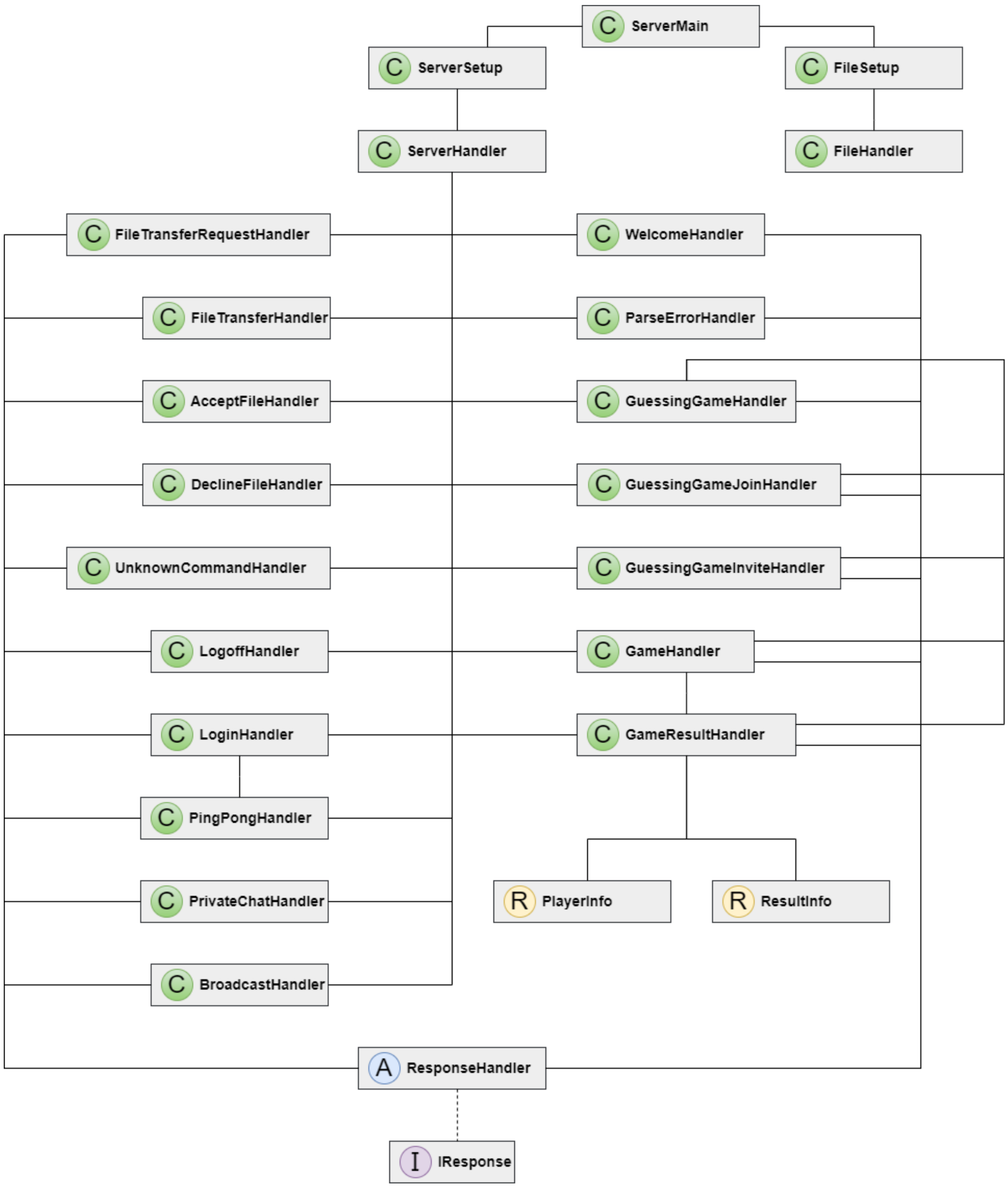
Communications

- **communications**
    - **Command:** Defines constants for various commands.
    - **ErrorCode:** Defines constants for various codes.
    - **Message:** Defines constants for various messages.
    - **Status:** Defines constants such as ’OK’, ‘ERROR’, ‘ACCEPTED’, and ‘DECLINED’.
    - **protocol**
        - **messages**
            - This package contains a various records.
        - **Utils**
            - **Utils:** This class defines each record’s header with a specific command, converts objects to message strings and parsing messages back into Java objects, to simplify communication between client and server.

Server-side

- servers
  - **ServerMain:** Initializes and starts separate threads for server and file transfer operations.
  - **ServerSetup:** Listens for incoming client connection on a specified port, creates a new ‘ServerHandler’ instance for each client connection, and keeps track of connected users through users map.
  - **FileSetup:** Listens for incoming file transfer connections on a specified port, creates a new ‘FileHandler’ instance for each connection, and maintains a map of connected users specifically for file transfers.
  - handlers
    - **IHandler:** This interface defines methods for sending various types of responses from the server to the client.
    - **ResponseHandler:** This abstract class implements ‘IHandler’ interface to override its methods, ensuring that subclasses override these methods with specific implementations as needed.
    - **ServerHandler:**
      1. Initializes input and output streams for communication with clients.
      2. Handles client connections and disconnections.
      3. Listens for client’s messages and direct them to a specific handler.
      4. Provides several utility methods for sending and printing messages, managing files, and accessing client information.
    - broadcasts
      - **BroadcastHandler:** Handles broadcast message from clients, ensuring the message is valid and sending it to all other logged-in users expect the sender, and notifies the sender about his/her message status.
    - fileTransfers
      - **FileTransferHandler:** Handles different types of requests that comes from the clients, such as file transfer request, accept file, decline file and directs them to a specific handler.
      - **FileTransferRequestHandler:** Handles incoming transfer requests, checks the request and sending responses to the client accordingly.
      - **AcceptFileHandler:** Handles the acceptance of file transfer requests.
      - **DeclineFileHandler:** Handles the rejection of file transfer requests.
      - **FileHandler:** Handles the file transfer between clients, receiving file data from the sender and forwarding it to the receiver. It extends ‘Thread’ to handle concurrent file transfer operations.
    - guessingGames
      - **GuessingGameHandler:** This class provides methods to start and reset one game instance at a time using Singleton, handles different types of guessing game messages from the client, such as requesting, joining, and playing a guessing game. Additionally, it contains countdown methods to allow users to join and play in a specific period of time and some other methods to be used in the other guessing game classes.

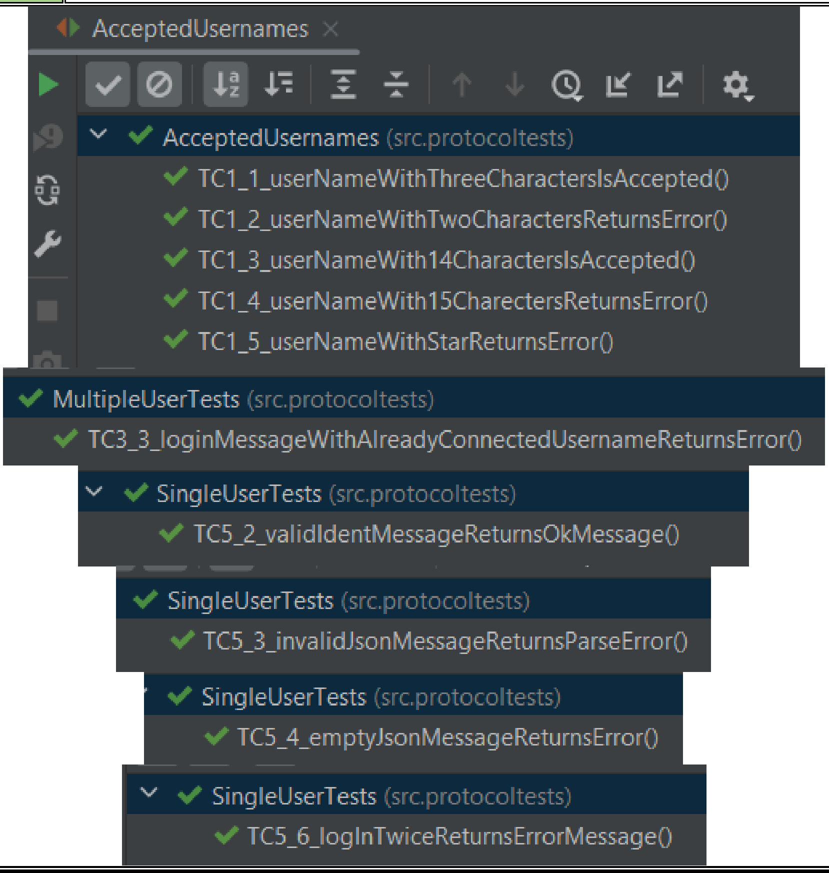
    - **GameInviteHandler:** Handles user request to initiate a guessing game.
    - **GameJoinHandler:** The class responds to join requests with appropriate status messages and codes, informing players about the success or failure of their join attempt.
    - **GameHandler:** Handles player guesses in the guessing game, ensures each guess is within the valid range, updates player information upon guesses, and handles game results when the game ends.
    - **GameResultHandler:** Calculates and sends the result of the guessing game to all involved players who guessed correctly.
    - **PlayerInfo:** records player name and the time that took the player to guess correctly.
    - **ResultInfo:** records player's position, name, time taken, and whether the player is a winner or not.
- logins
    - **LoginHandler:** Handles user login requests, ensuring the validity of usernames and handles different scenarios.
- logoffs
    - **LogoffHandler:** Handles user logoff requests, sending bye response to client, updating user status, and removes client from server.
- onlineUsers
    - **OnlineUserHandler:** Handles sending list of online users to client.
- parseErrors
    - **ParseErrorHandler:** Handles cases where users sends invalid Json request.
- pingPongs
    - **PingPongHandler:** Sends periodic ping messages and handles corresponding pong responses between server and client to maintain connection stability, and handles disconnections if pong responses are not received in time.
- privateChats
    - **PrivateChatHandler:** Handles private messages of clients, by sending a private message from sender to receiver, and responds accordingly in case of errors or invalid input.
- unknownCommands
    - **UnknownCommandHandler:** Handles unknown commands.
- usersStatus
    - **UserStatusHandler:** Handles sending 'Joined' or 'Left' status response to online users when a user logs off.
- welcomes
    - **WelcomeHandler:** Sends welcome message to client upon successful connection.


ServerMain
ServerSetup
FileSetup
ServerHandler
FileHandler
FileTransferRequestHandler
WelcomeHandler
FileTransferHandler
ParseErrorHandler
AcceptFileHandler
GuessingGameHandler
DeclineFileHandler
GuessingGameJoinHandler
UnknownCommandHandler
GuessingGameInviteHandler
LogoffHandler
GameHandler
LoginHandler
GameResultHandler
PingPongHandler
PrivateChatHandler
PlayerInfo
ResultInfo
BroadcastHandler
ResponseHandler
IResponse

Tests

Login Tests

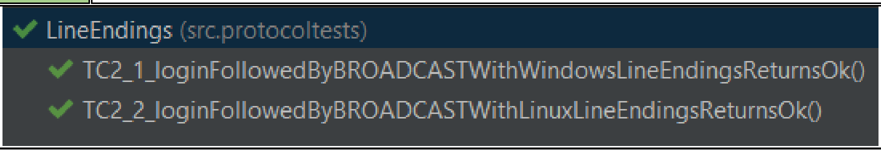
Test case	Result	Details
TC1_1	OK	User logs in with three characters and returns **ok**.
TC1_2	OK	User attempts to log in with two characters and returns **5001** error.
TC1_3	OK	User logs in with fourteen characters and returns **ok**.
TC1_4	OK	User attempts to log in with fifteen characters and returns an **5001** error.
TC1_5	OK	User attempts to log in using star as a character and returns **5001** error.
TC1_6	OK	User attempts to log in using an existing name and returns **5000** error.
TC3_3	OK	User attempts to log in using an existing name and returns **5000** error.
TC5_2	OK	User logs in with a valid username and returns **ok**.
TC5_3	OK	User sends an invalid Json message and returns **ParseError**.
TC5_4	OK	User sends a command without body and returns **5001** error.
TC5_6	OK	Logged in user attempts to log in for the second time and returns **5002** error.



```
✓ AcceptedUsernames (src.protocoltests)
    ✓ TC1_1_userNameWithThreeCharactersIsAccepted()
    ✓ TC1_2_userNameWithTwoCharactersReturnsError()
    ✓ TC1_3_userNameWith14CharactersIsAccepted()
    ✓ TC1_4_userNameWith15CharectersReturnsError()
    ✓ TC1_5_userNameWithStarReturnsError()
✓ MultipleUserTests (src.protocoltests)
    ✓ TC3_3_loginMessageWithAlreadyConnectedUsernameReturnsError()
✓ SingleUserTests (src.protocoltests)
    ✓ TC5_2_validIdentMessageReturnsOkMessage()
✓ SingleUserTests (src.protocoltests)
    ✓ TC5_3_invalidJsonMessageReturnsParseError()
✓ SingleUserTests (src.protocoltests)
    ✓ TC5_4_emptyJsonMessageReturnsError()
✓ SingleUserTests (src.protocoltests)
    ✓ TC5_6_logInTwiceReturnsErrorMessage()
```

Line ending Tests

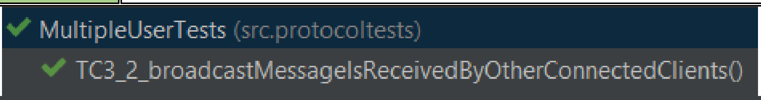
Test case	Result	Details
TC2_1	OK	User logs in with Windows line endings followed by a broadcast request, and the system responds with **OK** for both login and broadcast.
TC2_2	OK	User logs in with Linux line endings followed by a broadcast request, and the system responds with **OK** for both login and broadcast.



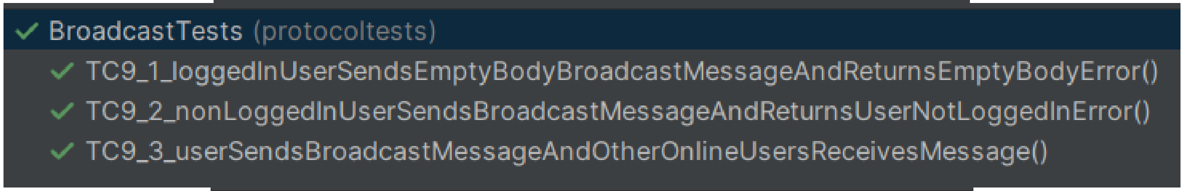
✓ LineEndings (src.protocoltests)
✓ TC2_1_loginFollowedByBROADCASTWithWindowsLineEndingsReturnsOk()
✓ TC2_2_loginFollowedByBROADCASTWithLinuxLineEndingsReturnsOk()

Broadcast Tests

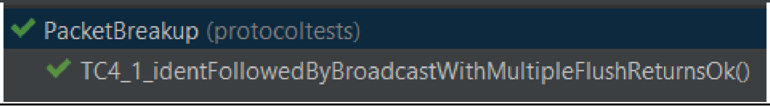
Test case	Result	Details
TC3_2	OK	Broadcast messages from one user are correctly received by the other logged in user and both users receive **OK** status responses for their broadcast action.
TC9_1	OK	User sends an empty broadcast message and returns an **EMPTY_BODY** error.
TC9_2	OK	user sends broadcast message and returns user not logged in.
TC9_3	OK	User sends broadcast chat and other online users receives message
TC4_1	OK	The login and broadcast messages are sent with multiple flush and returns **OK** status for both the login and broadcast actions.



✓ MultipleUserTests (src.protocoltests)
✓ TC3_2_broadcastMessageIsReceivedByOtherConnectedClients()



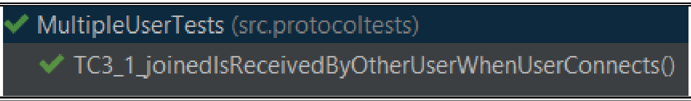
✓ BroadcastTests (protocoltests)
✓ TC9_1_loggedInUserSendsEmptyBodyBroadcastMessageAndReturnsEmptyBodyError()
✓ TC9_2_nonLoggedInUserSendsBroadcastMessageAndReturnsUserNotLoggedInError()
✓ TC9_3_userSendsBroadcastMessageAndOtherOnlineUsersReceivesMessage()



✓ PacketBreakup (protocoltests)
✓ TC4_1_identFollowedByBroadcastWithMultipleFlushReturnsOk()

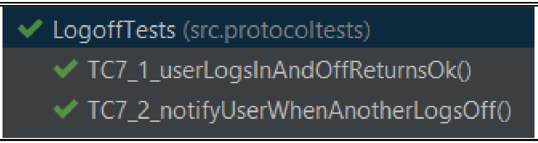
Joined Tests

Test case	Result	Details
TC3_1	OK	When user1 and user2 connect, the system correctly notifies online users by sending the logged in **username** with a **joined** command



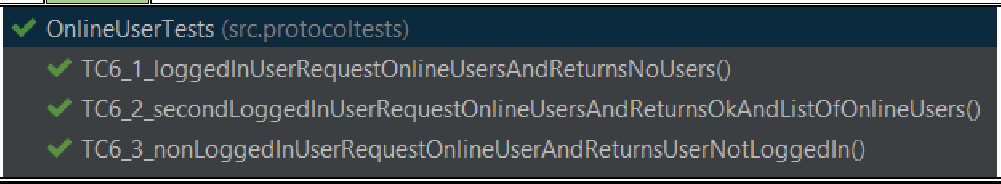
✓ MultipleUserTests (src.protocoltests)
✓ TC3_1_joinedIsReceivedByOtherUserWhenUserConnects()

Logoff Tests

Test case	Result	Details
TC7_1	OK	Logged in user logs off and returns **OK**.
TC7_2	OK	When user logs off, other online users will be notified that the user left.
		

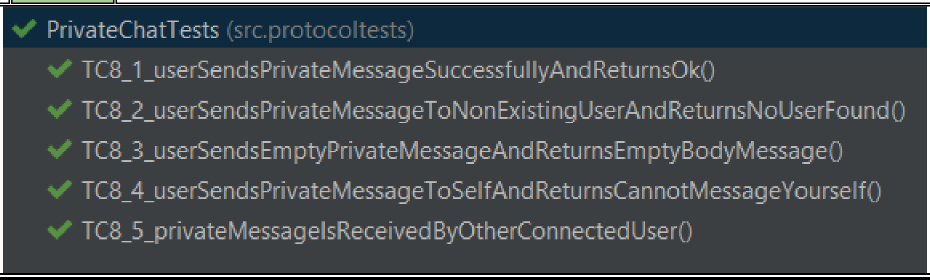
```
✓ LogoffTests (src.protocoltests)
    ✓ TC7_1_userLogsInAndOffReturnsOk()
    ✓ TC7_2_notifyUserWhenAnotherLogsOff()
```

User Status Tests

Test case	Result	Details
TC6_1	OK	Logged in user request to see online users and returns **NO_USERS** error code.
TC6_2	OK	Logged in user request to see online users and returns **OK** with a list of online users.
TC6_3	OK	Non logged in user request to see online users and returns and receives **6000** error code.
		

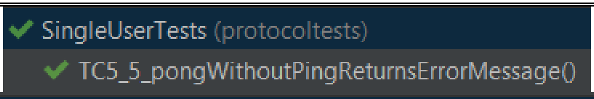
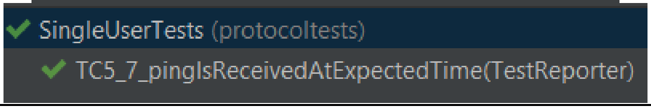
```
✓ OnlineUserTests (src.protocoltests)
    ✓ TC6_1_loggedInUserRequestOnlineUsersAndReturnsNoUsers()
    ✓ TC6_2_secondLoggedInUserRequestOnlineUsersAndReturnsOkAndListOfOnlineUsers()
    ✓ TC6_3_nonLoggedInUserRequestOnlineUserAndReturnsUserNotLoggedIn()
```

Privatechat Tests

Test case	Result	Details
TC8_1	OK	Logged in user sends private message successfully and receives an **OK**.
TC8_2	OK	Logged in user sends private message to non existing user and returns the **USER_NOT_FOUND** error code.
TC8_3	OK	Logged in user sends empty private message and returns **EMPTY_BODY** error code.
TC8_4	OK	Logged in user sends private message to him/herself and returns **CANNOT_MESSAGE_YOURSELF** error code.
TC8_5	OK	Logged in user sends private message successfully and receiver receives the **sender name** and the **message**.
		

```
✓ PrivateChatTests (src.protocoltests)
    ✓ TC8_1_userSendsPrivateMessageSuccessfullyAndReturnsOk()
    ✓ TC8_2_userSendsPrivateMessageToNonExistingUserAndReturnsNoUserFound()
    ✓ TC8_3_userSendsEmptyPrivateMessageAndReturnsEmptyBodyMessage()
    ✓ TC8_4_userSendsPrivateMessageToSelfAndReturnsCannotMessageYourself()
    ✓ TC8_5_privateMessageIsReceivedByOtherConnectedUser()
```

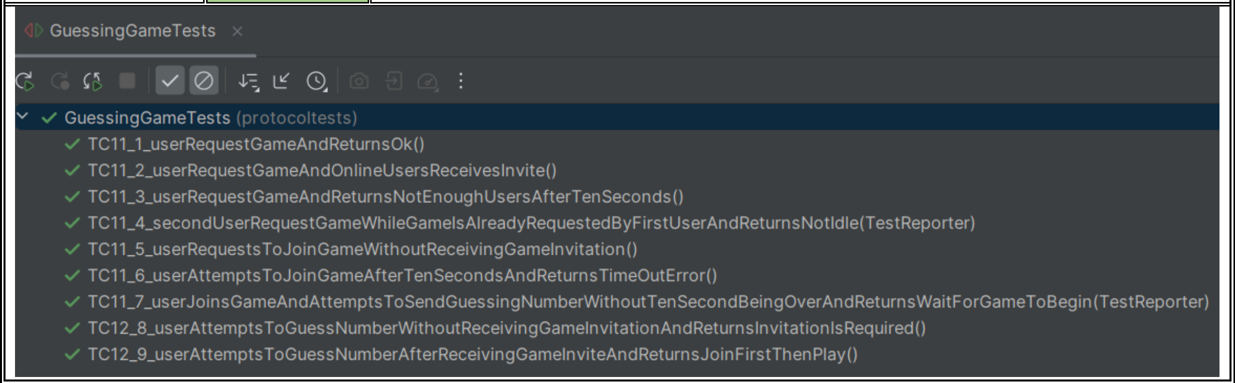
Ping Pong Tests

Test case	Result	Details
TC5_5	OK	PONG message is sent without a receiving PING message first. The system responds with an **8000** error code.
TC5_7	OK	PING message is received at the expected time and the test checks that the response time falls within the allowed range.
 		

```
✓ SingleUserTests (protocoltests)
    ✓ TC5_5_pongWithoutPingReturnsErrorMessage()
```

```
✓ SingleUserTests (protocoltests)
    ✓ TC5_7_pingIsReceivedAtExpectedTime(TestReporter)
```

Guessing Game Tests

Test case	Result	Details
TC11_1	OK	User requests a guessing game and receives a successful response
TC11_2	OK	User requests a guessing game, and online users receive an invitation
TC11_3	OK	User requests a guessing game, but after ten seconds, user receives not enough users joined response
TC11_4	OK	Second user requests a game while it is already requested by the first user, and returns a not idle response
TC11_5	OK	User requests to join a game without receiving a game invitation and returns an invitation is required
TC11_6	OK	User attempts to join a game after ten seconds and returns, cannot join now response
TC11_7	OK	User joins a game and attempts to send a guessing number before ten seconds are over, resulting in a “wait for the game to begin” response.
TC11_8	OK	User attempts to guess a number without receiving a game invitation and receives an "Invitation is required" response.
TC11_9	OK	User attempts to guess a number after receiving game invitation, without joining first, and returns user needs to join first response.
		

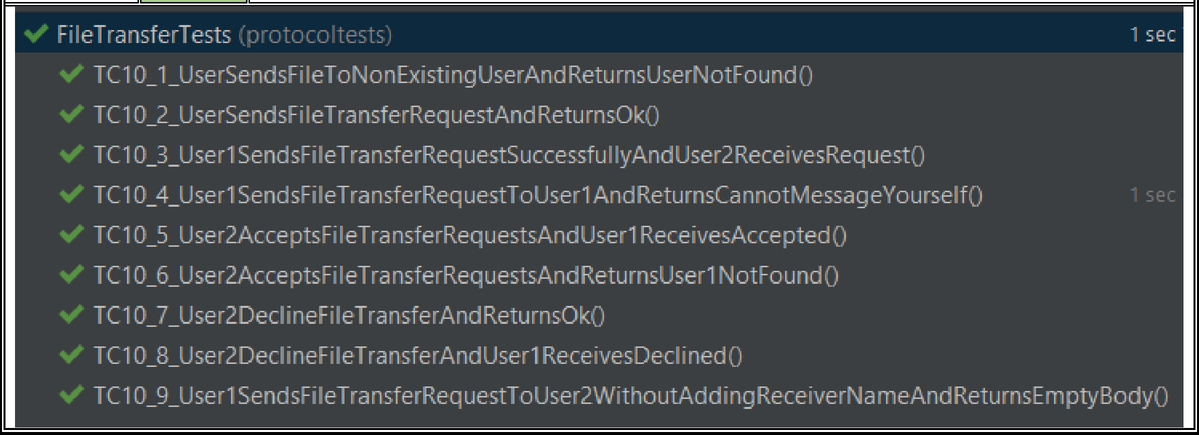
```
GuessingGameTests ×

✓ GuessingGameTests (protocoltests)
    ✓ TC11_1_userRequestGameAndReturnsOk()
    ✓ TC11_2_userRequestGameAndOnlineUsersReceivesInvite()
    ✓ TC11_3_userRequestGameAndReturnsNotEnoughUsersAfterTenSeconds()
    ✓ TC11_4_secondUserRequestGameWhileGameIsAlreadyRequestedByFirstUserAndReturnsNotIdle(TestReporter)
    ✓ TC11_5_userRequestsToJoinGameWithoutReceivingGameInvitation()
    ✓ TC11_6_userAttemptsToJoinGameAfterTenSecondsAndReturnsTimeOutError()
    ✓ TC11_7_userJoinsGameAndAttemptsToSendGuessingNumberWithoutTenSecondBeingOverAndReturnsWaitForGameToBegin(TestReporter)
    ✓ TC12_8_userAttemptsToGuessNumberWithoutReceivingGameInvitationAndReturnsInvitationIsRequired()
    ✓ TC12_9_userAttemptsToGuessNumberAfterReceivingGameInviteAndReturnsJoinFirstThenPlay()
```

File Transfer Tests

Automated Tests

Test case	Result	Details
TC10_1	OK	User1 sends a file transfer request to a non-existing user (user2), and the server responds with **USER_NOT_FOUND** error.
TC10_2	OK	User1 sends a file transfer request to user2, and the server responds with **OK**.
TC10_3	OK	User1 sends a file transfer request to user2. User1 receives **OK**, and user2 receives the **sender name** and the **filename**.
TC10_4	OK	User1 attempts to send a file transfer request to themselves, resulting in a **CANNOT_MESSAGE_YOURSELF** error.
TC10_5	OK	User2 accepts a file transfer request from user1, and user1 receives an **ACCEPTED** response.
TC10_6	OK	User2 attempts to accept a file transfer request from user1 after user1 disconnects, and the server responds with **USER_NOT_FOUND** error.
TC10_7	OK	User2 declines a file transfer request from user1, and the server responds with **OK**.
TC10_8	OK	User2 declines a file transfer request from user1, and user1 receives a **DECLINED** response.
TC10_9	OK	User1 sends a file transfer request to user2 without specifying the receiver's name, resulting in an **EMPTY_BODY** error.



```
✓ FileTransferTests (protocoltests)                                                          1 sec
    ✓ TC10_1_UserSendsFileToNonExistingUserAndReturnsUserNotFound()
    ✓ TC10_2_UserSendsFileTransferRequestAndReturnsOk()
    ✓ TC10_3_User1SendsFileTransferRequestSuccessfullyAndUser2ReceivesRequest()
    ✓ TC10_4_User1SendsFileTransferRequestToUser1AndReturnsCannotMessageYourself()            1 sec
    ✓ TC10_5_User2AcceptsFileTransferRequestsAndUser1ReceivesAccepted()
    ✓ TC10_6_User2AcceptsFileTransferRequestsAndReturnsUser1NotFound()
    ✓ TC10_7_User2DeclineFileTransferAndReturnsOk()
    ✓ TC10_8_User2DeclineFileTransferAndUser1ReceivesDeclined()
    ✓ TC10_9_User1SendsFileTransferRequestToUser2WithoutAddingReceiverNameAndReturnsEmptyBody()
```

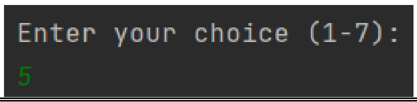
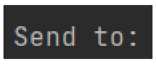
Manual Tests

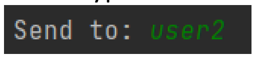
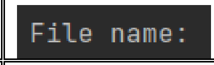
10th Test

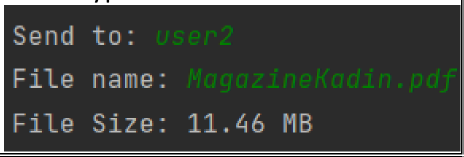
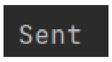
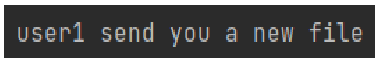
Test Case No.	TC10_10
Name	Transmit Large Binary PDF File
Short description	Clients successfully sends a binary PDF file of 2 or more MB.
System	File transfer
Requirement No.	RQ-U303

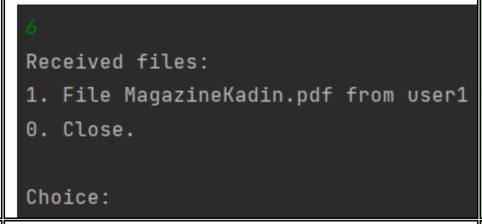
Preconditions
1. Both User (sender) and (receiver) needs to be logged in. 2. A pdf file with at least 2MB is available.

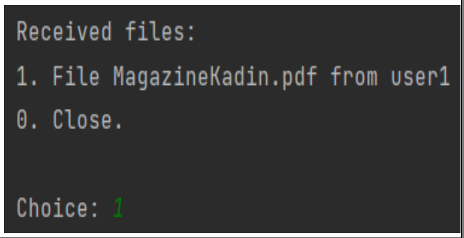
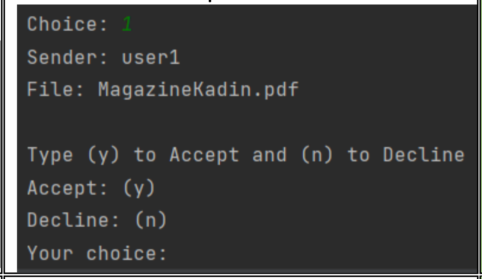
Steps	Action	Expected outcome	Pass/Fail

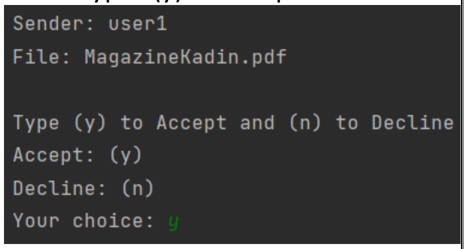
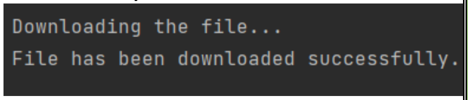
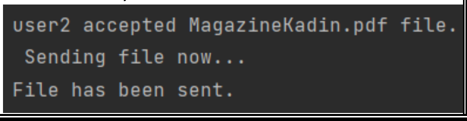
1	User1 types option 5 from the main menu
	
	`Enter your choice (1-7):`
	`5`
		User1 is prompted to enter the receiver name.
		
		`Send to:`
			Pass

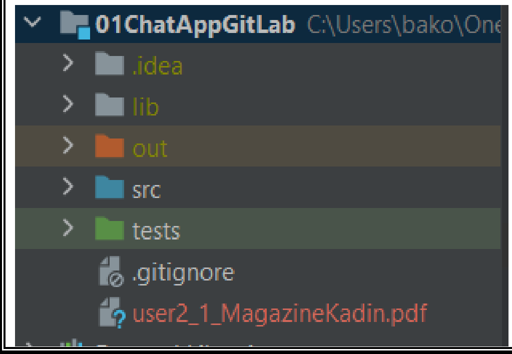
2	User1 types the receiver name  `Send to: user2`	Then user1 is prompted to enter the filename.  `File name:`	Pass

3	User1 types the filename.
	
	`Send to: user2`
	`File name: MagazineKadin.pdf`
	`File Size: 11.46 MB`
		User1 will receive a confirm message.
		
		`Sent`
		User2 will receive a notification.
		
		`user1 send you a new file`
			Pass

4	User2 types option 6 from the main menu	User2 sees the received file with the name of the file sender.
		
		`6`
		`Received files:`
		`1. File MagazineKadin.pdf from user1`
		`0. Close.`
		
		`Choice:`
			Pass

5	User2 selects option 1 from Received files menu
	
	`Received files:`
	`1. File MagazineKadin.pdf from user1`
	`0. Close.`
	
	`Choice: 1`
		User2 sees accept or decline menu
		
		`Choice: 1`
		`Sender: user1`
		`File: MagazineKadin.pdf`
		
		`Type (y) to Accept and (n) to Decline`
		`Accept: (y)`
		`Decline: (n)`
		`Your choice:`
			Pass

6	User2 types (y) to accept the file.
	
	`Sender: user1`
	`File: MagazineKadin.pdf`
	
	`Type (y) to Accept and (n) to Decline`
	`Accept: (y)`
	`Decline: (n)`
	`Your choice: y`
		User2 sees the confirm message and notified that the download has began and once the file is downloaded, user2 will be notified that the download has been done successfully.
		
		`Downloading the file...`
		`File has been downloaded successfully.`
		
		User1 will be notified that user2 accepted the transfer. And once the file is sent, user1 will be notified.
		
		`user2 accepted MagazineKadin.pdf file.`
		` Sending file now...`
		`File has been sent.`
			Pass

Postconditions
User1 have sent and user2 received the file successfully.  01ChatAppGitLab C:\Users\bako\One .idea lib out src tests .gitignore user2_1_MagazineKadin.pdf

11th Test

Test Case No.	TC10_11
Name	Transmit Large TIFF Image (Memory Test)
Short description	Clients successfully sends a large TIF image file (nearly 10GB size > available memory) between two clients.
System	File transfer
Requirement No.	RQ-U302

Preconditions
1. Both User (sender) and (receiver) needs to be logged in. 2. The Tif image with approximately 10GB is available.

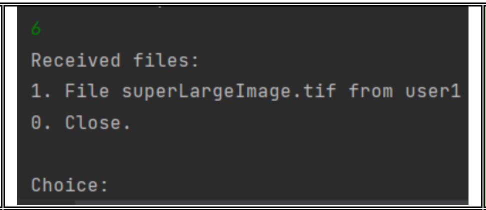
Steps	Action	Expected outcome	Pass/Fail

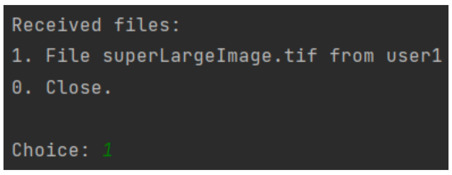
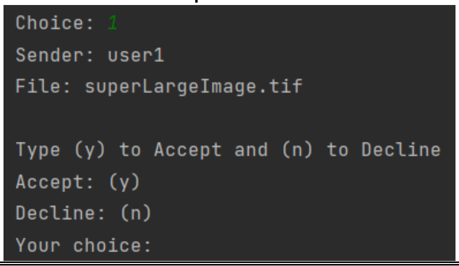
1	User1 types option 5 from the main menu
	`Enter your choice (1-7):`
	`5`
		User1 is prompted to enter the receiver name.
		`Send to:`
			Pass

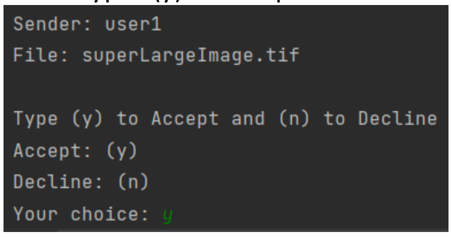
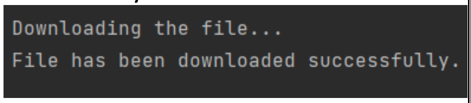
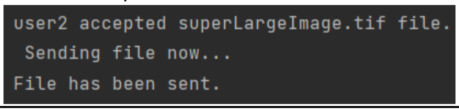
2	User1 types the receiver name `Send to: user2`	Then user1 is prompted to enter the filename. `File name:`	Pass

3	User1 types the filename.
	`Send to: user2`
	`File name: superLargeImage.tif`
	`File Size: 9976.76 MB`
		User1 will receive a confirm message.
		`Sent`
		User2 will receive a notification.
		`user1 send you a new file`
			Pass

4	User2 types option 6 from the main menu	User2 sees the received file with the name of the file sender.	Pass

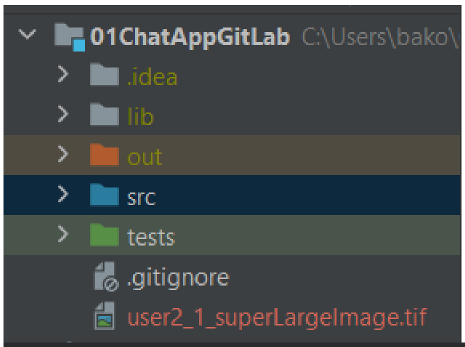
		
		`6`
		`Received files:`
		`1. File superLargeImage.tif from user1`
		`0. Close.`
		
		`Choice:`	

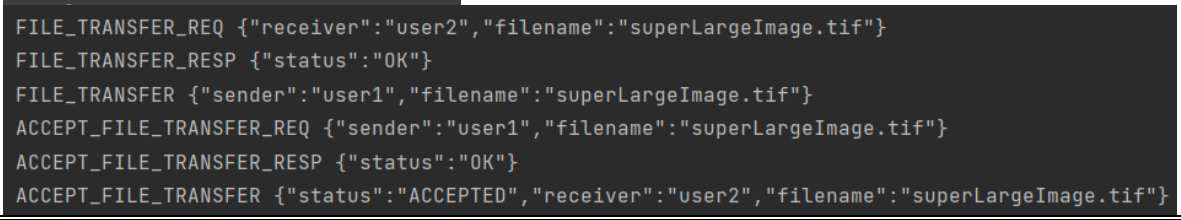
5	User2 selects option 1 from Received files menu
	
	`Received files:`
	`1. File superLargeImage.tif from user1`
	`0. Close.`
	
	`Choice: 1`
		User2 sees accept or decline menu
		
		`Choice: 1`
		`Sender: user1`
		`File: superLargeImage.tif`
		
		`Type (y) to Accept and (n) to Decline`
		`Accept: (y)`
		`Decline: (n)`
		`Your choice:`
			Pass

6	User2 types (y) to accept the file.
	
	`Sender: user1`
	`File: superLargeImage.tif`
	
	`Type (y) to Accept and (n) to Decline`
	`Accept: (y)`
	`Decline: (n)`
	`Your choice: y`
		User2 sees the confirm message and notified that the download has began and once the file is downloaded, user2 will be notified that the download has been done successfully.
		
		`Downloading the file...`
		`File has been downloaded successfully.`
		
		User1 will be notified that user2 accepted the transfer. And once the file is sent, user1 will be notified.
		
		`user2 accepted superLargeImage.tif file.`
		` Sending file now...`
		`File has been sent.`
			Pass

Postcondition

User1 have sent and user2 received the file successfully.





```
FILE_TRANSFER_REQ {"receiver":"user2","filename":"superLargeImage.tif"}
FILE_TRANSFER_RESP {"status":"OK"}
FILE_TRANSFER {"sender":"user1","filename":"superLargeImage.tif"}
ACCEPT_FILE_TRANSFER_REQ {"sender":"user1","filename":"superLargeImage.tif"}
ACCEPT_FILE_TRANSFER_RESP {"status":"OK"}
ACCEPT_FILE_TRANSFER {"status":"ACCEPTED","receiver":"user2","filename":"superLargeImage.tif"}
```

Reflection

Throughout the implementation process of the chat application, I encountered many challenges that gave me valuable lessons. In this reflection, I will delve into two particularly impactful experiences, sharing my realizations, outlining the key actions I took to overcome these challenges, and what my key steps will be in the future. To structure my insights, I have used the **STARR** method.

Adapting new approaches

- **S – Situation:**

During the implementation of the file transfer, I faced a challenge related to sending the username from the server to the client, because I would have used that username for the file transfer. Initially, I included the username in the login OK status response and that was working but it was not okay because based on the login protocols, the login response, should only have had the OK status., and this caused issues with the user acceptance tests, and to make these tests pass, I had to make changes to the login tests by sending the username along with the OK status and the big problem with this was; it was not allowed to make changes to the existing tests! All of them was supposed to be passing without changing anything! Which I only knew about this in the lessons of week-8, because that was the only two lessons that I attended unfortunately.

- **T – Task:**

So my task was to find an alternative way to send the username without compromising the existing login test protocols.

- **A – Action:**

Gladly, the teacher suggested a more efficient approach that I could use to make the file transfer work and with his approach, I wouldn't have needed to send the username with the login OK status. His idea was that I could use bytes, by separating them into three parts: one for the command, some for the name, and the rest for the file. Despite dedicating two days implementing this approach, I couldn't succeed. Therefore, as a time-sensitive alternative, I decided to use a setter method to set the current username on the client side in the handle joined method. Because in the joined method, I am sending the username anyway based on the protocols. Also, the handle joined method gets executed with the successful login method. This approach allowed me to send the username without changing the login response protocol.

- **R – Result:**

Initial approach: The initial approach of sending the username with the login OK status response caused problem during user acceptance tests.

Teacher's Guidance: The teacher showed me a more efficient byte-based solution, but due to time constraints, I couldn't implement it.

Alternative solution: I used the handle joined method to set the current username on the client side, which proved to be a faster but acceptable solution, and it ensured that the file transfer works without changing any existing tests as well.

- **R – Reflection:**

Upon reflection, this experience has shown me how crucial it is to be adaptable in the world of software development and how it can improve my problem-solving skills. Additionally, I realized keeping up with the classes, and attending regularly could have saved me a lot of time and led to more efficient solutions through more discussions with the teacher.

Moving forward, I am committed to explore different solutions early on, seeking insights, and confirming my approaches before diving into implementation.

Importance of Consistency and Testing

- **S – Situation:**

While working on the chat application, I ran into a big problem. I decided to finish adding all the features before doing any testing. This turned out to be a bad move because when I finally tested everything, none of the tests worked. The issue was that my protocols didn't match what the tests expected. I also realized I hadn't been using JSON format consistently. I only used it when showing protocols on the server side, but the server wasn't handling JSON-formatted responses to or from the client.

- **T – Task:**

I had two things to fix: first, to make my protocols match the expected ones in the tests without changing the tests, and second, to consistently use JSON format for communication between both the client and server side.

- **A – Action:**

I made a plan to start from the beginning. This time, I focused on one feature at a time using JSON format. After implementing each feature, I tested it immediately.

- **R – Result:**

Starting fresh and focusing on one feature at a time with immediate testing improved the development process. Gladly I was able to make every test pass and made sure all the communication between client and the server is JSON-formatted. Additionally, as I was finishing with each feature and ensuring that its test is working, it gave me more confidence in the coming features that I implemented because I no longer had the fear whether what I have done is correct and its test will pass not.

- **R – Reflection:**

Upon reflection, I have come to realize the crucial role of testing in ensuring both peace of mind and effective time management throughout the development process. Delaying tests is likely to lead to setbacks, focusing on testing continuously as I go along implementing each feature, rather than just waiting until the end. Additionally, it is clear to me now how the consistent use of JSON format can improve the quality and is better for readability and understanding the implementation.

Moving forward, I am committed to focus more on testing and prioritize it in future projects, and reading documentation thoroughly and make sure I understand my requirements before I start with implementation. I realize the need to ask questions when things are unclear, rather than just making assumptions, because for the JSON format issue, I now understand that my failure to seek clarification led to an incorrect approach. Finally, questioning assumptions and testing my features without delaying will be my strategy for a smoother development process and to avoid such pitfalls in the future.

Additionally, another thing that is worth noting is how I handled displaying online users. The way I implemented it, if there are no users online, I treated it as an error. But when looking back at this, I realize a no user online scenario shouldn't be considered as an error but rather a valid scenario. Unfortunately, by the time I realized this, it was too late, because the I have already finished the implementation for both server and client sides, along with the tests. Given the progress made and the time I had left, I decided not to make changes to avoid any sudden errors.

In conclusion, learning to adapt when I face obstacles and realizing the importance of thorough testing were game-changers. Furthermore, questioning assumptions and seeking clarification early on can have huge impact. Finally, In future projects, I'll keep these lessons in mind, aiming for a smoother development process because development is not just about coding; it's about understanding, being flexible, and making sure I'm on the right track from the start.