The Server and User Nodes use a callback mechanism to communicate with each other using ProjectLib messaging services. When a commit is requested on the Server side, the Server will notify all involved user nodes and ask for their decision, and each user node will then check if the collage is desired and send back their decision. The communication uses a two-phase commit structure: when a request to collage is made, the start commit function will be called on the Server side. The server will start the preparation stage where it sends a request to commit to all the user nodes. The message will include the collage file, collage contents, and source files from that specific user. And a class Commit_message is defined to store the information, which is then serialized by ObjectOutputStream and ByteArrayOutputStream. Then at each user node, the user will see first if any of the source files might be used in any other collages or does not exist and then check the content of the collage. If the collage satisfies all the requirements, the user node will agree to the commit and lock all the source files so they cannot be used in other collages, and send back the decision to the server. Otherwise, no action will be taken on the user side. Once the server receives replies from all users or reaches the timeout threshold, if everyone voted yes for the collage, the server will then commit the collage and broadcast the decision to all user nodes. Otherwise, the server will broadcast the abort decision to all user nodes. In either case, to ensure the result is propagated to all user nodes, the server will keep sending the decision to the user until all users have replied with an ACK. This will be done by keeping a list of the user nodes involved in the collage and removing each user when an ACK is received. Then at the end of the timeout threshold, the server will re-broadcast the decision to the user nodes remaining on the list, and repeat the process until the list is empty.

A message is considered lost if a response hasn't been received after 3000ms of sending the message. If a message is lost in the preparation stage, either the server asks user nodes or user nodes to send back the reply, then the user is considered to vote to abort in this case. After the server handles the commit or abort operation after the preparation stage, the server will then send the result to all the user nodes and expect an ACK byte from the user node. If a message is lost in this process (whether sending the result or ACK replies), then the server will re-transmit the same message and restart the timer to ensure that the result is successfully propagated to the user nodes.

Write-ahead logging is used for the recovery mechanism and a separate log file is kept for each node. Transactions and results of operations are logged both in the server and user nodes to ensure successful recovery from node failures, and each time a new log is added fsync will be called to ensure that the logs are written in the disk. Specifically, on the server side, events that need to be logged in

1. When the server attempted to send the request to all the user nodes
2. When the server is able to make a decision (commit or abort) and perform the commit operations
3. When the server finishes receiving the ACK byte from all the user nodes, this will mark the finish point of a transaction

In the case where a server node failure occurs, the logged file will be scanned to find any transactions that haven't finished yet and re-perform the necessary operations. For each log,

the transaction ID, event, collage filename, and hashmap that maps each user node ID to a list of the corresponding source files are recorded. So, based on the three types of log, each transaction can be in three stages at the point of recovery:

1. Since the server crashed before committing, we can just abort the commit send the decision to all the user nodes, and wait to receive all the ACK bytes (retransmit if necessary)
2. Based on the decision it logged, send the decision to the user nodes and wait for their ACK bytes (retransmit if necessary)
3. The transaction will not modify the committed state, so nothing needs to be done

And for the user node side, events that need to be logged include:

1. When the user agrees or disagrees with the commit request from the server
2. When the user receives the final decision from the server
3. When the user finishes the transaction after sending the ACK

For each log, the event, collage file name, and source files are recorded so the node has all the necessary information during recovery.
Then in the case where a user node failure occurs. Based on the three types of logs, each request can also be in three stages at the point of recovery:

1. If the node crashes after the user agrees, the user node will relock the source files. And if the user disagrees, no recovery needs to be done.
2. If the node crashes after receiving the decision, then it will attempt to delete (if the decision is committed)and unlock the source files again
3. If the node crashes after sending the ACK, then this transaction is marked as complete. And no recovery action needs to be performed.

The timeout threshold I use to indicate a packet is lost is 3000ms since we can assume that a message that is not lost is guaranteed to arrive within 3 seconds of being sent. So if we are unable to receive a reply in 3000ms, this means the packet is lost somewhere in the transmission. And we can consider it as a vote abort if we are in the preparation stage, or resend the message if we are in the decision distribution stage.

For every recovery and commit transaction, a new thread will be created on the server side, and each thread will have its blocking queue for receiving messages. The while loop in the main function is used to receive the message from all the user nodes and then insert the message into the corresponding message queue for further processing.