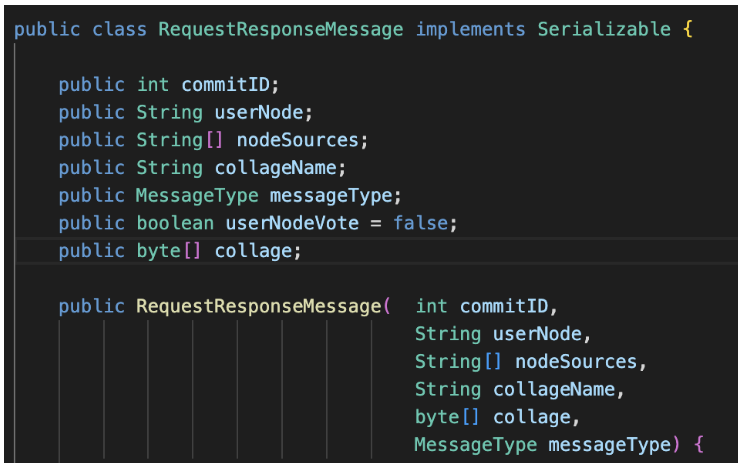
Project 4: Two-phase Commit for Group Photo Collage

Design Document

Hamza Khalid (hkhalid)

Communication protocol between the Server and the User Nodes

The project implements a two-phase commit protocol. After the commit is started the server asks all the participating user nodes for their votes. The user nodes save the images that are part of an ongoing commit in a busyFiles Map to make sure no other commits use these images until the current commit finishes. After getting the votes from all the nodes, the server notifies the nodes about the voting decision and waits for their acknowledgment. If the server doesn’t get the acknowledgment back, it sends the voting decision again indefinitely until it receives the acknowledgments from all the user nodes. The server saves the collage on the server in case of a successful commit and deletes the images associated with that collage to make sure no one uses those images ever again. The Server and the User Nodes communicate with each other in a specific message format shown below.



```
public class RequestResponseMessage implements Serializable {

    public int commitID;
    public String userNode;
    public String[] nodeSources;
    public String collageName;
    public MessageType messageType;
    public boolean userNodeVote = false;
    public byte[] collage;

    public RequestResponseMessage(  int commitID,
                                    String userNode,
                                    String[] nodeSources,
                                    String collageName,
                                    byte[] collage,
                                    MessageType messageType) {
```

Serialization is used before transmitting the message and deserialization is used after receiving the message.

Mechanism to handle lost messages

○ The recovery mechanism for the Server

A new log file is created at the start of every commit. This log file logs contains the collage name and commit id at the first line. The second line of the log file contains the image sources. And, the third line contains the voting decision (ABORT or COMMIT) of the commit before a crash. After the crash, the server looks for all the log files named commit_id.tmp. The server then reads all these files and starts new recovery threads for every commit. If the last voting decision before the crash is ABORT or COMMIT, the server distributes this voting decision to all the nodes and waits for their acknowledgments. After the crash recovery or if everything goes as expected (no crash), the server finishes the commit thread and deletes the log file.

○ *The recovery mechanism for the User Nodes*

A log file is created at the start of every user node. All the files/images that are currently in use are logged into this log file. When the user node recovers after a crash, it first looks for the log file, reads it, populates the busy files (or files in use) data structure in the user node, and deletes this log file. A new log file is generated after the crash recovery.

○ Timeout thresholds

The timeout threshold used for the server to wait for a response from user nodes (after asking them to vote) is 6000 ms which is 2 times 3000 ms. A user node is guaranteed to give a response within 3000 ms. We multiplied 3000 by 2 because of the roundtrip.

The timeout threshold used for the server to wait for the acknowledgment from the user nodes after distributing the voting decisions is 3000 ms. After this time, the server distributes the voting decision again.