Go Assignment

Description of features of the application
Features:

0. Sending Messages

Messages are stored in Linked List and Stack . This provides us an efficient message retrieval and deleting functionalities

0. See all the messages in the chat

We can see the whole chat and print out the messages including the Sender name and message itself

0. Filter messages based on Sender Name
0. Filter based on specific keyword/part of keyword
0. Delete the latest message

Description of how data structures are being applied and their suitability
Data structures:

0. LinkedList

a. Dynamic Size: Linked List can easily expand and shrink based on the messages requirement
b. Ease of traversal: It is easy to traverse through the list to get to the Node/ messages that I need. Using this functionality I was able to create search features
c. Ease of Insertion/Deletion: It is also easy to insert/delete as each node is linked to the next, we can move and add or delete from wherever we want to. With the help of pointers we can easily choose a location we need and delete. We do not have insertion in the middle of the list as a chat does not have that functionality. We can only insert at the bottom of the list.

0. Stack

a. The main reason I have implemented a stack is because it is very efficient to remove the latest node/message from a stack. This is completely in line with a messaging app as I can easily find the latest message as it is at the top.
b. Following the LIFO principle helped me to implement the undo feature easily
c. Future implementation: I could also implement redo feature where a user can cancel his undo action. For which I can use this stack again easily

Description of the data and format in the various files
The app is divided by functionalities.
There are a few files

0. main.go —> Where the programme starts running. Global variables, instances of linkedList, stacks, seeding of initial data and running messaging

interface are all done here

0. listFunctions.go—> This is where majority of the algorithms are stored. We have functions to seeMessage(), addMessage(),removeNode(), searchName(),searchWord(). All these functions are specific to lists
0. stackFunctions.go—> Functions for stacks such as pop()and addMessage(). Only applicable to stack
0. combinedFunc.go—> These has functions which are implemented on both stacks and linkedList. Functions such as adding a message to the structures, creating a node or unending messages from both.
0. messagingInterface.go —> This is where the actual interface runs and brings together all the functions that we have created. It continuously runs on the terminal until the user decides to cancel
0. structs.go —> Type definitions for messages, nodes, list, stack are stored here
0.

MessagingList and MessagingStack created in main are passed around and majority of the features are methods on these two structures. The core element of this app is the message struct which holds senderName, message and uniqueID. It is then passed to a Node which is added to these two structures

Description of various error handling and concurrency mechanisms incorporated

0. Error Handling/Panic: In messaging interface as well as all functions. We return an error to catch any problems that might occur as well as user recover to catch the panics that we have created and send incase of issues. Such as when trying to pop when there is nothing to pop or when a chat is empty and trying to display the messages
0. Concurrency: I have used Mutex to lock the key feature which is adding messages to the data structures and also go routines so that the initial seeding can be done concurrently. I felt that locking the most important part and identifying gives me a good use of mutex.

How to run:
Just key in
./goAssignment
and press enter