

MSCS-632: Residency Project: Deliverable 3 Report: Simple Chat Application: Final Report

Shabnam Shaikh, Sakchham Sangroula, Sandesh Pokharel, Nihar Turumelle, Romika Souda

University of the Cumberlands

MSCS-632 Advanced Programming Languages

Dr. Vanessa Cooper

April 06, 2025

1. Introduction

This report presents a detailed comparison of the final implementation of our CLI-based Chat Application using two different programming languages: **Rust** and **Go**. The project was designed as part of the MSCS-632 Advanced Programming Languages course and aims to showcase how different languages approach the same application problem using their own syntax rules, concurrency models, and language-specific paradigms.

Throughout the development, our goal was to maintain functional parity between the two versions while also exploring the unique idioms and strengths of each language. The Rust version focuses on safety, strict ownership, and asynchronous message handling using tokio (Jonsson et al., 2020). On the other hand, the Go version leverages lightweight concurrency using goroutines and channels, offering a more traditional menu-driven approach to CLI interactions (Google, 2023).

GitHub Link: <https://github.com/sanspokharel26677/MSCS-632-Residency-Project#>

This report discusses the differences, challenges, and developer experiences between the two implementations, using code snippets and screenshots to support our evaluation.

 **Screenshot :** Application outputs from both Rust and Go versions running in the terminal.

```

sandesh@sandesh-Inspiron-7373:~/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/HSCS-632-Residency-Project/rust_chat_app$ go run *
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: add user
Enter a username for the new user: sans
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: sak
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: add user
Invalid response, please enter send message / add user / filter user / filter message
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: add user
Enter a username for the new user: sakc
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: sen
d message
Enter sender username: sans
Enter recipient username: sakc
Enter message (or 'exit' to quit): Hi sakc
15:37:18 | sans => sakc: Hi sakc
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: sen
d message
Enter sender username: sakc
Enter recipient username: sans
Enter message (or 'exit' to quit): hello sans
15:38:27 | sakc -> sans: hello sans
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: fil
ter message
Enter the keyword to filter by: hello
--- Messages containing 'hello' ---
15:38:27 | sakc -> sans: hello sans
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: ■
```

✖ Technical and Functional

```

sandesh@sandesh-Inspiron-7373:~/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/HSCS-632-Residency-Project/rust_chat_app$ cd ./rust_chat_app/
sandesh@sandesh-Inspiron-7373:~/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/HSCS-632-Residency-Project/rust_chat_app$ ls
chat.chat
sandesh@sandesh-Inspiron-7373:~/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/HSCS-632-Residency-Project/rust_chat_app$ cd rust_chat_app/
sandesh@sandesh-Inspiron-7373:~/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/HSCS-632-Residency-Project/rust_chat_app$ ls
Cargo.lock  Cargo.toml  .git  .gitignore
sandesh@sandesh-Inspiron-7373:~/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/HSCS-632-Residency-Project/rust_chat_app$ cd src/
sandesh@sandesh-Inspiron-7373:~/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/HSCS-632-Residency-Project/rust_chat_app$ rust_chat_app/vcs$ ls
main.rs
sandesh@sandesh-Inspiron-7373:~/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/HSCS-632-Residency-Project/rust_chat_app$ rust_chat_app/vcs$ cargo run
   Finished dev profile [unoptimized + debugInfo] target(s) in 0.27s
     Running `./rust_chat_app`
=====
Welcome to the Rust Chat App!
=====
How many users would you like to create? 2
Enter username for User 1: Sandesh
Enter username for User 2: Shab
=====
chat is starting! Type your messages one by one.
Type 'exit' to leave the chat as a user.

[ Active users: ["Sandesh", "Shab"]
Who wants to send a message? Sandesh
[Sandesh] Enter your message: Hi Shab
[15:20:37][Sandesh]: Hi Shab

[ Active users: ["Sandesh", "Shab"]
Who wants to send a message? Shab
[Shab] Enter your message: Hello Sandesh
[15:20:51][Shab]: Hello Sandesh

[ Active users: ["Sandesh", "Shab"]
Who wants to send a message? Sandesh
[Sandesh] Enter your message: How are you?
[15:23:13][Sandesh]: How are you?
```

2. Application Summary & Final Features

The final version of the application in both languages includes the following core features:

- Dynamic user creation
- CLI-based message input and output
- Message history with timestamps
- Per-user message tracking
- Message filtering by user and keyword
- User exit functionality
- End-of-chat summary report

Despite differences in syntax and control flow, both versions achieve identical output expectations and match the feature checklist outlined in the project requirements.

Rust

- Turn-based interaction
- User rotation for message input
- Message queue (VecDeque)
- Shared server state using Arc<Mutex<>>

Go

- Menu-based interaction with user commands
- Real-time message routing using goroutines
- Per-user channels for message receipt
- Message history stored globally for filtering

 **Screenshot:** Rust and Go feature menus or CLI flows

```

sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_L... sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/go_chat_app$ cd .. ./rust_chat_app/
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/rust_chat_app$ ls
Cargo.toml  lib.rs
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/rust_chat_app/rust_chat_app$ cd rust_chat_app/
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/rust_chat_app/rust_chat_app$ ls
Cargo.lock  lib.rs
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/rust_chat_app/rust_chat_app$ cd src/
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/rust_chat_app/rust_chat_app/src$ ls
main.rs
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/rust_chat_app/rust_chat_app/src$ cargo run
    Finished 'dev' profile [unoptimized + debuginfo] target(s) in 0.27s
     Running `'/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/rust_chat_app/rust_chat_app/target/debug/rust_chat_app'`
=====
👋 Welcome to the Rust Chat App!
=====
How many users would you like to create? 2
Enter username for User 1: Sandesh
Enter username for User 2: Shab
=====
📝 Chat is starting! Type your messages one by one.
Type '/exit' to leave the chat as a user.
=====
🕒 Active users: ["Sandesh", "Shab"]
Who wants to send a message? Sandesh
[Sandesh] Enter your message: Hi Shab
[15:20:37][Sandesh]: Hi Shab
=====
🕒 Active users: ["Sandesh", "Shab"]
Who wants to send a message? Shab
[Shab] Enter your message: Hello Sandesh
[15:20:51][Shab]: Hello Sandesh
=====
🕒 Active users: ["Sandesh", "Shab"]
Who wants to send a message?

```

```

sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/go_chat_app$ go run *
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/go_chat_app$ go run *
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: add user
Enter a username for the new user: sans
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: add user
Enter a username for the new user: ram
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: send message
Enter sender username: sans
Enter recipient username: ram
Enter message (or 'exit' to quit): hi ram
16:11:50 | sans -> ram: hi ram
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: send message
Enter sender username: ram
Enter recipient username: sans
Enter message (or 'exit' to quit): hi sans
16:12:04 | ram -> sans: hi sans
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: filter message
Enter the keyword to filter by: hi
---
--- Messages containing 'hi' ---
16:11:50 | sans -> ram: hi ram
16:12:04 | ram -> sans: hi sans
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: filter user
Enter the username to filter messages by: sans
16:11:50 | sans -> ram: hi ram
16:12:04 | ram -> sans: hi sans
What would you like to do? (send message / add user / filter user / filter message) [Type 'exit' to quit]: exit
Exiting the chat application.
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Advanced_Programming_Languages/MSCS-632-Residency-Project/go_chat_app$ 

```

3. Code Architecture & Design Patterns

Rust Architecture

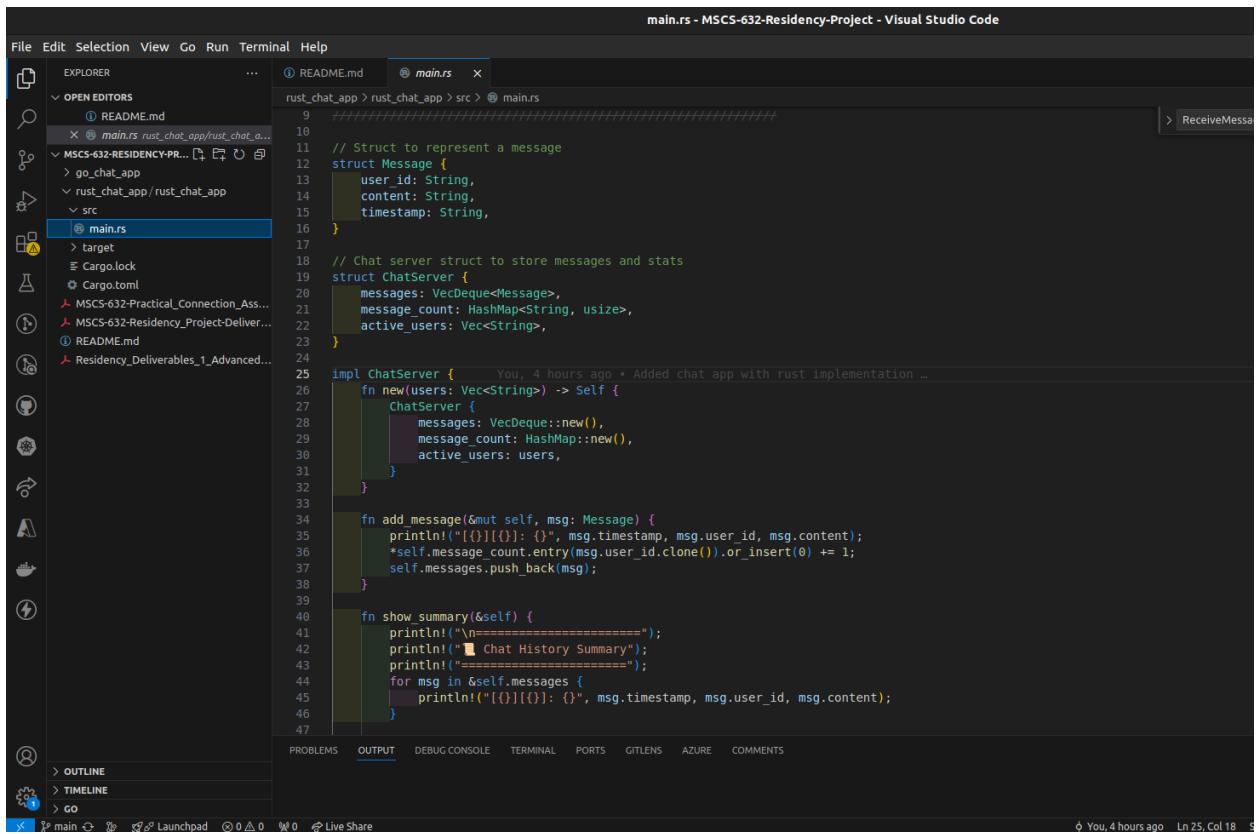
Rust was implemented in a single file (main.rs) using multiple structs and a shared server instance. The key architectural decision was to use Arc<Mutex<>> to allow concurrent, thread-safe access to the shared ChatServer.

```
let server = Arc::new(Mutex::new(ChatServer::new()));
```

The server holds all active users and message history. Each user is managed by an async fn task and takes turns sending messages through the terminal.

```
async fn start_user(id: &str, server: Arc<Mutex<ChatServer>>) {  
    // CLI interaction loop  
}
```

💡 Screenshot Placeholder: Rust start_user() function and shared state setup



The screenshot shows the Visual Studio Code interface with the main.rs file open in the editor. The code defines a struct Message and a struct ChatServer. The ChatServer struct contains a VecDeque for messages, a HashMap for message counts, and a Vec for active users. It has methods to add messages and show a summary. The code is annotated with comments explaining its purpose. The Explorer sidebar shows the project structure, including README.md, Cargo.lock, and Cargo.toml files. The bottom status bar indicates the code was last modified 4 hours ago at line 25, column 18.

```
main.rs - MSCS-632-Residency-Project - Visual Studio Code  
File Edit Selection View Go Run Terminal Help  
EXPLORER ... README.md @ main.rs x  
OPEN EDITORS  
rust_chat_app > rust_chat_app > src > main.rs  
9 // Struct to represent a message  
10 11 struct Message {  
12     user_id: String,  
13     content: String,  
14     timestamp: String,  
15 }  
16  
17 // Chat server struct to store messages and stats  
18 struct ChatServer {  
19     messages: VecDeque<Message>,  
20     message_count: HashMap<String, usize>,  
21     active_users: Vec<String>,  
22 }  
23  
24 impl ChatServer {  
    fn new(users: Vec<String>) -> Self {  
        ChatServer {  
            messages: VecDeque::new(),  
            message_count: HashMap::new(),  
            active_users: users,  
        }  
    }  
25  
26    fn add_message(&mut self, msg: Message) {  
27        println!("{}({}): {}", msg.timestamp, msg.user_id, msg.content);  
28        *self.message_count.entry(msg.user_id.clone()).or_insert(0) += 1;  
29        self.messages.push_back(msg);  
30    }  
31  
32  
33    fn show_summary(&self) {  
34        println!("{}\n=====");  
35        println!("| Chat History Summary");  
36        println!("| =====");  
37        for msg in &self.messages {  
38            println!("| ({})({}): {}", msg.timestamp, msg.user_id, msg.content);  
39        }  
40    }  
41  
42  
43  
44  
45  
46  
47
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS AZURE COMMENTS

You, 4 hours ago Ln 25, Col 18

Go Architecture

The Go version uses multiple source files (main.go, chat.go, and user.go) to separate concerns.

Each user has a Received channel and a goroutine that continuously listens for incoming messages.

```
go user.ReceiveMessages()
```

Messages are passed through the ChatServer which acts as the router.

```
type ChatServer struct {  
    Users map[string]*User  
    Messages []Message  
}
```

💡 Screenshot: Go goroutine handling and message routing in chat.go and user.go

```
chat.go - MSCS-632-Residency-Project - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS MSCS-632-RESIDENCY-PR... go_chat_app
go_chat_app > chat.go
chat.go
main.go
user.go
rust_chat_app
MSCS-632-Practical_Connection_Ass...
MSCS-632-Residency_Project-Deliver...
README.md
Residency_Deliverables_1_Advanced...
chat.go
< 100% 100% Launchpad 0 △ 0 Live Share
chat.go
go.chat_app > chat.go stores and displays messages, and supports ...
// filtering by user and keyword.
//
12 // filtering by user and keyword.
13 // filtering by user and keyword.
14 //
15 // Message holds the content sent by a user along with metadata.
16 type Message struct {
17     UserIDFrom string    // ID of the user who sent the message
18     UserIDTo   string    // ID of the user to whom the message is directed
19     Content    string    // Actual message text
20     Time       time.Time // Timestamp of when message was sent
21 }
22
23 // ChatServer handles storing and displaying messages.
24 type ChatServer struct {
25     Users    map[string]*User // Map of users to allow user management
26     Messages []Message      // Slice to store chat history
27 }
28
29 // FilterByUser prints all messages from a specific user.
30 func (c *ChatServer) FilterByUser(userID string) {
31     // Check if the user exists in the chat server
32     if _, exists := c.Users(userID); !exists {
33         fmt.Printf("User with ID %s does not exist.\n", userID)
34         return // Exit if the user does not exist
35     }
36     var sentMessages = c.Users(userID).SentMessages
37     for _, msg := range sentMessages {
38         if msg.UserIDFrom == userID || msg.UserIDTo == userID {
39             fmt.Printf("%s | %s -> %s: %s\n", msg.Time.Format("15:04:05"), msg.UserIDFrom, msg.UserIDTo, msg.Content)
40         }
41     }
42
43     var receivedMessages = c.Users(userID).ReceivedMessages
44     for _, msg := range receivedMessages {
45         if msg.UserIDFrom == userID || msg.UserIDTo == userID {
46             fmt.Printf("%s | %s -> %s: %s\n", msg.Time.Format("15:04:05"), msg.UserIDFrom, msg.UserIDTo, msg.Content)
47         }
48     }
49 }
```

The screenshot shows the Visual Studio Code interface with the file `chat.go` open in the editor. The code implements a `ChatServer` struct with methods for filtering messages by user ID and printing them. It uses goroutines and channels for message routing. The code editor has syntax highlighting for Go, and the status bar at the bottom shows the current line and column (Ln 60, Col 1) and tab size (Tab Size: 4).

```

user.go - MSCS-632-Residency-Project - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER OPEN EDITORS
  < user.go > user.go
  go_chat_app > user.go
  MSCS-632-RESIDENCY-PR...
  go_chat_app
  |> chat.go
  |> main.go
  |> user.go
  > rust_chat_app
  MSCS-632-Practical_Connection_Ass...
  MSCS-632-Residency_Project-Deliver...
  README.md
  Residency_Deliverables_1_Advanced...

Sakchham Sangroula, 7 hours ago | 2 authors (Sakchham Sangroula and one other)
1 package main
2
3 import (
4     "fmt"
5 )
6
7 // User represents a chat participant with a unique ID and
8 // a channel to send messages to the main chat server.
9 type User struct {
10     ID           string      // Unique identifier for the user
11     Input        chan Message // Channel used to receive messages
12     SentMessages []Message   // Store messages sent by this user, if needed
13     ReceivedMessages []Message // Store messages received by this user, if needed
14 }
15
16 func AddUser(users map[string]*User, id string, input chan Message) (*User, error) {
17     if _, exists := users[id]; exists {
18         return nil, fmt.Errorf("User with ID %s already exists", id)
19     }
20
21     user := &User{
22         ID:           id,
23         Input:        input,
24         SentMessages: []Message{},
25         ReceivedMessages: []Message{},
26     }
27     users[id] = user
28     return user, nil
29 }
30
31 func RemoveUser(users map[string]*User, id string) {
32     delete(users, id)
33 }
34
35 func GetUser(users map[string]*User, id string) (*User, bool) {
36     user, exists := users[id]
37

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS AZURE COMMENTS

Sakchham Sangroula, 7 hours ago | Ln 11, Col 70 Tab Size: 4

4. Language-Specific Features & Syntax Differences

Memory Safety and Ownership (Rust)

Rust's strict ownership system required careful handling of shared data. All shared state had to be wrapped in `Arc<Mutex<>>`, and all updates to shared resources required explicit locking (Klabnik & Nichols, 2019).

Concurrency Models

- **Rust:** Uses async functions, the tokio runtime, and `.await` for non-blocking I/O.

- **Go:** Uses goroutines (lightweight threads) and channels to send/receive data concurrently (Google, 2023).

Data Structures

Rust's collections like VecDeque and HashMap are strongly typed and require explicit borrowing and mutability.

Go uses slices, maps, and basic structs with simpler syntax.

Syntax & Type System

- Rust has a more verbose, strict syntax with an emphasis on safety.
- Go uses a minimalist, beginner-friendly syntax that focuses on simplicity and readability.

💡 Screenshot Placeholder: Rust locking and async blocks vs Go goroutines and channels

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** main.rs - MSCS-632-Residency-Project - Visual Studio Code
- File Explorer:** Shows the project structure:
 - rust_chat_app > rust_chat_app > src > @ main.rs
 - rust_chat_app > go_chat_app
 - rust_chat_app /rust_chat_app
 - src
 - @ main.rs (selected)
 - target
 - debug
 - .rustc_info.json
 - CACHEDIR.TAG
 - Cargo.lock
 - Cargo.toml
 - MSCS-632-Practical_Connection_Ass...
 - MSCS-632-Residency_Project-Deliver...
 - README.md
 - Residency_Deliverables_1_Advanced...
- Editor:** The main.rs file is open, showing Rust code for a chat application. The code includes functions for reading input, handling user creation, and a main loop for active users. A specific line of code is highlighted: `temp.trim().to_string()`. The code uses various Rust features like `let mut` for mutable references and `Vec::new()` for creating vectors.
- Bottom Status Bar:** Shows the current file is main.rs, has 0 changes, and was last modified 4 hours ago at line 25, column 18.

chat.go - MSCS-632-Residency-Project - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER OPEN EDITORS MSCS-632-RESIDENCY-PROJECT

user.go chat.go main.go

```
go_chat_app > chat.go
1 package main    You, yesterday * Go implementation- initial commit
2
3 import (
4     "fmt"
5     "strings"
6     "time"
7 )
8
9
10 // This file defines the Message struct and ChatServer. //
11 // ChatServer stores and displays messages, and supports //
12 // filtering by user and keyword. //
13
14
15 // Message holds the content sent by a user along with metadata.
16 type Message struct {
17     UserIDFrom string    // ID of the user who sent the message
18     UserIDTo   string    // ID of the user to whom the message is directed
19     Content    string    // Actual message text
20     Time       time.Time // Timestamp of when message was sent
21 }
22
23 // ChatServer handles storing and displaying messages.
24 type ChatServer struct {
25     Users map[string]*User // Map of users to allow user management
26     Messages []Message    // Slice to store chat history
27 }
28
29 // FilterByUser prints all messages from a specific user.
30 func (c *ChatServer) FilterByUser(userID string) {
31     // Check if the user exists in the chat server
32     if _, exists := c.Users(userID); !exists {
33         fmt.Println("User with ID %s does not exist.\n", userID)
34         return // Exit if the user does not exist
35     }
36     var sentMessages = c.Users(userID).SentMessages
37     for _, msg := range sentMessages {
38         if msg.UserIDFrom == userID || msg.UserIDTo == userID {
39             fmt.Printf("%s | %s -> %s: %s\n", msg.Time.Format("15:04:05"), msg.UserIDFrom, msg.UserIDTo, msg.Content)
40         }
41     }
42 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS AZURE COMMENTS

You, yesterday Ln 1, Col 4 (1 selected) Tab Size: 4

user.go - MSCS-632-Residency-Project - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER OPEN EDITORS MSCS-632-RESIDENCY-PROJECT

user.go

```
go_chat_app > user.go
Sakchham Sangroula, 7 hours ago | 2 authors (Sakchham Sangroula and one other)
1 package main
2
3 import (
4     "fmt"
5 )
6
7 // User represents a chat participant with a unique ID and
8 // a channel to send messages to the main chat server.
9 type User struct {
10     ID        string    // Unique identifier for the user
11     Input     chan Message // Channel used to receive messages
12     SentMessages []Message // Store messages sent by this user, if needed
13     ReceivedMessages []Message // Store messages received by this user, if needed
14 }
15
16 func AddUser(users map[string]*User, id string, input chan Message) (*User, error) {
17     if _, exists := users[id]; exists {
18         return nil, fmt.Errorf("User with ID %s already exists", id)
19     }
20
21     user := &User{
22         ID:        id,
23         Input:    input, // Input channel will be set when used in the chat server
24         SentMessages: []Message{}, // Initialize sent messages slice
25         ReceivedMessages: []Message{}, // Initialize received messages slice
26     }
27     users[id] = user
28     return user, nil
29 }
30
31 func RemoveUser(users map[string]*User, id string) {
32     delete(users, id)
33 }
34
35 func GetUser(users map[string]*User, id string) (*User, bool) {
36     user, exists := users[id]
37 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS AZURE COMMENTS

Sakchham Sangroula, 7 hours ago Ln 18, Col 19 (1 selected) Tab Size: 4

5. Challenges Faced During Implementation

Rust Challenges

- **Ownership and Borrowing:** Rust's strict ownership model required a deep understanding of when and how data is borrowed or moved. We faced frequent compiler errors when attempting to access shared mutable state without proper locking.
- **Async Context Issues:** Working with `tokio::spawn` meant we had to ensure all asynchronous tasks were `Send`. This led to errors when locking Mutexes across await points, forcing us to refactor parts of the code to isolate those accesses.
- **Verbose Error Handling:** While Rust's compiler is helpful, we had to explicitly handle every possible failure scenario such as locking errors, unwraps, and input handling, which increased code length and complexity.

Go Challenges

- **Channel Coordination:** Ensuring that every goroutine properly read from and wrote to channels without leaking or blocking required careful design. A missed range or `close()` would crash the app.
- **Message Flow Control:** Go lacks strict enforcement of type contracts, so message passing logic could silently fail if not thoroughly tested. We had to include manual nil-checks and extra validation.

- **Managing File Structure:** Since the Go implementation was split into main.go, chat.go, and user.go, it required more effort to keep track of imports, initialization sequences, and consistent naming across files.

Shared Challenges

- **CLI Responsiveness:** Building an interactive and intuitive terminal interface while processing input and output asynchronously (Rust) or concurrently (Go) was tricky. Ensuring the prompt flow stayed user-friendly was a repeated challenge.
- **Real-Time Filtering and Logging:** After each user exits, filtering messages by user and keyword needed us to manage message storage properly and loop through historical data structures accurately without mutating the live state.
- **Feature Parity Across Languages:** Maintaining consistent functionality across Rust and Go while respecting each language's idiomatic practices was a non-trivial balancing act throughout the project.

6. Performance, Readability, and Language Experience

Performance

- Rust: Fast and safe, but complex setup
- Go: Fast and simple, suited for rapid prototyping

Readability & Maintainability

- Rust: Enforces best practices with more effort
- Go: Easy for teams to follow and maintain

Developer Experience

- Rust: Rewarding but steep
- Go: Smooth and productive

7. Conclusion & Team Reflection

Both languages allowed us to meet all functional goals but offered different experiences. Rust emphasized memory safety and concurrency control, while Go encouraged faster iterations with fewer barriers. As a team, we gained insight into choosing the right language for the right job.

8. References

Google. (2023). *Effective Go: Concurrency*. https://go.dev/doc/effective_go#concurrency

Jonsson, C., Jonsson, L., & Contributors. (2020). *Asynchronous Programming in Rust with async-std and Tokio*. <https://tokio.rs/tokio/tutorial>

Klabnik, S., & Nichols, C. (2019). *The Rust Programming Language* (2nd ed.). No Starch Press.
<https://doc.rust-lang.org/book/>