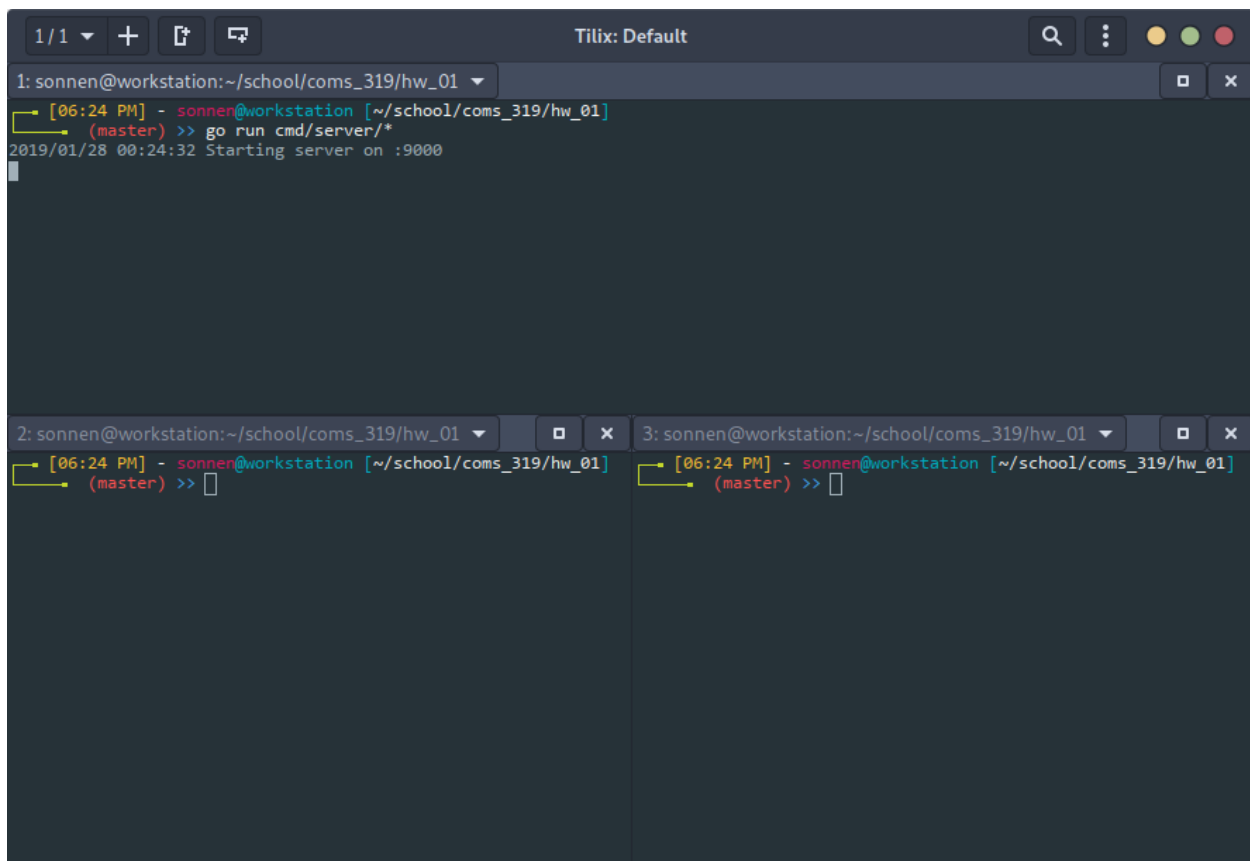


For my assignment I decided to use Golang for both the client and server. I have made basic web-servers in Go serving static files, so I was already somewhat familiar with the http library. After reading through the assignment I decided that using a WebSocket for the client/server communication would probably be the easiest and require the least amount of handling of http requests. The only design decision I really had to think about was how I would prevent the server from sending incoming messages to all clients. This is because my initial design was to have two threads: one handling incoming connections, upgrading them to a WebSocket and continuing to listen for messages, and another thread taking messages posted to a go “channel” and handling send them to all current connections. Since all my active connections were being stored in a map, and then iterated through to send all messages to all clients, I decided to make the key of the map a UUID. This allowed me to attach the UUID of a connection to all its messages. With the UUID embedded in the messages dropped into the msg channel, I was able to create logic that would not send a message to a active connection that matched the UUID of the given message.

A big friction-point of this assignment was trying to get the output to be pretty on Window’s console emulators, something I found to be impossible without extensive third-party libraris. Unlike OSX and Linux, Windows did not recognize the escape sequences I was using to flush a message as it was written to Stdout. On Linux/OSX this was done to format Stdout and present it in the terminal in color with the username present. I finally gave up on making it pretty on Windows and settled with double printing all messages typed to Stdout. This error is not present when the client/server are running with OSX, Linux, or Docker.

*The following screenshots are displayed with the server being in the top terminal, and two separate clients running in the bottom two terminals.*

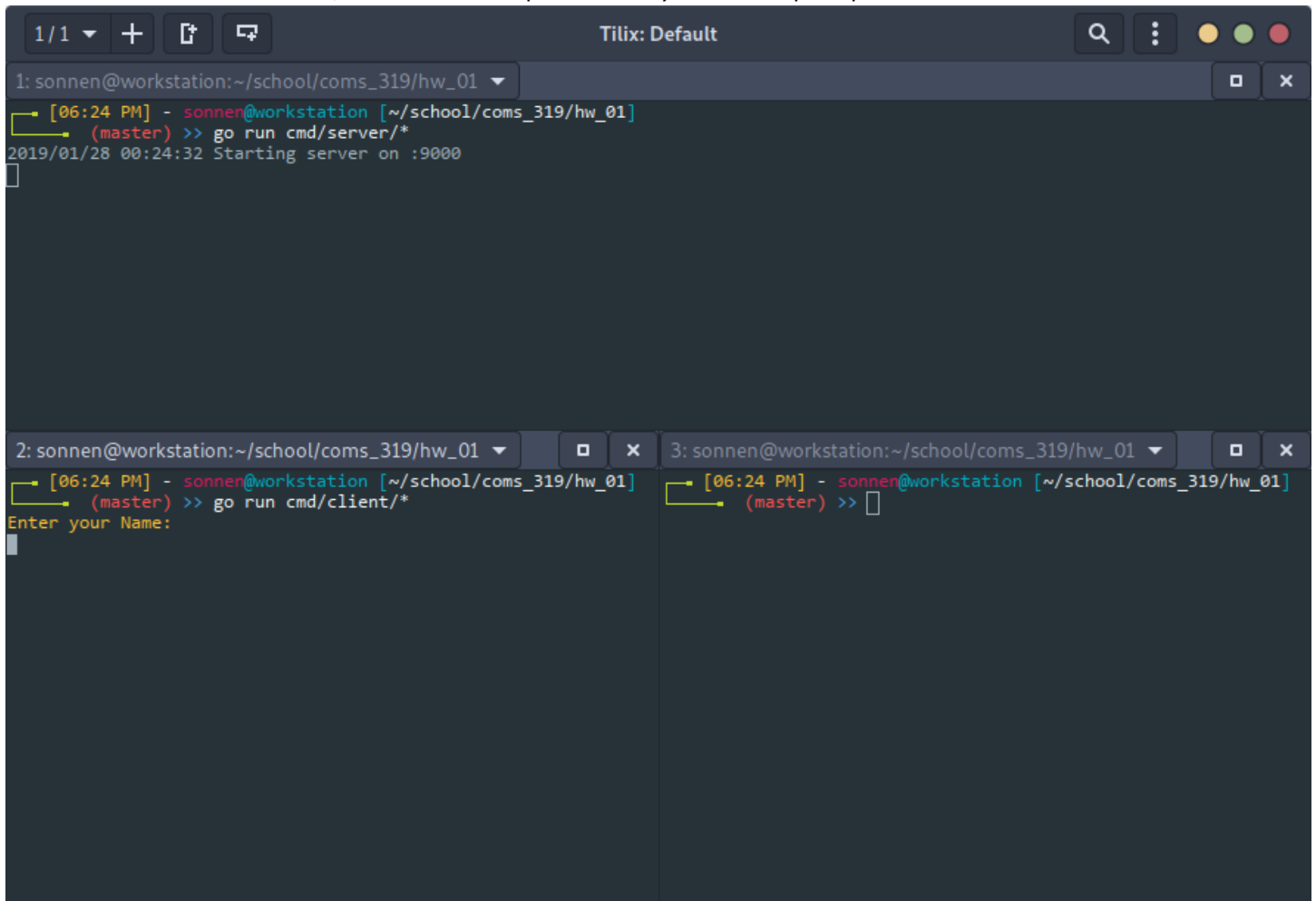


The image shows three terminal windows from the Tilix application. The top window, titled '1: sonnen@workstation:~/school/coms\_319/hw\_01', shows the command 'go run cmd/server/\*' being executed, resulting in the output '2019/01/28 00:24:32 Starting server on :9000'. The bottom-left window, titled '2: sonnen@workstation:~/school/coms\_319/hw\_01', shows the prompt '(master) >>' with a cursor. The bottom-right window, titled '3: sonnen@workstation:~/school/coms\_319/hw\_01', also shows the prompt '(master) >>' with a cursor.

Figure 1 Server starting

## 1.1 Connect to Server

- a. When client starts, it should come up with *Enter your Name: prompt*.



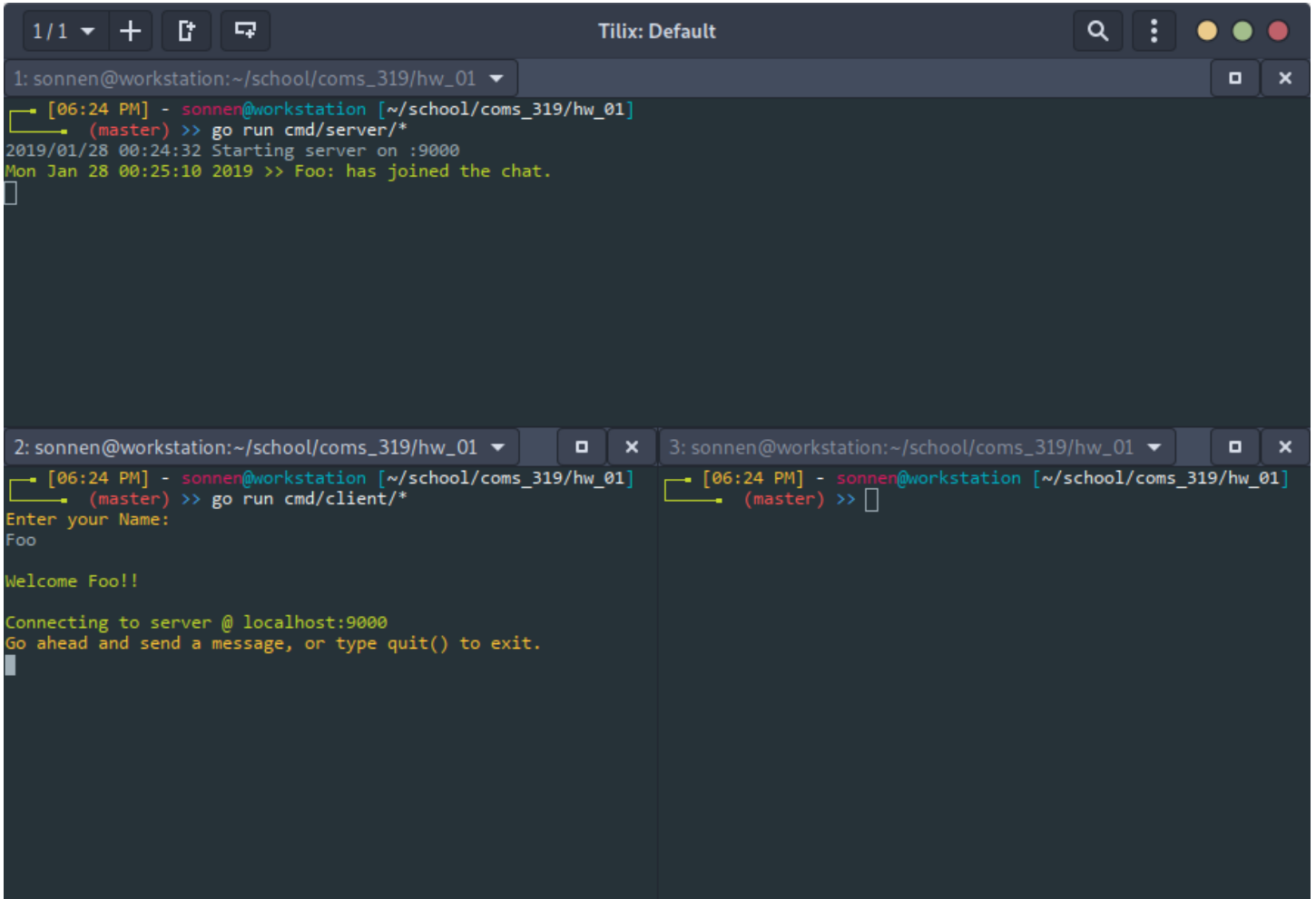
```
1: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/server/*
2019/01/28 00:24:32 Starting server on :9000

2: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/client/*
Enter your Name:

3: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >>
```

Figure 2 Enter name prompt, bottom Left

- b. After entering a name, the client should connect to server.



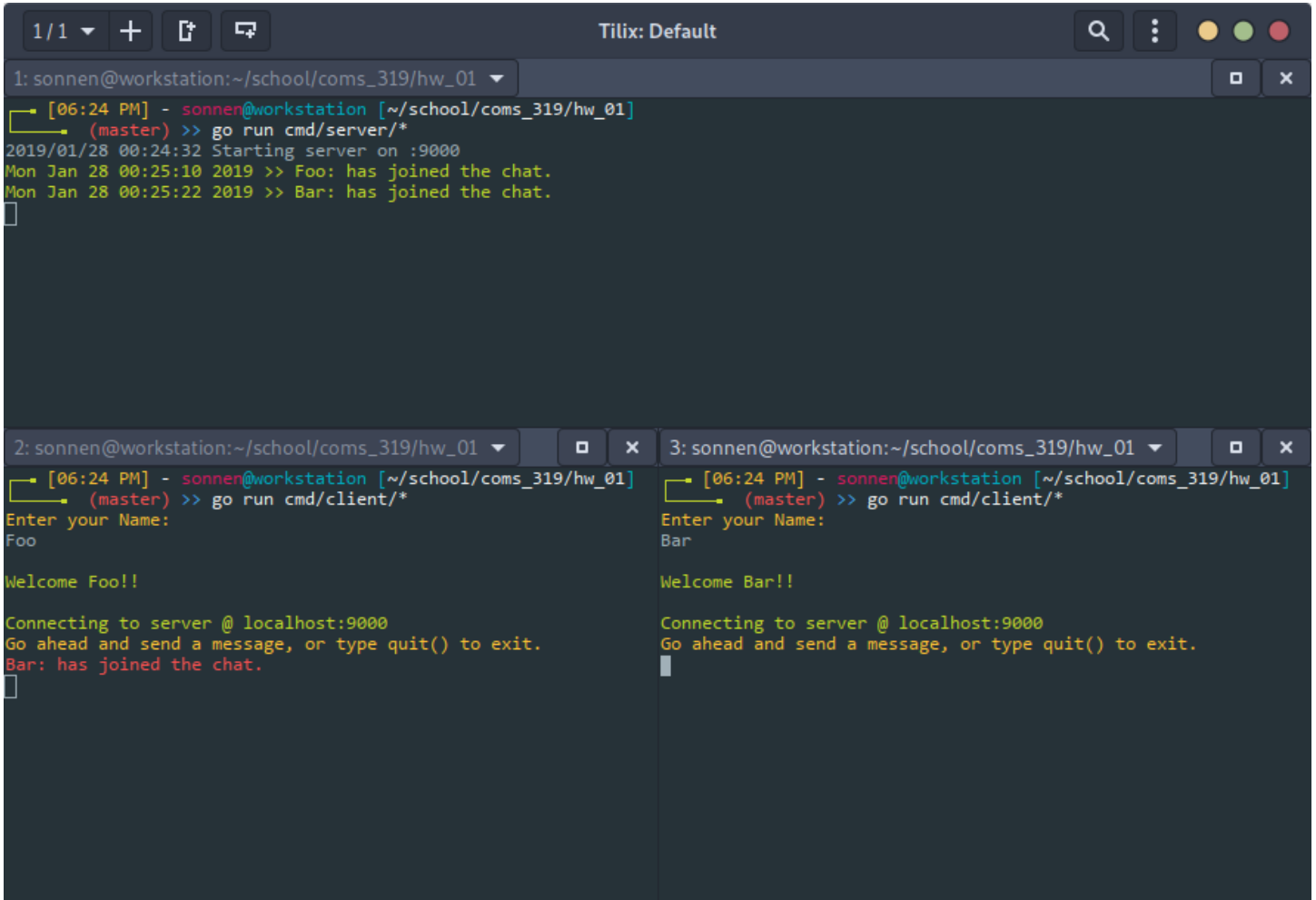
The screenshot shows a Tmux terminal window with three panes. The top pane (1) shows the server being started. The bottom-left pane (2) shows the client connecting and sending a message. The bottom-right pane (3) shows the master prompt.

```
1: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/server/*
2019/01/28 00:24:32 Starting server on :9000
Mon Jan 28 00:25:10 2019 >> Foo: has joined the chat.

2: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/client/*
Enter your Name:
Foo
Welcome Foo!!
Connecting to server @ localhost:9000
Go ahead and send a message, or type quit() to exit.

3: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >>
```

Figure 3 Client One connects



The screenshot shows a Tmux terminal window with three panes. The top pane shows the server running. The bottom-left pane shows client 'Foo' connecting and receiving a welcome message. The bottom-right pane shows client 'Bar' connecting and receiving a welcome message.

```
1: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/server/*
2019/01/28 00:24:32 Starting server on :9000
Mon Jan 28 00:25:10 2019 >> Foo: has joined the chat.
Mon Jan 28 00:25:22 2019 >> Bar: has joined the chat.

2: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/client/*
Enter your Name:
Foo
Welcome Foo!!
Connecting to server @ localhost:9000
Go ahead and send a message, or type quit() to exit.
Bar: has joined the chat.

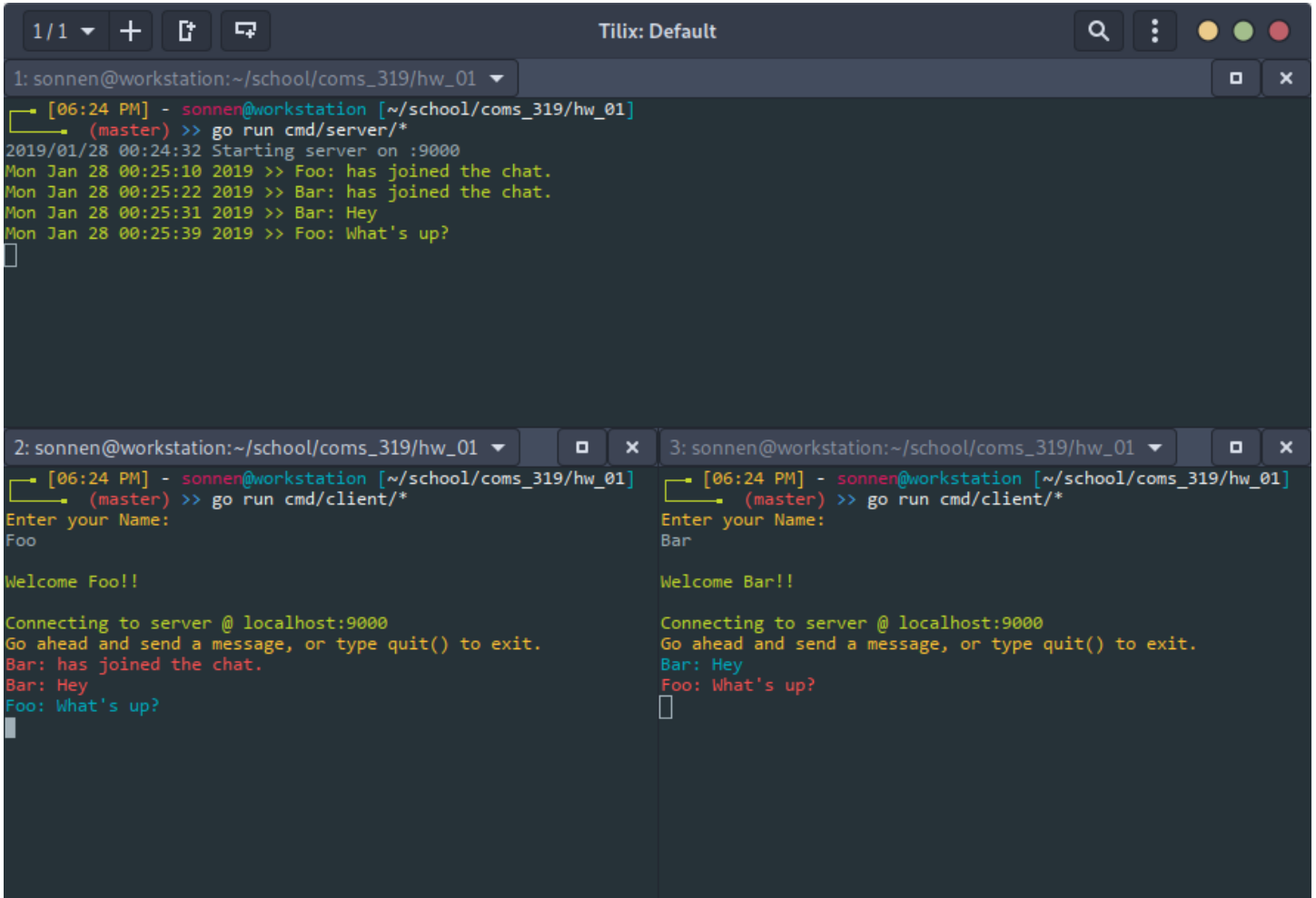
3: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/client/*
Enter your Name:
Bar
Welcome Bar!!
Connecting to server @ localhost:9000
Go ahead and send a message, or type quit() to exit.
```

Figure 4 Client Two Connects

## 1.2 Send a text message to the server

- Send username and chat message to server.

Each message is sent to the server WebSocket as a JSON object with username and text fields.



The image shows three terminal windows from the Tilix application. The top window, titled '1: sonnen@workstation:~/school/coms\_319/hw\_01', shows the server's output. It starts with a timestamp and the command 'go run cmd/server/\*'. The server logs the start time, then receives two join messages from 'Foo' and 'Bar', followed by a message from 'Bar' saying 'Hey' and a message from 'Foo' asking 'What's up?'. The bottom-left window, titled '2: sonnen@workstation:~/school/coms\_319/hw\_01', shows a client's perspective. It starts with the command 'go run cmd/client/\*', followed by the user entering 'Foo'. The client receives a 'Welcome Foo!!' message, connects to the server, and then receives broadcast messages from the server: 'Bar: has joined the chat.', 'Bar: Hey', and 'Foo: What's up?'. The bottom-right window, titled '3: sonnen@workstation:~/school/coms\_319/hw\_01', shows another client's perspective. It starts with the command 'go run cmd/client/\*', followed by the user entering 'Bar'. The client receives a 'Welcome Bar!!' message, connects to the server, and then receives broadcast messages from the server: 'Bar: Hey' and 'Foo: What's up?'.

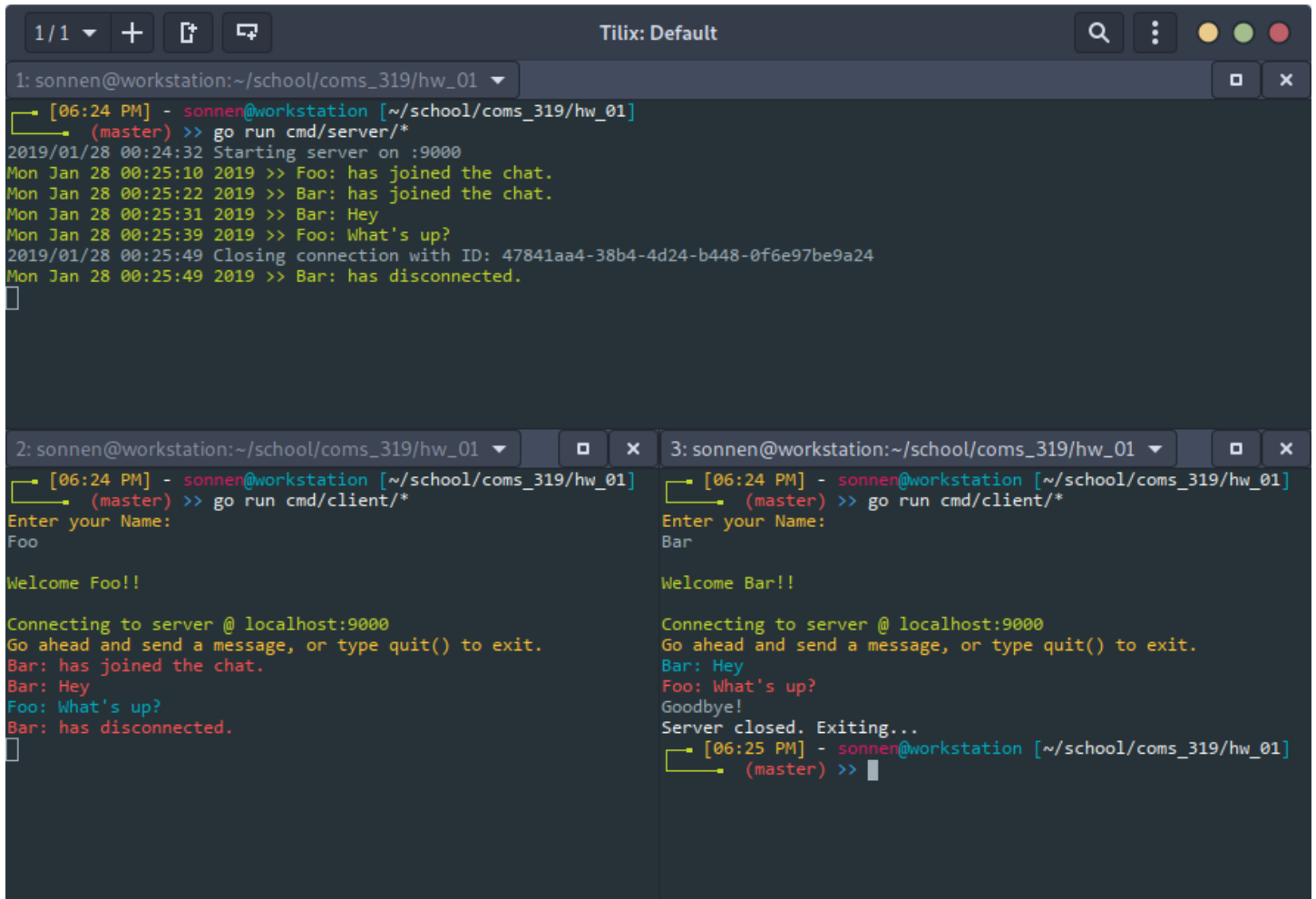
```
1: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/server/*
2019/01/28 00:24:32 Starting server on :9000
Mon Jan 28 00:25:10 2019 >> Foo: has joined the chat.
Mon Jan 28 00:25:22 2019 >> Bar: has joined the chat.
Mon Jan 28 00:25:31 2019 >> Bar: Hey
Mon Jan 28 00:25:39 2019 >> Foo: What's up?

2: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/client/*
Enter your Name:
Foo
Welcome Foo!!
Connecting to server @ localhost:9000
Go ahead and send a message, or type quit() to exit.
Bar: has joined the chat.
Bar: Hey
Foo: What's up?

3: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/client/*
Enter your Name:
Bar
Welcome Bar!!
Connecting to server @ localhost:9000
Go ahead and send a message, or type quit() to exit.
Bar: Hey
Foo: What's up?
```

Figure 5 Messages back and forth

- b. Server broadcast's Clients message to every client other than sending client.  
See figure 5, all messages are logged to the server's Stdout with the time the message was sent. Clients only receive messages that did not originate by them, this ensured by checking each message against a UUID assigned to each of the server's active connections.
- c. Message printed in each client's console and server's console.  
See figure 5. Each of the messages are printed in color with incoming messages appearing in red and outgoing messages in blue.



The screenshot shows a terminal window titled "Tilix: Default" with three tabs. The first tab, titled "1: sonnen@workstation:~/school/coms\_319/hw\_01", shows the server's output. The second tab, titled "2: sonnen@workstation:~/school/coms\_319/hw\_01", shows the output of a client named "Foo". The third tab, titled "3: sonnen@workstation:~/school/coms\_319/hw\_01", shows the output of a client named "Bar".

```
1: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/server/*
2019/01/28 00:24:32 Starting server on :9000
Mon Jan 28 00:25:10 2019 >> Foo: has joined the chat.
Mon Jan 28 00:25:22 2019 >> Bar: has joined the chat.
Mon Jan 28 00:25:31 2019 >> Bar: Hey
Mon Jan 28 00:25:39 2019 >> Foo: What's up?
2019/01/28 00:25:49 Closing connection with ID: 47841aa4-38b4-4d24-b448-0f6e97be9a24
Mon Jan 28 00:25:49 2019 >> Bar: has disconnected.

2: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/client/*
Enter your Name:
Foo
Welcome Foo!!

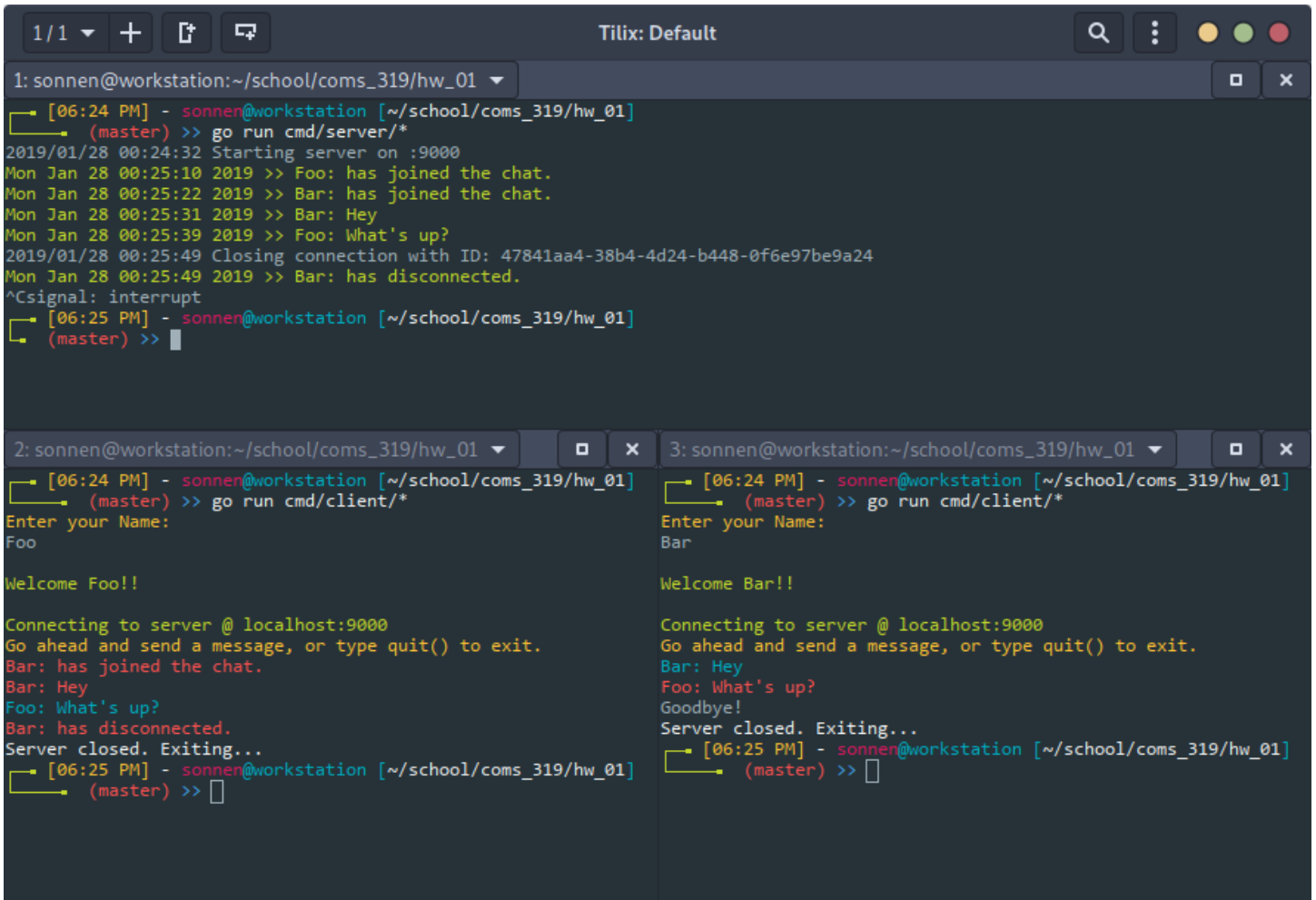
Connecting to server @ localhost:9000
Go ahead and send a message, or type quit() to exit.
Bar: has joined the chat.
Bar: Hey
Foo: What's up?
Bar: has disconnected.

3: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/client/*
Enter your Name:
Bar
Welcome Bar!!

Connecting to server @ localhost:9000
Go ahead and send a message, or type quit() to exit.
Bar: Hey
Foo: What's up?
Goodbye!
Server closed. Exiting...
[06:25 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >>
```

Figure 6 Client Two Disconnect

After a client types `quit()` they send two messages to the server: the first is a JSON message with their name and “has disconnected”, and the second is them closing their WebSocket. When a WebSocket is closed the server prints to Stdout the UUID of the client that closed their connection.



```
1: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/server/*
2019/01/28 00:24:32 Starting server on :9000
Mon Jan 28 00:25:10 2019 >> Foo: has joined the chat.
Mon Jan 28 00:25:22 2019 >> Bar: has joined the chat.
Mon Jan 28 00:25:31 2019 >> Bar: Hey
Mon Jan 28 00:25:39 2019 >> Foo: What's up?
2019/01/28 00:25:49 Closing connection with ID: 47841aa4-38b4-4d24-b448-0f6e97be9a24
Mon Jan 28 00:25:49 2019 >> Bar: has disconnected.
^Csignal: interrupt
[06:25 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >>

2: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/client/*
Enter your Name:
Foo
Welcome Foo!!

Connecting to server @ localhost:9000
Go ahead and send a message, or type quit() to exit.
Bar: has joined the chat.
Bar: Hey
Foo: What's up?
Bar: has disconnected.
Server closed. Exiting...
[06:25 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >>

3: sonnen@workstation:~/school/coms_319/hw_01
[06:24 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >> go run cmd/client/*
Enter your Name:
Bar
Welcome Bar!!

Connecting to server @ localhost:9000
Go ahead and send a message, or type quit() to exit.
Bar: Hey
Foo: What's up?
Goodbye!
Server closed. Exiting...
[06:25 PM] - sonnen@workstation [~/school/coms_319/hw_01]
(master) >>
```

Figure 7 Server exited

If the client loses connection with the server, it will print to Stdout saying the server has closed, and will exit the client application.