Task 1:

```go
import (
    "fmt"
    "log"
    "net/http"

    "github.com/gorilla/websocket"
)

// Upgrader upgrades HTTP connection to WebSocket
var upgrader = websocket.Upgrader{
    ReadBufferSize:  1024,
    WriteBufferSize: 1024,
    CheckOrigin: func(r *http.Request) bool {
        return true
    },
}
```

- Importing the Gorilla WebSocket package.
- upgrader upgrades HTTP connection to WebSocket (ReadBufferSize and WriteBufferSize are size buffer for reading/writing data, CheckOrigin true to allow all connection).

```go
// Handle WebSocket connections
func handleWebSocket(w http.ResponseWriter, r *http.Request) {
    // Upgrade HTTP connection to WebSocket
    conn, err := upgrader.Upgrade(w, r, nil)
    if err != nil {
        log.Printf("Failed to upgrade connection: %v", err)
        return
    }
    defer conn.Close()

    log.Printf("New client connected: %s", conn.RemoteAddr())

    // Read messages in a loop
    for {
        // Read message from client
        messageType, message, err := conn.ReadMessage()
        if err != nil {
            log.Printf("Client disconnected: %v", err)
            break
        }

        log.Printf("Received: %s", message)

        // Echo message back to client
        err = conn.WriteMessage(messageType, message)
        if err != nil {
            log.Printf("Failed to write message: %v", err)
            break
        }

        log.Printf("Echoed: %s", message)
    }
}
```

- w http.ResponseWriter: lets the server send a response to the client
- r *http.Request: contains the content of HTTP request from the client.
- upgrader.Upgrade converts the HTTP connection to WebSocket.
- Then read messages from client in a loop. If the client disconnects, the loop breaks.
- Echo messages back to sender.

```go
func main() {
    // Route for WebSocket connection
    http.HandleFunc("/ws", handleWebSocket)

    fmt.Println("🚀 WebSocket Echo Server started on :8080")
    fmt.Println("📱 Connect using CLI client: go run client/echo_client.go")

    err := http.ListenAndServe(":8080", nil)
    if err != nil {
        log.Fatal("Server failed:", err)
    }
}
```

- http.HandleFunc("/ws", handleWebSocket): registers /ws endpoint to handle WebSocket connections.
- http.ListenAndServe(":8080", nil): starts the HTTP server on port 8080
- log.Fatal logs any startup error and exits.

Output:

Server:

```
PS C:\Users\VUTHANHHUNG\Desktop\Netcen Pro\VuThanhNhan - ITITIU21267 - Lab8\Task1\server> go run .\ma
in.go
🚀 WebSocket Echo Server started on :8080
📱 Connect using CLI client: go run client/echo_client.go
2025/12/04 14:22:16 New client connected: [::1]:64577
2025/12/04 14:22:31 Received: Hello Server!
2025/12/04 14:22:31 Echoed: Hello Server!
```

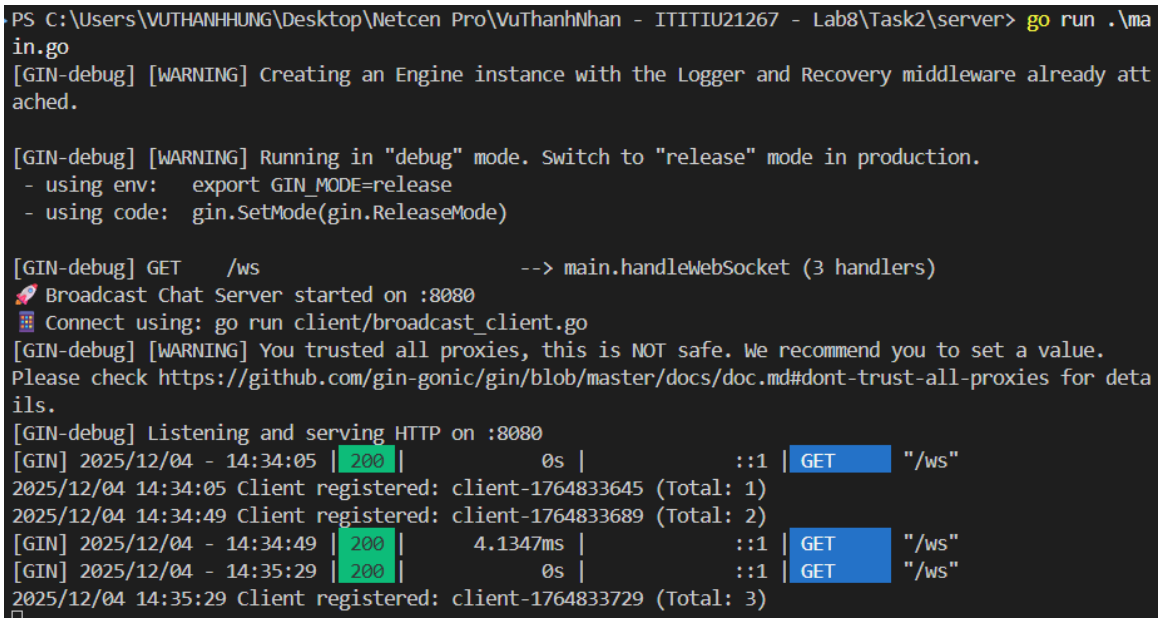
Client:

```
✓ Connected to echo server
Type messages and press Enter (Ctrl+C to exit)
---
Hello Server!
Echo: Hello Server!
```

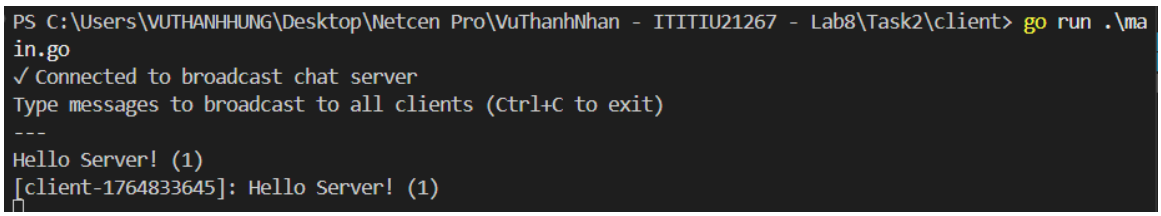
Task 2:

Output:

Server:



Client 1:



Client 2: