Rust-Go WebSocket Chat System

Săvianu Bogdan

April 24, 2025

1 Abstract

This project implements a real-time chat system using a WebSocket server written in Rust and clients written in Go. The server supports multiple concurrent clients and broadcasts messages between them.

2 Architecture

- **Rust Server:** Handles WebSocket connections, message broadcasting, and client management.
- **Go Clients:** Connect to the server, send user input, and display incoming messages.
- **Communication:** Full-duplex WebSocket protocol over TCP.

3 Setup Instructions

3.1 Rust Server

1. Install Rust: `https://rustup.rs`
2. Clone the repository and navigate to the server directory.
3. Run the server:

```
    cargo run
```

3.2 Go Client

1. Install Go: `https://go.dev/dl`
2. Navigate to the client directory.
3. Initialize the module and install dependencies:

```
    go mod init go_ws_client
    go get github.com/gorilla/websocket
```

4. Run the client:

```
    go run .
```

4 How It Works

The system is composed of a Rust-based WebSocket server and multiple Go clients. The server listens for incoming WebSocket connections and spawns a dedicated task for each client. Each task handles reading messages from the client and broadcasting them to all other connected clients using a shared broadcast channel.

Detailed Flow

1. The Rust server starts and listens on a specified TCP port.
2. A Go client connects to the server using the WebSocket protocol.
3. The server accepts the connection and spawns a task to manage it.
4. The client sends a message (e.g., user input).
5. The server receives the message and broadcasts it to all connected clients.
6. Each client receives the message and displays it in the terminal.

Sequence Diagram

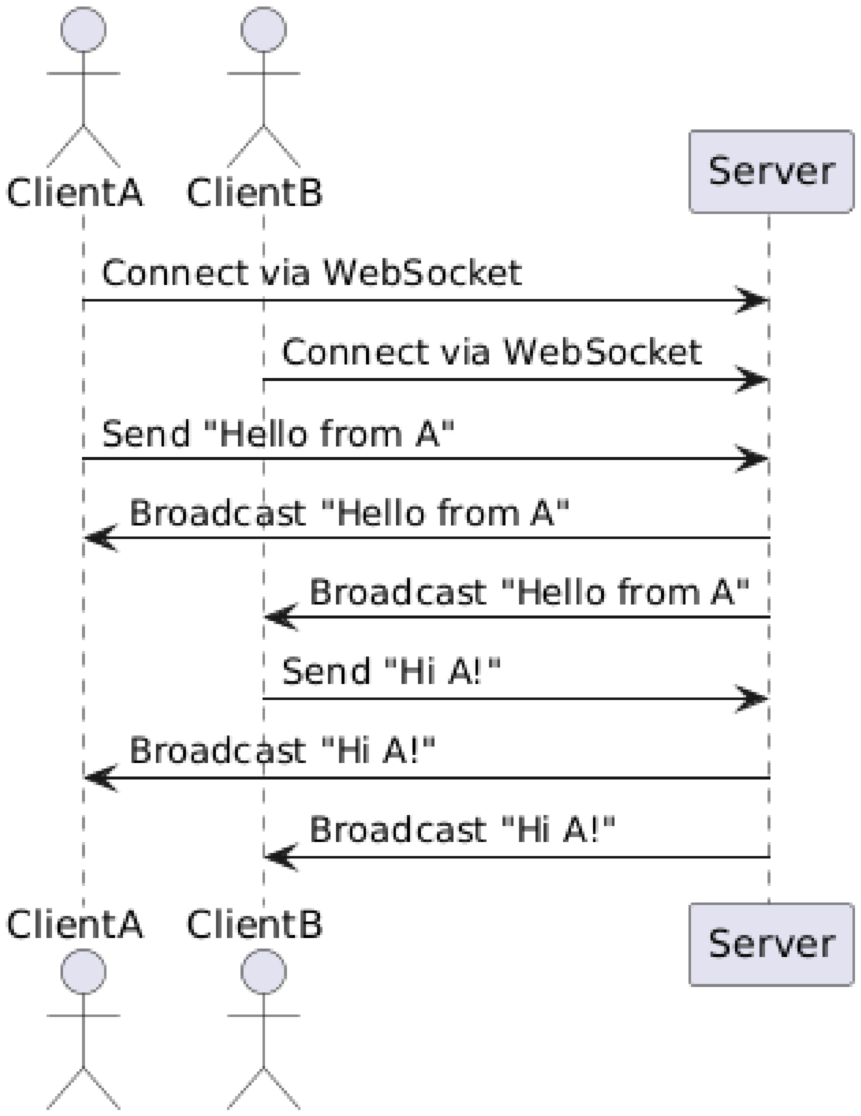

Figure 1: Rectangle result