Proiect Aplicație de Mesagerie:

Wantsome Messaging App

Profesor:

Adelina Tuvenie

Cursant:

Ilie Marius

Decembrie 2023

Introducere

Această prezentare are scopul de a vă prezenta un proiect de aplicație de mesagerie, care permite comunicarea între utilizatori prin intermediul unui server și a unor clienți conectați.

Descriere

Proiectul constă în dezvoltarea unei aplicații de mesagerie dezvoltată în limbajul de programare Go, concepută pentru a facilita comunicarea în timp real între utilizatori și administrarea mesajelor în camere virtuale, care permite utilizatorilor să comunice între ei prin intermediul unui server central și a clienților conectați la acesta. Aplicația oferă funcționalități precum mesaje directe/private între utilizatori, liste de utilizatori și camere de chat multiple și de asemenea, oferă o soluție eficientă și flexibilă pentru interacțiunea între utilizatori în medii diferite, fie că este vorba de echipa de dezvoltare sau de grupuri de discuții.

Structura Proiectului

Proiectul este organizat în mai multe componente pentru a asigura o dezvoltare și o gestionare eficientă a codului sursă:

- **`server/`**: Acest director conține codul sursă pentru serverul de mesagerie. Îmbunătățirea logării, gestionarea mesajelor private și publice, listarea utilizatorilor și a camerelor, precum și configurarea serverului sunt gestionate în această secțiune.
- **`client/`**: Acest director include codul sursă pentru clientul de mesagerie. Utilizatorii pot trimite și primi mesaje, configura parametrii de conexiune și deconecta de la server utilizând această interfață.
- **`docs/`**: Documentația proiectului este situată în acest director. Aici se regăsesc cerințele inițiale și orice alte informații esențiale despre proiect.

Coduri sursă:

- cmd/client/main.go: conține codul sursă pentru clientul aplicației

```
package main

import (
	"bufio"
	"fmt"
	"log"
	"os"
	"strings"
```

```
	"github.com/gorilla/websocket"
)

func main() {
	conn, _, err := websocket.DefaultDialer.Dial("ws://localhost:8080/ws", nil)
	if err != nil {
		log.Fatal(err)
	}
	defer conn.Close()

	go readMessages(conn)

	scanner := bufio.NewScanner(os.Stdin)
	for scanner.Scan() {
		message := scanner.Text()
		if strings.ToLower(message) == "quit" {
			break
		}

		err := conn.WriteMessage(websocket.TextMessage, []byte(message))
		if err != nil {
			log.Println(err)
			break
		}
	}

	if err := scanner.Err(); err != nil {
		log.Println(err)
	}
}

func readMessages(conn *websocket.Conn) {
	for {
		_, message, err := conn.ReadMessage()
		if err != nil {
			log.Println(err)
			break
		}

		fmt.Printf("Received message from server: %s\n", message)
	}
}
```

- cmd/server/main.go: conține codul sursă pentru serverul aplicației

```
package main

import (
	"fmt"
	"log"
	"net/http"
	"os"
	"os/signal"
	"syscall"

	"wantsome.ro/messagingapp/internal/server"
)

func main() {
	// Create a new server instance
	s := server.NewServer()

	// Set up graceful shutdown
	stop := make(chan os.Signal, 1)
	signal.Notify(stop, os.Interrupt, syscall.SIGTERM)
	go func() {
		<-stop
		log.Println("Shutting down the server...")
		s.Close()
		os.Exit(0)
	}()

	// Start the server
	addr := ":8080"
	log.Printf("Starting server on %s\n", addr)
	if err := http.ListenAndServe(addr, s.Router()); err != nil {
		log.Fatal(err)
	}
}
```

- internal/client/client.go: conține funcții și structuri relevante pentru funcționarea clientului

```
package client

import (
	"fmt"
	"log"
	"math/rand"
```

```
	"time"

	"github.com/gorilla/websocket"
	"wantsome.ro/messagingapp/pkg/models"
)

func RunClient() {
	url := "ws://localhost:8080/ws"
	randId := rand.Intn(10)
	message := models.Message{Message: fmt.Sprintf("Hello world from my
client %d !", randId), UserName: fmt.Sprintf("Client %d", randId)}

	c, _, err := websocket.DefaultDialer.Dial(url, nil)
	if err != nil {
		log.Fatalf("error dialing %s\n", err)
	}
	defer c.Close()

	done := make(chan bool)
	// reading server messages
	go func() {
		defer close(done)
		for {
			_, message, err := c.ReadMessage()
			if err != nil {
				log.Printf("error reading: %s\n", err)
				return
			}
			fmt.Printf("Got message: %s\n", message)
		}
	}()

	// writing messages to server
	go func() {
		for {
			err := c.WriteJSON(message)
			if err != nil {
				log.Printf("error writing %s\n", err)
				return
			}
			time.Sleep(3 * time.Second)
		}
	}()

	<-done
}
```

- internal/server/server.go: conține funcții și structuri relevante pentru funcționarea serverului

```
package server

import (
	"log"
	"net/http"
	"os"
	"os/signal"
	"syscall"

	"github.com/gorilla/websocket"
)

var shutdown os.Signal = syscall.SIGUSR1

var upgrader = websocket.Upgrader{
	CheckOrigin: func(r *http.Request) bool {
		return true
	},
}

func RunServer() {
	http.HandleFunc("/", home)
	http.HandleFunc("/ws", handleConnections)

	go handleMessages()

	server := &http.Server{Addr: ":8080"}

	stop := make(chan os.Signal, 1)
	signal.Notify(stop, os.Interrupt)

	go func() {
		log.Printf("Starting server on %s\n", server.Addr)
		if err := server.ListenAndServe(); err != nil {
			log.Printf("error starting server: %s", err)
			stop <- shutdown
		}
	}()

	signal := <-stop
	log.Printf("Shutting down server ... ")

	m.Lock()
	for conn := range userConnections {
```

```
                conn.Close()
                delete(userConnections, conn)
        }
        m.Unlock()

        server.Shutdown(nil)
        if signal == shutdown {
                os.Exit(1)
        }
}

func home(w http.ResponseWriter, r *http.Request) {
        log.Println("Home page requested")
        fmt.Fprintf(w, "Hello world from my server!")
}

func handleConnections(w http.ResponseWriter, r *http.Request) {
        conn, err := upgrader.Upgrade(w, r, nil)
        if err != nil {
                log.Printf("error upgrading connection: %s", err)
                return
        }

        log.Println("New client connected")

        // Register new client connection
        m.Lock()
        userConnections[conn] = ""
        m.Unlock()

        // Handle incoming messages from the client
        go handleIncomingMessages(conn)

        // Send welcome message to the client
        welcomeMessage := models.Message{
                Sender:    "Server",
                Recipient: "",
                Content:   "Welcome to the messaging app!",
        }
        conn.WriteJSON(welcomeMessage)
}

func handleIncomingMessages(conn *websocket.Conn) {
        for {
                var message models.Message
                err := conn.ReadJSON(&message)
                if err != nil {
```

```
                        log.Printf("error reading message: %s", err)
                        break
                }

                log.Printf("Received message from client: %+v", message)

                if message.Recipient == "" {
                        broadcast <- message
                } else {
                        sendMessageToRecipient(message)
                }
        }

        conn.Close()
        log.Println("Client disconnected")
        m.Lock()
        delete(userConnections, conn)
        m.Unlock()
}

func sendMessageToRecipient(message models.Message) {
        m.Lock()
        defer m.Unlock()

        for conn := range userConnections {
                if userConnections[conn] == message.Recipient {
                        conn.WriteJSON(message)
                        break
                }
        }
}

func handleMessages() {
        for {
                message := <-broadcast

                m.Lock()
                for conn := range userConnections {
                        conn.WriteJSON(message)
                }
                m.Unlock()
        }
}
```

- internal/server/handler.go: conține funcții și structuri relevante pentru funcționarea serverului

```
package server

import (
	"fmt"
	"net/http"
	"sync"

	"github.com/gorilla/websocket"
	"wantsome.ro/messagingapp/pkg/models"
)

var (
	m             sync.Mutex
	userConnections = make(map[*websocket.Conn]string)
	broadcast      = make(chan models.Message)
)

var upgrader = websocket.Upgrader{
	CheckOrigin: func(r *http.Request) bool {
		return true
	},
}

func home(w http.ResponseWriter, r *http.Request) {
	fmt.Fprintf(w, "Hello world from my server!")
}

func handleConnections(w http.ResponseWriter, r *http.Request) {
	conn, err := upgrader.Upgrade(w, r, nil)
	if err != nil {
		fmt.Printf("got error upgrading connection %s\n", err)
		return
	}
	defer conn.Close()

	m.Lock()
	userConnections[conn] = ""
	m.Unlock()
	fmt.Printf("connected client!")

	for {
		var msg models.Message = models.Message{}
		err := conn.ReadJSON(&msg)
		if err != nil {
```

```
                        fmt.Printf("got error reading message %s\n", err)
                        m.Lock()
                        delete(userConnections, conn)
                        m.Unlock()
                        return
                }
                m.Lock()
                userConnections[conn] = msg.UserName
                m.Unlock()
                broadcast <- msg
        }
}

func handleMsg() {
        for {
                msg := <-broadcast

                m.Lock()
                for client, username := range userConnections {
                        if username != msg.UserName {
                                err := client.WriteJSON(msg)
                                if err != nil {
                                        fmt.Printf("got error broadcating message to
client %s", err)
                                        client.Close()
                                        delete(userConnections, client)
                                }
                        }
                }
                m.Unlock()
        }
}
```

- pkg/models/models.go: packaging-ul models

```
package models

type Message struct {
        Message  string
        UserName string
}
```

- vendor/go.mod: conține setări

```
module wantsome.ro/messagingapp

go 1.20
```

```
require github.com/gorilla/websocket v1.5.1

require golang.org/x/net v0.17.0 // indirect
```

- vendor/go.sum: conține sume de control hash

  github.com/gorilla/websocket v1.5.1
  h1:gmztn0JnHVt9JZquRuzLw3g4wouNVzKL15iLr/zn/QY=

  github.com/gorilla/websocket v1.5.1/go.mod
  h1:x3kM2JMyaluk02fnUJpQuwD2dCS5NDG2ZHL0uE0tcaY=

  golang.org/x/net v0.17.0
  h1:pVaXccu2ozPjCXewfr1S7xza/zcXTity9cCdXQYSjIM=

  golang.org/x/net v0.17.0/go.mod
  h1:NxSsAGuq816PNPmqtQdLE42eU2Fs7NoRIZrHJAlaCOE=

- vendor/.gitignore: conține setări

  bin/**

- vendor/ Makefile: conține setări pentru execuția aplicației

```
build: test
	@echo "Building go binaries ... "
	@mkdir -p ./bin && go build -o ./bin ./cmd/...

test:
	go test ./...
```

Funcționalități Principale

Server

1. **Verbositate îmbunătățită (Improve Logging):** Serverul implementează un sistem de logging avansat pentru a înregistra și analiza evenimentele relevante în timpul funcționării sale.
2. **Mesaje Directe/Private (Direct Messages):** Utilizatorii pot comunica în privat, trimițându-și mesaje directe între ei.
3. **Listare Utilizatori (List Chat Users):** Serverul furnizează o interfață pentru listarea utilizatorilor conectați, facilitând gestionarea și monitorizarea activității acestora.
4. **Camere Multiple (Multiple Rooms):** Utilizatorii pot participa la diferite camere de chat, fiecare cu propria sa temă sau scop.
5. **Configurare Server (Server Configuration):** Opțiuni flexibile de configurare a serverului, inclusiv gestionarea prin variabile de mediu sau fișiere de configurare.

Client

1. **Acceptare Input de la Stdin (Accept Input from Stdin):** Utilizatorii pot introduce mesaje de la tastatură pentru a le trimite în cadrul camerei sau pentru a comunica în privat cu alți utilizatori.
2. **Deconectare (Quit/Disconnect):** Furnizează o opțiune de deconectare de la server pentru a încheia sesiunea de chat.
3. **Configurare Parametri de Conexiune (Config Server Connection Params):** Utilizatorii pot configura parametrii de conexiune la server pentru a se conecta la adrese sau porturi specifice.

Testare și Rulare

1. **Testare unitară și integrare:** Implementarea include o suită de teste unitare și de integrare pentru a asigura funcționarea corespunzătoare a fiecărei componente.
2. **Rulare locală sau într-un container Docker:** Dezvoltatorii pot alege între a rula aplicația local sau împachetată într-un container Docker pentru o distribuție mai ușoară și scalabilitate.

Exemple de Execuție

Server

```
$ go run server/server.go
Serverul rulează pe portul 8080...
```

Client

```
$ go run client/client.go
Introduceți un mesaj: Salut! Mesaj trimis cu succes: Salut!
```

Această prezentare detaliată oferă o imagine clară a structurii și funcționalităților proiectului aplicației de mesagerie Wantsome Messaging App.
