**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

# Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по ДЗ

Вариант 21

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-35Б |  |  |
| Ханифов С.В. |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2024 г

**Описание задания**

Мессенджен на голанге, для взаимодействия используется вебсокет.

**Текст программы**

"C:\Go-Projects\Small-Messenger\cmd\client\main.go":

"package main

import (

"messenger/internal/iface"

"os"

"github.com/rs/zerolog"

"github.com/rs/zerolog/log"

)

func main() {

im := iface.NewInterfaceManager()

log.Info().Msg("client is running")

im.RunApp()

}

func init() {

log.Logger = log.Output(zerolog.ConsoleWriter{Out: os.Stderr})

}

"

"C:\Go-Projects\Small-Messenger\cmd\client\token.env":

"REGISTRATION\_TOKEN=

"

"C:\Go-Projects\Small-Messenger\cmd\server\config.env":

"DB\_USER = postgres

DB\_PASSWORD = mypass

DB\_NAME = messenger

DB\_SSL\_MODE = disable

DB\_DRIVER = postgres

PORT = 80"

"C:\Go-Projects\Small-Messenger\cmd\server\main.go":

"package main

import (

"fmt"

"messenger/config"

controller "messenger/internal/controller/server"

"messenger/internal/middleware"

"messenger/internal/repository"

"messenger/internal/service/service"

"os"

"os/signal"

"syscall"

"github.com/gin-gonic/gin"

"github.com/pkg/errors"

"github.com/rs/zerolog"

"github.com/rs/zerolog/log"

"github.com/rs/zerolog/pkgerrors"

)

func main() {

router := gin.Default()

cfg := config.New()

db := config.NewDB(cfg)

repo := repository.New(db)

cs := service.New(repo)

cr := controller.New(repo, cs)

router.POST("/register", cr.Register)

router.POST("/login", cr.Login)

auth := router.Group("/")

auth.Use(middleware.Authentication())

auth.GET("/chats", cr.GetChats)

auth.GET("/chats/:id/history", cr.GetLastChatMessages)

auth.GET("/token", cr.ValidateTokenHandler)

auth.GET("/ws/chats", cr.NewChat)

auth.GET("/ws/chats/:id", cr.JoinChat)

errs := make(chan error)

go func() {

err := router.Run(fmt.Sprintf(":%s", cfg.PORT))

errs <- errors.Wrap(err, "running server")

}()

go func() {

c := make(chan os.Signal, 1)

signal.Notify(c, syscall.SIGINT, syscall.SIGTERM)

buf := <-c

errs <- errors.New(buf.String())

}()

log.Fatal().Err(<-errs).Msg("")

}

func init() {

log.Logger = log.Output(zerolog.ConsoleWriter{Out: os.Stderr})

zerolog.ErrorStackMarshaler = pkgerrors.MarshalStack

}

"

"C:\Go-Projects\Small-Messenger\config\config.go":

"package config

import (

"github.com/rs/zerolog/log"

"github.com/spf13/viper"

)

type Config struct {

DB\_USER string

DB\_PASSWORD string

DB\_NAME string

DB\_SSL\_MODE string

DB\_DRIVER string

PORT string

}

func New() \*Config {

viper.SetConfigFile("config.env")

err := viper.ReadInConfig()

if err != nil {

log.Fatal().Err(err).Msg("cant load cfg")

}

return &Config{

viper.GetString("DB\_USER"),

viper.GetString("DB\_PASSWORD"),

viper.GetString("DB\_NAME"),

viper.GetString("DB\_SSL\_MODE"),

viper.GetString("DB\_DRIVER"),

viper.GetString("PORT"),

}

}

"

"C:\Go-Projects\Small-Messenger\config\db.go":

"package config

import (

"fmt"

"github.com/rs/zerolog/log"

"github.com/jmoiron/sqlx"

\_ "github.com/lib/pq"

)

func NewDB(cfg \*Config) \*sqlx.DB {

connStr := fmt.Sprintf("user=%s password=%s dbname=%s sslmode=%s", cfg.DB\_USER, cfg.DB\_PASSWORD, cfg.DB\_NAME, cfg.DB\_SSL\_MODE)

db, err := sqlx.Open(cfg.DB\_DRIVER, connStr)

if err != nil {

log.Fatal().Err(err).Msg("cant connect db")

}

if err := db.Ping(); err != nil {

log.Fatal().Err(err).Msg("cant ping db")

}

return db

}

"

"C:\Go-Projects\Small-Messenger\internal\authentication\auth.go":

"package authentication

import (

"time"

"github.com/dgrijalva/jwt-go"

"github.com/pkg/errors"

)

var (

secretKey = []byte("super-secret-key")

sighError = errors.New("unknown sigh method")

tokenError = errors.New("invalid token")

)

type Claims struct {

Id int

jwt.StandardClaims

}

func CreateToken(Id int) (string, error) {

claims := &Claims{

Id: Id,

StandardClaims: jwt.StandardClaims{

ExpiresAt: time.Now().Add(time.Hour \* 72).Unix(),

},

}

token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)

t, err := token.SignedString(secretKey)

return t, errors.Wrap(err, "signing token")

}

func ValidateToken(token string) (\*Claims, error) {

claims := &Claims{}

parsedToken, err := jwt.ParseWithClaims(token, claims, func(t \*jwt.Token) (interface{}, error) {

if \_, ok := t.Method.(\*jwt.SigningMethodHMAC); !ok {

return nil, errors.Wrap(sighError, "signing method")

}

return secretKey, nil

})

if err != nil {

return nil, err

}

if !parsedToken.Valid {

return nil, errors.Wrap(tokenError, "validating token")

}

return claims, nil

}

"

"C:\Go-Projects\Small-Messenger\internal\controller\client\chat.go":

"package controller

import (

"fmt"

"messenger/internal/models"

"messenger/internal/utils"

"net/http"

"github.com/rs/zerolog/log"

)

func (hm \*HandlersManager) GetChatsHandler() map[string]string {

resp := make(map[string]string)

response, err := hm.client.R().

SetHeader("Authorization", utils.GetToken()).

SetResult(&resp).

Get(fmt.Sprintf("https://%s/chats", hm.addr))

if err != nil {

log.Error().Err(err).Msg("cant request getting chats")

return nil

}

if response.StatusCode() != http.StatusOK {

log.Error().Msg(string(response.Body()))

}

return resp

}

func (hm \*HandlersManager) ChatHistoryHandler(chatId string) []models.ShortMessage {

var resp []models.ShortMessage

response, err := hm.client.R().

SetHeader("Authorization", utils.GetToken()).

SetResult(&resp).

Get(fmt.Sprintf("https://%s/chats/%s/history", hm.addr, chatId))

if err != nil {

log.Error().Err(err).Msg("cant request getting chats")

return nil

}

if response.StatusCode() != http.StatusOK {

log.Error().Msg(string(response.Body()))

}

return resp

}

"

"C:\Go-Projects\Small-Messenger\internal\controller\client\manager.go":

"package controller

import (

"sync"

"github.com/go-resty/resty/v2"

)

var (

hm \*HandlersManager

once sync.Once

)

type HandlersManager struct {

client \*resty.Client

addr string

}

func GetHandlersManager() \*HandlersManager {

once.Do(func() {

client := resty.New()

hm = &HandlersManager{client: client, addr: "genuine-fish-light.ngrok-free.app"}

})

return hm

}

"

"C:\Go-Projects\Small-Messenger\internal\controller\client\user.go":

"package controller

import (

"fmt"

"messenger/internal/utils"

"net/http"

"github.com/rs/zerolog/log"

)

func (hm \*HandlersManager) RegistrationHandler(username, password string) {

type Response struct {

Token string `json:"token"`

}

var resp Response

response, err := hm.client.R().

SetHeader("Content-Type", "application/json").

SetBody(map[string]interface{}{

"username": username,

"password": password,

}).

SetResult(&resp).

Post(fmt.Sprintf("https://%s/register", hm.addr))

if err != nil {

log.Error().Err(err).Msg("cant request registration")

return

}

if response.StatusCode() != http.StatusOK {

log.Error().Msg(string(response.Body()))

return

}

utils.WriteToken(resp.Token)

}

func (hm \*HandlersManager) LoginHandler(username, password string) {

type Response struct {

Token string `json:"token"`

}

var resp Response

response, err := hm.client.R().

SetHeader("Content-Type", "application/json").

SetBody(map[string]interface{}{

"username": username,

"password": password,

}).

SetResult(&resp).

Post(fmt.Sprintf("https://%s/login", hm.addr))

if err != nil {

log.Error().Err(err).Msg("cant request login")

return

}

if response.StatusCode() != http.StatusOK {

log.Error().Msg(string(response.Body()))

return

}

utils.WriteToken(resp.Token)

}

func (hm \*HandlersManager) ValidateTokenHandler() bool {

token := utils.GetToken()

response, err := hm.client.R().

SetHeader("Authorization", token).

Get(fmt.Sprintf("https://%s/token", hm.addr))

if err != nil {

log.Error().Err(err).Msg("cant request token validation")

return false

}

return response.StatusCode() == http.StatusOK

}

"

"C:\Go-Projects\Small-Messenger\internal\controller\client\websocket.go":

"package controller

import (

"context"

"fmt"

chatSvc "messenger/internal/service/client"

"messenger/internal/utils"

"sync"

"github.com/gobwas/ws"

"github.com/rs/zerolog/log"

)

func (hm \*HandlersManager) NewChatHandler(recipient string) {

conn, \_, \_, err := ws.Dialer{

Header: ws.HandshakeHeaderHTTP{

"Authorization": []string{utils.GetToken()},

"Recipient": []string{recipient},

},

}.Dial(context.TODO(), fmt.Sprintf("wss://%s/ws/chats", hm.addr))

if err != nil {

log.Error().Err(err).Msg("cant request create new chat")

return

}

var wg sync.WaitGroup

wg.Add(2)

go chatSvc.Reader(&wg, conn)

go chatSvc.Writer(&wg, conn)

wg.Wait()

}

func (hm \*HandlersManager) JoinChatHandler(chatId string) {

conn, \_, \_, err := ws.Dialer{

Header: ws.HandshakeHeaderHTTP{

"Authorization": []string{utils.GetToken()},

},

}.Dial(context.TODO(), fmt.Sprintf("wss://%s/ws/chats/%s", hm.addr, chatId))

if err != nil {

log.Error().Err(err).Msg("cant request join chat")

return

}

var wg sync.WaitGroup

wg.Add(2)

go chatSvc.Reader(&wg, conn)

go chatSvc.Writer(&wg, conn)

wg.Wait()

}

"

"C:\Go-Projects\Small-Messenger\internal\controller\server\chat.go":

"package handlers

import (

"fmt"

"messenger/internal/models"

"net/http"

"strconv"

"github.com/gin-gonic/gin"

"github.com/pkg/errors"

"github.com/rs/zerolog/log"

)

func (cr \*Controller) GetChats(c \*gin.Context) {

id := c.GetInt("id")

chats, err := cr.repo.GetAllUserChats(id)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling GetAllUserChats")).Msg("")

c.String(http.StatusInternalServerError, err.Error())

return

}

c.JSON(http.StatusOK, chats)

}

func (cr \*Controller) GetLastChatMessages(c \*gin.Context) {

chatId, err := strconv.Atoi(c.Params.ByName("id"))

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "parsing string to int")).Msg("")

c.String(http.StatusBadRequest, err.Error())

return

}

messages, err := cr.repo.GetLastChatMessages(chatId)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling GetLastChatMessages")).Msg("")

c.String(http.StatusInternalServerError, err.Error())

return

}

c.JSON(http.StatusOK, messages)

}

func (cr \*Controller) NewChat(c \*gin.Context) {

cr.cs.Upgrade(c)

senderId := c.GetInt("id")

sender, err := cr.repo.GetUserById(senderId)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling GetUserById")).Msg("")

return

}

recipientUsername := c.GetHeader("Recipient")

recipient, err := cr.repo.GetUserByName(recipientUsername)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling GetUserByName")).Msg("")

return

}

chatId, err := cr.repo.CreateChat(models.Chat{

Name: fmt.Sprintf("%s and %s", sender.Username, recipient.Username),

})

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling CreateChat")).Msg("")

return

}

err = cr.repo.AddChatMember(models.ChatMember{

ChatId: chatId,

UserId: sender.Id,

})

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling AddChatMember")).Msg("")

return

}

err = cr.repo.AddChatMember(models.ChatMember{

ChatId: chatId,

UserId: recipient.Id,

})

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling AddChatMember")).Msg("")

return

}

cr.cs.JoinChat(senderId, chatId)

}

func (cr \*Controller) JoinChat(c \*gin.Context) {

err := cr.cs.Upgrade(c)

if err != nil {

c.String(http.StatusInternalServerError, err.Error())

log.Error().Stack().Err(errors.Wrap(err, "upgrading to websocket")).Msg("")

return

}

chatId, err := strconv.Atoi(c.Params.ByName("id"))

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "parsing string to int")).Msg("")

return

}

senderId := c.GetInt("id")

cr.cs.JoinChat(senderId, chatId)

}

"

"C:\Go-Projects\Small-Messenger\internal\controller\server\controller.go":

"package handlers

import (

"messenger/internal/interfaces"

"messenger/internal/service/service"

)

type Controller struct {

repo interfaces.Repository

cs \*service.ChatService

}

func New(repo interfaces.Repository, cs \*service.ChatService) \*Controller {

return &Controller{repo: repo, cs: cs}

}

"

"C:\Go-Projects\Small-Messenger\internal\controller\server\user.go":

"package handlers

import (

"messenger/internal/authentication"

"messenger/internal/models"

"net/http"

"github.com/gin-gonic/gin"

"github.com/pkg/errors"

"github.com/rs/zerolog/log"

"golang.org/x/crypto/bcrypt"

)

func (cr \*Controller) Register(c \*gin.Context) {

user := models.User{}

err := c.ShouldBindJSON(&user)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "binding json")).Msg("")

c.String(http.StatusBadRequest, err.Error())

return

}

securedPass, err := bcrypt.GenerateFromPassword([]byte(user.Password), bcrypt.DefaultCost)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "generating hashed password")).Msg("")

c.String(http.StatusInternalServerError, err.Error())

return

}

user.Password = string(securedPass)

userId, err := cr.repo.CreateUser(user)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling CreateUser")).Msg("")

c.String(http.StatusInternalServerError, err.Error())

return

}

token, err := authentication.CreateToken(userId)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling CreateToken")).Msg("")

c.String(http.StatusInternalServerError, err.Error())

return

}

c.JSON(http.StatusOK, gin.H{"token": token})

}

func (cr \*Controller) Login(c \*gin.Context) {

user := models.User{}

c.ShouldBindJSON(&user)

scannedUser, err := cr.repo.GetUserByName(user.Username)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling GetUserByName")).Msg("")

c.String(http.StatusInternalServerError, err.Error())

return

}

err = bcrypt.CompareHashAndPassword([]byte(scannedUser.Password), []byte(user.Password))

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "comparing password to hashed password")).Msg("")

c.String(http.StatusForbidden, err.Error())

return

}

token, err := authentication.CreateToken(scannedUser.Id)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling CreateToken")).Msg("")

c.String(http.StatusInternalServerError, err.Error())

return

}

c.JSON(http.StatusOK, gin.H{"token": token})

}

func (cr \*Controller) ValidateTokenHandler(c \*gin.Context) {

c.JSON(http.StatusOK, "")

}

"

"C:\Go-Projects\Small-Messenger\internal\iface\manager.go":

"package iface

import (

controller "messenger/internal/controller/client"

)

type InterfaceManager struct {

hm \*controller.HandlersManager

}

func NewInterfaceManager() \*InterfaceManager {

return &InterfaceManager{hm: controller.GetHandlersManager()}

}

func (im \*InterfaceManager) RunApp() {

im.switchToMainMenu()

}

"

"C:\Go-Projects\Small-Messenger\internal\iface\switchers.go":

"package iface

import (

"fmt"

"os"

"github.com/rs/zerolog/log"

)

func (im \*InterfaceManager) switchToStartMenu() {

str := `Register Menu

1) Register

2) Login

3) Quit`

fmt.Println(str)

var option int

for {

\_, err := fmt.Scanf("%d\n", &option)

if err != nil {

log.Error().Err(err).Msg("incorrect input")

continue

}

switch option {

case 1:

im.switchToRegistrationMenu()

case 2:

im.switchToLoginMenu()

case 3:

os.Exit(0)

default:

log.Info().Msg(fmt.Sprintf("there is no option %d)", option))

continue

}

break

}

}

func (im \*InterfaceManager) switchToRegistrationMenu() {

fmt.Println("Please enter login and password in format login password")

var username, password string

for {

\_, err := fmt.Scanf("%s %s\n", &username, &password)

if err != nil {

log.Error().Err(err).Msg("incorrect input please check correct format")

continue

}

break

}

im.hm.RegistrationHandler(username, password)

im.switchToMainMenu()

}

func (im \*InterfaceManager) switchToLoginMenu() {

fmt.Println("Please enter login and password in format login password")

var username, password string

for {

\_, err := fmt.Scanf("%s %s\n", &username, &password)

if err != nil {

log.Error().Err(err).Msg("incorrect input please check correct format")

continue

}

break

}

im.hm.LoginHandler(username, password)

im.switchToMainMenu()

}

func (im \*InterfaceManager) switchToChatsMenu() {

str := "Chats Menu\n1) New chat\n"

chats := im.hm.GetChatsHandler()

options := make(map[int]string)

var i int = 2

for id, name := range chats {

str += fmt.Sprintf("%d) Chat : %s\n", i, name)

options[i] = id

i++

}

str += fmt.Sprintf("%d) Quit", i)

fmt.Println(str)

var option int

for {

\_, err := fmt.Scanf("%d\n", &option)

if err != nil {

log.Error().Err(err).Msg("incorrect input")

continue

}

switch option {

case 1:

im.switchToChatCreationChatMenu()

case i:

os.Exit(0)

default:

if option > i-1 || option < 1 {

log.Info().Msg(fmt.Sprintf("there is no option %d)", option))

continue

}

im.switchToJoinChatMenu(options[option])

}

break

}

}

func (im \*InterfaceManager) switchToChatCreationChatMenu() {

fmt.Println("Please enter recipient's username\nTo exit write \"/back\"")

var username string

for {

\_, err := fmt.Scanf("%s\n", &username)

if err != nil {

log.Error().Err(err).Msg("incorrect input")

continue

}

if username == "/back" {

im.switchToMainMenu()

}

break

}

im.switchToNewChatMenu(username)

}

func (im \*InterfaceManager) switchToNewChatMenu(username string) {

str := fmt.Sprintf("Chat with %s", username)

fmt.Println(str)

im.hm.NewChatHandler(username)

}

func (im \*InterfaceManager) switchToJoinChatMenu(chatId string) {

messages := im.hm.ChatHistoryHandler(chatId)

for i := len(messages) - 1; i >= 0; i-- {

fmt.Printf("%s : %s\n", messages[i].SenderUsername, messages[i].Message)

}

im.hm.JoinChatHandler(chatId)

}

func (im \*InterfaceManager) switchToMainMenu() {

if im.hm.ValidateTokenHandler() {

im.switchToChatsMenu()

} else {

im.switchToStartMenu()

}

}

"

"C:\Go-Projects\Small-Messenger\internal\interfaces\interfaces.go":

"package interfaces

import (

"messenger/internal/models"

)

type Repository interface {

CreateUser(user models.User) (int, error)

GetUserByName(username string) (models.User, error)

GetUserIdByName(username string) (id int, err error)

GetUserById(id int) (models.User, error)

GetUsernameById(id int) (string, error)

CreateChat(chat models.Chat) (id int, err error)

GetAllUserChats(userId int) (map[int]string, error)

AddChatMember(cm models.ChatMember) error

GetAllChatMembers(chatId int) ([]int, error)

SaveMessage(msg models.Message) error

GetLastChatMessages(chatId int) ([]models.ShortMessage, error)

}

"

"C:\Go-Projects\Small-Messenger\internal\middleware\auth.go":

"package middleware

import (

"messenger/internal/authentication"

"net/http"

"github.com/gin-gonic/gin"

)

func Authentication() gin.HandlerFunc {

return func(c \*gin.Context) {

token := c.GetHeader("Authorization")

claims, err := authentication.ValidateToken(token)

if err != nil {

c.AbortWithError(http.StatusForbidden, err)

return

}

c.Set("id", claims.Id)

c.Next()

}

}

"

"C:\Go-Projects\Small-Messenger\internal\models\db.go":

"package models

import "time"

type User struct {

Id int `json:"id"`

Username string `json:"username"`

Password string `json:"password"`

}

type Message struct {

Id int

ChatId int

UserId int

Date time.Time

Message string

}

type ShortMessage struct {

SenderUsername string `json:"sender\_username"`

Message string `json:"message"`

}

type Chat struct {

Id int

Name string

}

type ChatMember struct {

Id int

ChatId int

UserId int

}

"

"C:\Go-Projects\Small-Messenger\internal\repository\chats.go":

"package repository

import "messenger/internal/models"

func (dm \*repository) CreateChat(chat models.Chat) (id int, err error) {

err = dm.db.QueryRow("insert into chats (name) values ($1) returning id", chat.Name).Scan(&id)

return

}

func (dm \*repository) GetAllUserChats(userId int) (map[int]string, error) {

req := `SELECT chat\_members.chat\_id, chats.name

FROM chats

JOIN chat\_members ON chat\_members.chat\_id = chats.id

WHERE chat\_members.user\_id = $1`

rows, err := dm.db.Query(req, userId)

if err != nil {

return nil, err

}

result := make(map[int]string)

for rows.Next() {

var (

id int

name string

)

err = rows.Scan(&id, &name)

if err != nil {

return nil, err

}

result[id] = name

}

return result, nil

}

"

"C:\Go-Projects\Small-Messenger\internal\repository\chat\_members.go":

"package repository

import "messenger/internal/models"

func (dm \*repository) AddChatMember(cm models.ChatMember) error {

\_, err := dm.db.Exec("insert into chat\_members (chat\_id,user\_id) values ($1,$2)", cm.ChatId, cm.UserId)

return err

}

func (dm \*repository) GetAllChatMembers(chatId int) ([]int, error) {

rows, err := dm.db.Query("select user\_id from chat\_members where chat\_id = $1", chatId)

if err != nil {

return nil, err

}

var result []int

for rows.Next() {

var userId int

err := rows.Scan(&userId)

if err != nil {

return nil, err

}

result = append(result, userId)

}

return result, nil

}

"

"C:\Go-Projects\Small-Messenger\internal\repository\messages.go":

"package repository

import (

"messenger/internal/models"

)

func (dm \*repository) SaveMessage(msg models.Message) error {

\_, err := dm.db.Exec("insert into messages (chat\_id,user\_id,message) values ($1,$2,$3)", msg.ChatId, msg.UserId, msg.Message)

return err

}

func (dm \*repository) GetLastChatMessages(chatId int) ([]models.ShortMessage, error) {

req := `SELECT users.username, messages.message

FROM messages

JOIN users ON messages.user\_id = users.id

WHERE messages.chat\_id = $1

ORDER BY messages.id DESC

LIMIT 20`

rows, err := dm.db.Query(req, chatId)

if err != nil {

return nil, err

}

var messages []models.ShortMessage

for rows.Next() {

var senderUsername, msg string

err = rows.Scan(&senderUsername, &msg)

if err != nil {

return nil, err

}

messages = append(messages, models.ShortMessage{SenderUsername: senderUsername, Message: msg})

}

return messages, nil

}

"

"C:\Go-Projects\Small-Messenger\internal\repository\repository.go":

"package repository

import (

"messenger/internal/interfaces"

"github.com/jmoiron/sqlx"

\_ "github.com/lib/pq"

)

type repository struct {

db \*sqlx.DB

}

func New(db \*sqlx.DB) interfaces.Repository {

return &repository{db: db}

}

"

"C:\Go-Projects\Small-Messenger\internal\repository\users.go":

"package repository

import (

"messenger/internal/models"

\_ "github.com/lib/pq"

"github.com/pkg/errors"

)

func (dm \*repository) CreateUser(user models.User) (int, error) {

var id int

err := dm.db.QueryRow("insert into users (username,password) values ($1,$2) returning id", user.Username, user.Password).Scan(&id)

return id, errors.Wrap(err, "inserting into users")

}

func (dm \*repository) GetUserByName(username string) (models.User, error) {

scannedUser := models.User{}

err := dm.db.QueryRow("select \* from users where username = $1", username).Scan(&scannedUser.Id, &scannedUser.Username, &scannedUser.Password)

return scannedUser, errors.Wrap(err, "selecting from users by username")

}

func (dm \*repository) GetUserIdByName(username string) (id int, err error) {

err = dm.db.QueryRow("select id from users where username = $1", username).Scan(&id)

return id, errors.Wrap(err, "selecting id from users users by username")

}

func (dm \*repository) GetUserById(id int) (models.User, error) {

user := models.User{}

err := dm.db.QueryRow("select \* from users where id = $1", id).Scan(&user.Id, &user.Username, &user.Password)

return user, errors.Wrap(err, "selecting from users by id")

}

func (dm \*repository) GetUsernameById(id int) (string, error) {

var username string

err := dm.db.QueryRow("select username from users where id = $1", id).Scan(&username)

return username, errors.Wrap(err, "selecting username from users by id")

}

"

"C:\Go-Projects\Small-Messenger\internal\service\client\chat.go":

"package websocket

import (

"bufio"

"fmt"

"net"

"os"

"sync"

"github.com/gobwas/ws/wsutil"

"github.com/rs/zerolog/log"

)

func Reader(wg \*sync.WaitGroup, conn net.Conn) {

defer wg.Done()

for {

msg, err := wsutil.ReadServerText(conn)

if err != nil {

log.Error().Err(err).Msg("cant read message from conn")

return

}

fmt.Println(string(msg))

}

}

func Writer(wg \*sync.WaitGroup, conn net.Conn) {

defer wg.Done()

r := bufio.NewReader(os.Stdin)

for {

r.Reset(os.Stdin)

msg, err := r.ReadString('\n')

if err != nil {

log.Error().Err(err).Msg("cant read message from console")

return

}

fmt.Println()

err = wsutil.WriteClientText(conn, []byte(msg))

if err != nil {

log.Error().Err(err).Msg("cant write message to conn")

return

}

}

}

"

"C:\Go-Projects\Small-Messenger\internal\service\service\chat.go":

"package service

import (

"messenger/internal/models"

"github.com/gobwas/ws/wsutil"

"github.com/pkg/errors"

"github.com/rs/zerolog/log"

)

func (cs \*ChatService) JoinChat(id, chatId int) {

cs.startReader(id, chatId)

cs.startWriter(id)

}

func (wm \*ChatService) startReader(id, chatId int) {

go func() {

cm := wm.getChatMember(id)

for {

msg, err := wsutil.ReadClientText(cm.conn)

if err != nil {

wm.deleteChatMember(id)

log.Error().Stack().Err(errors.Wrap(err, "reading client message")).Msg("")

return

}

wm.broadcast(id, chatId, msg)

errCh := make(chan error)

go func(errCh chan<- error) {

err := wm.repo.SaveMessage(models.Message{

ChatId: chatId,

UserId: id,

Message: string(msg),

})

errCh <- err

}(errCh)

err = <-errCh

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "saving message to db")).Msg("")

return

}

}

}()

}

func (wm \*ChatService) startWriter(id int) {

go func() {

cm := wm.getChatMember(id)

// type Message struct {

// senderUsername string

// message string

// }

// senderUsername, err := wm.dm.GetUsernameById(id)

// if err != nil {

// log.Error().Err(err).Msg("cant get username from db")

// return

// }

for {

// message := Message{senderUsername: senderUsername}

msg := <-cm.out

// message.message=string(msg)

err := wsutil.WriteServerText(cm.conn, msg)

if err != nil {

wm.deleteChatMember(id)

log.Error().Stack().Err(errors.Wrap(err, "writing message to client")).Msg("")

return

}

}

}()

}

func (wm \*ChatService) broadcast(id, chatId int, msg []byte) {

members, err := wm.repo.GetAllChatMembers(chatId)

if err != nil {

log.Error().Stack().Err(errors.Wrap(err, "calling GetAllChatMembers")).Msg("")

return

}

for \_, recipientId := range members {

if recipientId != id && wm.isConnected(recipientId) {

recipient := wm.getChatMember(recipientId)

recipient.out <- msg

}

}

}

"

"C:\Go-Projects\Small-Messenger\internal\service\service\chat\_service.go":

"package service

import (

"messenger/internal/interfaces"

"net"

"sync"

)

type ChatService struct {

repo interfaces.Repository

chatMembers map[int]\*chatMember

mu sync.Mutex

}

type chatMember struct {

conn net.Conn

out chan []byte

}

func New(repo interfaces.Repository) \*ChatService {

return &ChatService{repo: repo, chatMembers: make(map[int]\*chatMember)}

}

"

"C:\Go-Projects\Small-Messenger\internal\service\service\members.go":

"package service

import "net"

func (cs \*ChatService) addChatMember(id int, conn net.Conn) {

cs.mu.Lock()

cs.chatMembers[id] = &chatMember{conn: conn, out: make(chan []byte, 5)}

cs.mu.Unlock()

}

func (cs \*ChatService) getChatMember(id int) \*chatMember {

cs.mu.Lock()

cm := cs.chatMembers[id]

cs.mu.Unlock()

return cm

}

func (cs \*ChatService) deleteChatMember(id int) {

cs.mu.Lock()

if \_, ok := cs.chatMembers[id]; ok {

cs.chatMembers[id].conn.Close()

close(cs.chatMembers[id].out)

delete(cs.chatMembers, id)

}

cs.mu.Unlock()

}

func (cs \*ChatService) isConnected(id int) bool {

cs.mu.Lock()

\_, ok := cs.chatMembers[id]

cs.mu.Unlock()

return ok

}

"

"C:\Go-Projects\Small-Messenger\internal\service\service\websocket.go":

"package service

import (

"github.com/gin-gonic/gin"

"github.com/gobwas/ws"

)

func (cs \*ChatService) Upgrade(c \*gin.Context) error {

conn, \_, \_, err := ws.UpgradeHTTP(c.Request, c.Writer)

if err != nil {

return err

}

id := c.GetInt("id")

cs.addChatMember(id, conn)

return nil

}

"

"C:\Go-Projects\Small-Messenger\internal\utils\token\_env.go":

"package utils

import (

"fmt"

"os"

"github.com/joho/godotenv"

"github.com/rs/zerolog/log"

)

func WriteToken(token string) {

envFile := "token.env"

file, err := os.OpenFile(envFile, os.O\_TRUNC|os.O\_CREATE|os.O\_WRONLY, 0644)

if err != nil {

log.Error().Err(err).Msg("")

return

}

defer file.Close()

envLine := fmt.Sprintf("REGISTRATION\_TOKEN=%s\n", token)

if \_, err := file.WriteString(envLine); err != nil {

log.Error().Err(err).Msg("cant write to token.env")

return

}

}

func GetToken() string {

err := godotenv.Load("token.env")

if err != nil {

log.Error().Err(err).Msg("cant load token.env")

return ""

}

return os.Getenv("REGISTRATION\_TOKEN")

}

