**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 домашнему заданию

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| студент группы ИУ5-34Б | преподаватель каф. ИУ5 |
| Шараев В. А./  Байдаков В.М. | Нардид Анатолий  Николаевич |

Москва, 2024 г

**Постановка задачи:**

Ginol-мессенджер

Разработать web-приложение для обмена сообщениями между пользователями с использованием языка программирования Golang, фреймворков Gin, React, технологии Websocket для установления соединения и обмена сообщениями между двумя пользователями и GORM как ORM для системы PostgreSQL для взаимодействия с базой данных, в которой содержится информация о пользователях.

В проекте присутствует страница логина и регистрации пользователей, страница ввода логина пользователя, с которым осуществляется обмен сообщениями и окно чата, в котором и общаются пользователи.

**Инструкция по использованию:**

При открытии web-приложения открывается окно с кнопками регистрации и логина. Если пользователь первый раз использует приложение, ему нужно зарегистрироваться (его данные попадают в базу данных пользователей), иначе пользователь заходит по своему логину и паролю. Далее приложение запрашивает ник собеседника, с которым пользователь планирует общаться.

Если такой пользователь отсутствует в базе данных, об этом будет выведено соответствующее предупреждение. Если же собеседник найден, то открывается окно чата, где пользователи могут начать обмениваться сообщениями.

**Текст программы:**

Backend:

Backend/routes/websocket.go

package routes

import (

"log"

"time"

"net/http"

"github.com/gin-gonic/gin"

"github.com/gorilla/websocket"

)

type Message struct {

Sender string `json:"sender"`

Text string `json:"text"`

Time time.Time `json:"time"`

Target string `json:"target"`

}

type ChatInstance struct{

Username string `json:"username"`

FriendUsername string` json:"friend"`

}

var clients = make(map[string]\*websocket.Conn) // активные ws подключения

var upgrader = websocket.Upgrader{ //

ReadBufferSize: 1024,

WriteBufferSize: 1024,

CheckOrigin: func(r \*http.Request) bool {

return true

},

}

func HandleWebSocket(c \*gin.Context, chatInst ChatInstance) { // установка WebSocket соединения

username := c.Query("username")

friend := c.Query("friend")

if username == "" || friend == "" {

c.JSON(http.StatusBadRequest, gin.H{"error": "Missing username or friend"})

return

}

ws, err := upgrader.Upgrade(c.Writer, c.Request, nil)

if err != nil {

log.Println("Error upgrading to WebSocket:", err)

return

}

defer ws.Close()

clients[username] = ws // сохраняем соединение

log.Printf("WebSocket connection established for user: %s", username)

for {

var msg Message

err := ws.ReadJSON(&msg)

if err != nil {

log.Printf("Error reading message from %s: %v", username, err)

delete(clients, username)

break

}

msg.Sender = username

msg.Time = time.Now()

msg.Target = friend

if friendConn, ok := clients[friend]; ok {

err = friendConn.WriteJSON(msg)

if err != nil {

log.Printf("Error sending message to %s: %v", friend, err)

delete(clients, friend)

}

} else {

log.Printf("Friend %s is not connected", friend)

}

}

}

backend/user/auth.go

package user

import (

"os"

"golang.org/x/crypto/bcrypt"

)

func (u \*Credentials) Crypt() error {

hashedPassword, err := bcrypt.GenerateFromPassword([]byte(u.Password), bcrypt.DefaultCost)

if err != nil {

return err

}

u.Password = string(hashedPassword)

return nil

}

func (u \*Credentials) RegistrationHandler() (string, bool) {

u.Crypt()

return SendRequest(\*u, os.Getenv("SERVER\_REG\_API\_URL"))

}

backend/user/login.go

package user

import (

"bytes"

"encoding/json"

"io"

"log"

"net/http"

"os"

)

type Credentials struct {

Username string `json:"username"`

Password string `json:"password"`

}

type Answer struct {

LoggedUser string `json:"loggedUser"`

FriendNickname string `json:"friendNickname"`

}

func SendCheckRequest(friendNickname string) (bool, error) {

payload := Answer{

FriendNickname: friendNickname,

}

jsonData, err := json.Marshal(payload)

if err != nil {

log.Println("error marshaling JSON:", err)

return false, err

}

url := os.Getenv("SERVER\_CHECK\_NICKNAME\_API\_URL")

if url == "" {

log.Println("api server check connection unable")

return false, nil

}

req, err := http.NewRequest("POST", url, bytes.NewBuffer(jsonData))

if err != nil {

log.Println("error POST request:", err)

return false, err

}

req.Header.Set("Content-Type", "application/json")

client := &http.Client{}

resp, err := client.Do(req)

if err != nil {

log.Println("error of sending msg:", err)

return false, err

}

defer resp.Body.Close()

body, err := io.ReadAll(resp.Body)

if err != nil {

log.Println("error of reading:", err)

return false, err

}

log.Printf("api check nickname: %s", body)

var response struct {

Exists bool `json:"exists"`

}

if err := json.Unmarshal(body, &response); err != nil {

log.Println("error unmarshal check nickname:", err)

return false, err

}

return response.Exists, nil

}

func (c \*Credentials) LoginHandler() (bool, string) {

log.Printf("Attempting login with username: %s and password: %s", c.Username, c.Password)

token, success := SendRequest(\*c, os.Getenv("SERVER\_LOGIN\_API\_URL"))

return success, token

}

func SendRequest(credentials Credentials, url string) (string, bool) {

jsonData, err := json.Marshal(credentials)

if err != nil {

log.Fatalln("Error marshalling credentials:", err)

return "", false

}

req, err := http.NewRequest("POST", url, bytes.NewBuffer(jsonData))

if err != nil {

log.Fatalln("Error creating POST request:", err)

return "", false

}

req.Header.Set("Content-Type", "application/json")

client := &http.Client{}

resp, err := client.Do(req)

if err != nil {

log.Fatalln("Error sending request:", err)

return "", false

}

defer resp.Body.Close()

body, err := io.ReadAll(resp.Body)

if err != nil {

log.Fatalln("Error reading response:", err)

return "", false

}

log.Printf("Response from external API: %s", body)

var result map[string]interface{}

if err := json.Unmarshal(body, &result); err != nil {

log.Fatalf("Error parsing JSON response: %s", err)

return "", false

}

// Извлечение токена

token, tokenExists := result["token"].(string)

// Проверка "bool" поля

success := false

if val, ok := result["bool"]; ok {

if valStr, isString := val.(string); isString && valStr == "true" {

success = true

}

if valBool, isBool := val.(bool); isBool && valBool {

success = true

}

}

if success && tokenExists {

return token, true

}

return "", false

}

backend/main.go

package main

import (

"log"

"net/http"

"github.com/gin-gonic/gin"

"github.com/joho/godotenv"

"github.com/kusneid/Ginol/backend/routes"

"github.com/kusneid/Ginol/backend/user"

)

func main() {

r := gin.Default()

if err := godotenv.Load(); err != nil {

log.Fatal("Error loading .env file")

}

r.POST("/api/register", func(c \*gin.Context) {

var userAdded user.Credentials

if err := c.ShouldBindJSON(&userAdded); err != nil {

c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid data"})

}

log.Println("Registration API call handled")

regResult, token := userAdded.RegistrationHandler()

log.Println("Registration result:", regResult)

c.JSON(http.StatusOK, gin.H{"bool": true, "username": userAdded.Username, "token": token})

})

r.POST("/api/login", func(c \*gin.Context) {

var credentials user.Credentials

if err := c.ShouldBindJSON(&credentials); err != nil {

c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid data"})

return

}

log.Println("Login API call handled")

authResult, token := credentials.LoginHandler()

log.Println("Auth result:", authResult)

if !authResult {

c.JSON(http.StatusConflict, gin.H{"loginStatus": "false"})

return

}

c.JSON(http.StatusOK, gin.H{"bool": true, "username": credentials.Username, "token": token})

})

r.POST("/api/check-nickname", func(c \*gin.Context) {

var union user.Answer

if err := c.ShouldBindJSON(&union); err != nil {

c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid data"})

return

}

if union.FriendNickname == union.LoggedUser {

log.Fatalln("can't connect same accounts sorry bro")

return

}

log.Println("check nickname api handled")

//fmt.Println("ERR:", union.FriendNickname)

value, err := user.SendCheckRequest(union.FriendNickname)

if err != nil {

c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid data"})

}

if value {

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

} else {

c.JSON(http.StatusBadRequest, gin.H{"exists": false})

}

})

var chatInst routes.ChatInstance

r.POST("/api/chat-reg", func(c \*gin.Context) {

if err := c.ShouldBindJSON(&chatInst); err != nil {

c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid data"})

return

}

log.Printf("Chat registration: Username=%s, Friend=%s", chatInst.Username, chatInst.FriendUsername)

c.JSON(http.StatusOK, gin.H{"message": "Chat registration successful"})

})

r.GET("/ws/chat", func(c \*gin.Context){

routes.HandleWebSocket(c, chatInst)

})

r.Run(":8080")

}

Frontend:

frontend/src/components/chatpage.tsx

import React, { useState, useEffect, useRef } from 'react';

import { useLocation } from 'react-router-dom';

import '../style.css';

interface Message {

id: number;

username: string;

text: string;

time: string;

}

const ChatPage: React.FC = () => {

const [messages, setMessages] = useState<Message[]>([]);

const [inputText, setInputText] = useState('');

const location = useLocation();

const { username, friend, userToken, friendToken } = location.state as {

username: string;

friend: string;

userToken: string;

friendToken: string;

};

const chatBoxRef = useRef<HTMLDivElement>(null);

const [socket, setSocket] = useState<WebSocket | null>(null);

useEffect(() => {

const ws = new WebSocket(`ws://localhost:8080/ws?token=${userToken}`);

setSocket(ws);

ws.onmessage = (event) => {

const message: Message = JSON.parse(event.data);

if ([message.username, friend].includes(username) && [username, message.username].includes(friend)) {

setMessages((prevMessages) => [...prevMessages, message]);

}

};

ws.onclose = () => {

console.log('WebSocket connection closed');

};

return () => {

ws.close();

};

}, [userToken, username, friend]);

useEffect(() => {

chatBoxRef.current?.scrollTo({

top: chatBoxRef.current.scrollHeight,

behavior: 'smooth',

});

}, [messages]);

const handleSendMessage = () => {

if (inputText.trim() && socket) {

const newMessage: Message = {

id: Date.now(),

username,

text: inputText,

time: new Date().toISOString(),

};

socket.send(

JSON.stringify({

...newMessage,

token: userToken,

recipientToken: friendToken,

})

);

setMessages((prevMessages) => [...prevMessages, newMessage]);

setInputText('');

}

};

return (

<div className="container">

<h1 className="static-title">Chat with {friend}</h1>

<div className="chat-box" ref={chatBoxRef}>

{messages.map((msg, index) => (

<div key={index} className={`message ${msg.username === username ? 'sent' : 'received'}`}>

<strong>{msg.username}</strong>: {msg.text}

<span className="time">{new Date(msg.time).toLocaleTimeString().slice(0, 5)}</span>

</div>

))}

</div>

<div className="input-container">

<input

type="text"

placeholder="Write a message..."

value={inputText}

onChange={(e) => setInputText(e.target.value)}

onKeyDown={(e) => e.key === 'Enter' && handleSendMessage()}

/>

<button onClick={handleSendMessage} disabled={!inputText.trim()}>

Send

</button>

</div>

</div>

);

};

export default ChatPage;

frontend/src/components/connection.tsx

import React, { useState } from 'react';

import { useNavigate, useLocation } from 'react-router-dom';

import '../style.css';

function ConnectionPage() {

const [nickname, setNickname] = useState('');

const navigate = useNavigate();

const location = useLocation();

const { nickname: loggedUserNickname } = (location.state as { nickname: string }) || {};

const handleInputChange = (e: React.ChangeEvent<HTMLInputElement>) => {

setNickname(e.target.value);

};

const handleEnterChat = async () => {

if (nickname.trim()) {

try {

const token = localStorage.getItem('token');

if (!token) {

alert("User is not authenticated");

navigate('/login');

return;

}

const response = await fetch('/api/check-nickname', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${token}`,

},

body: JSON.stringify({

loggedUser: loggedUserNickname,

friendNickname: nickname

})

});

if (!response.ok) {

console.error(`Error: Received status ${response.status}`);

alert("An error occurred. Please try again.");

return;

}

const data = await response.json();

if (data.exists) {

navigate(`/chat/${nickname}`, { state: { username: loggedUserNickname, friend: nickname } });

} else {

alert("Nickname not found in the database.");

}

} catch (error) {

console.error("Error checking nickname:", error);

alert("An error occurred. Please try again.");

}

} else {

alert('Please enter a nickname.');

}

};

return (

<div className="container">

<h1 style={{ marginBottom: '0', fontSize: '2rem' }}>Enter nickname</h1>

<h1 style={{ marginTop: '0', fontSize: '2rem' }}>of your friend:</h1>

<input

type="text"

placeholder="Type your friend's nickname..."

value={nickname}

onChange={handleInputChange}

/>

<button onClick={handleEnterChat}>Enter chat</button>

</div>

);

}

export default ConnectionPage;

frontend/src/components/homepage.tsx

import React, { useState, useEffect } from 'react';

import { useNavigate } from 'react-router-dom';

import '../style.css';

const HomePage: React.FC = () => {

const navigate = useNavigate();

const [animateTitle, setAnimateTitle] = useState(true);

useEffect(() => {

const timer = setTimeout(() => setAnimateTitle(false), 3000); // Анимация 3 секунды

return () => clearTimeout(timer);

}, []);

return (

<div className="container">

<h1 className={animateTitle ? 'typing-animation' : 'static-title'}>Welcome to Ginol</h1>

<button onClick={() => navigate('/login')}>Login</button>

<button onClick={() => navigate('/register')}>Register</button>

</div>

);

};

export default HomePage;

frontend/src/components/loginpage.tsx

import React, { useState } from 'react';

import { useNavigate } from 'react-router-dom';

import '../style.css';

import { log } from 'console';

const Login: React.FC = () => {

const [username, setUsername] = useState('');

const [password, setPassword] = useState('');

const navigate = useNavigate();

const handleLogin = async (e: React.FormEvent) => {

e.preventDefault();

try {

const response = await fetch('/api/login', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ username, password }),

});

if (!response.ok) {

throw new Error(`HTTP error! status: ${response.status}`);

}

const data = await response.json();

if (data.bool === "true" || data.bool === true) {

localStorage.setItem('token', data.token);

navigate('/connection', { state: { nickname: data.username } });

} else {

alert('Login failed');

}

} catch (error) {

console.error('Error during login:', error);

alert('An error occurred during login. Please try again.');

}

};

return (

<div className="form-container">

<h1 className="form-title">Login</h1>

<form onSubmit={handleLogin}>

<input

type="text"

placeholder="Username"

value={username}

onChange={(e) => setUsername(e.target.value)}

className="form-input"

/>

<input

type="password"

placeholder="Password"

value={password}

onChange={(e) => setPassword(e.target.value)}

className="form-input"

/>

<button type="submit" className="form-button">Login</button>

</form>

</div>

);

};

export default Login;

frontend/src/components/registerpage.tsx

import React, { useState } from 'react';

import { useNavigate } from 'react-router-dom';

import '../style.css';

const RegisterPage: React.FC = () => {

const [username, setUsername] = useState('');

const [password, setPassword] = useState('');

const navigate = useNavigate();

const handleRegister = async () => {

const response = await fetch('http://localhost:3000/api/register', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ username, password }),

});

if (response.ok) {

const data = await response.json();

localStorage.setItem('token', data.token);

navigate('/connection', { state: { nickname: username } });

} else {

alert('Registration failed');

}

};

return (

<div className="container">

<h1>Registration</h1>

<input

type="text"

placeholder="Username"

value={username}

onChange={(e) => setUsername(e.target.value)}

/>

<input

type="password"

placeholder="Password"

value={password}

onChange={(e) => setPassword(e.target.value)}

/>

<button onClick={handleRegister}>Register</button>

</div>

);

};

export default RegisterPage;

frontend/src/app.tsx

import React from 'react';

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import HomePage from './components/homepage';

import LoginPage from './components/loginpage';

import RegisterPage from './components/registerpage';

import ConnectionPage from "./components/connection";

import ChatPage from "./components/chatpage";

const App: React.FC = () => {

return (

<Router>

<Routes>

<Route path="/" element={<HomePage />} />

<Route path="/login" element={<LoginPage />} />

<Route path="/register" element={<RegisterPage />} />

<Route path="/connection" element={<ConnectionPage />} />

<Route path="/chat/:nickname" element={<ChatPage />} />

</Routes>

</Router>

);

};

export default App;

frontend/src/index.tsx

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './app';

import './style.css'

const root = ReactDOM.createRoot(document.getElementById('root') as HTMLElement);

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

frontend/src/style.css

body {

font-family: 'Poppins', sans-serif;

display: flex;

justify-content: center;

align-items: center;

min-height: 100vh;

background: linear-gradient(130deg, #32324a, #e46060);

margin: 0;

}

.form-wrapper {

display: flex;

justify-content: center;

align-items: center;

min-height: 100vh;

width: 100%;

}

.form-container {

text-align: center;

padding: 2.5rem;

background-color: #1f1f2e;

border-radius: 16px;

box-shadow: 0 15px 35px rgba(0, 0, 0, 0.3);

max-width: 400px;

width: 100%;

transition: all 0.3s ease-in-out;

}

.form-title {

margin-bottom: 1.5rem;

font-size: 2rem;

font-weight: 700;

color: #ffffff;

}

.form-input {

width: 85%;

padding: 10px;

margin: 8px 0;

border: 1px solid #555555;

border-radius: 8px;

background-color: #2c2c3e;

color: #ffffff;

font-size: 0.9rem;

box-sizing: border-box;

transition: border-color 0.3s, box-shadow 0.3s;

box-shadow: inset 0 1px 4px rgba(0, 0, 0, 0.5);

outline: none;

}

.form-input:focus {

border-color: rgb(205, 109, 109);

box-shadow: 0 4px 8px rgba(255, 107, 107, 0.3);

}

.form-button {

width: 85%;

padding: 12px;

margin-top: 15px;

background-color: #ff6b6b;

color: white;

border: none;

border-radius: 25px;

font-size: 1.1rem;

font-weight: 600;

cursor: pointer;

transition: background-color 0.3s, transform 0.3s;

}

.form-button:hover {

background-color: #ff5b5b;

transform: scale(1.05);

}

.container {

text-align: center;

padding: 2.5rem;

background-color: #1f1f2e;

border-radius: 16px;

box-shadow: 0 15px 35px rgba(0, 0, 0, 0.3);

max-width: 500px;

width: 100%;

min-width: 320px;

transition: all 0.3s ease-in-out;

}

h1 {

margin-bottom: 1.5rem;

font-size: 2.5rem;

font-weight: 700;

color: #ffffff;

}

.typing-animation {

font-weight: 700;

color: #ffffff;

overflow: hidden;

border-right: 3px solid #ffffff;

white-space: nowrap;

width: 100%;

animation: typing 2.5s steps(20, end), blink-caret 0.5s step-end infinite;

font-size: 2.5rem;

}

.static-title {

font-weight: 700;

color: #ffffff;

font-size: 2.5rem;

}

@keyframes typing {

from {

width: 0;

}

to {

width: 100%;

}

}

@keyframes blink-caret {

50% {

border-color: transparent;

}

}

button {

width: 70%;

padding: 12px;

margin-top: 15px;

background-color: #ff6b6b;

color: white;

border: none;

border-radius: 25px;

font-size: 1.1rem;

font-weight: 600;

cursor: pointer;

transition: background-color 0.3s, transform 0.3s;

}

button:hover {

background-color: #ff5b5b;

transform: scale(1.05);

}

input[type="text"],

input[type="password"] {

width: 85%;

padding: 10px;

margin: 8px 0;

border: 1px solid #555555;

border-radius: 8px;

background-color: #2c2c3e;

color: #ffffff;

font-size: 0.9rem;

box-sizing: border-box;

transition: border-color 0.3s, box-shadow 0.3s;

box-shadow: inset 0 1px 4px rgba(0, 0, 0, 0.5);

outline: none;

}

input[type="text"]:focus,

input[type="password"]:focus {

border-color: rgb(205, 109, 109);

box-shadow: 0 4px 8px rgba(255, 107, 107, 0.3);

}

.footer {

margin-top: 20px;

font-size: 0.85rem;

color: #b0b0b0;

}

.chat-box {

width: 100%;

max-width: 400px;

height: 400px;

overflow-y: auto;

background-color: #1f1f2e;

padding: 1rem;

border-radius: 8px;

margin-bottom: 1rem;

display: flex;

flex-direction: column;

gap: 0.5rem;

align-items: center; /\* Center messages vertically in box \*/

}

.message {

color: #ffffff;

font-size: 1rem;

background: #2c2c3e;

padding: 10px;

border-radius: 8px;

width: fit-content;

max-width: 80%;

}

.message.sent {

align-self: flex-end;

background-color: #ff6b6b;

}

.message.received {

align-self: flex-start;

background-color: #2c2c3e;

}

.message strong {

color: #ff6b6b;

font-weight: 600;

}

.time {

font-size: 0.8rem;

color: #b0b0b0;

margin-left: 10px;

}

.input-container {

display: flex;

align-items: center;

gap: 0.5rem;

width: 100%;

max-width: 400px;

}

.input-container input[type="text"] {

flex: 1;

padding: 10px;

border-radius: 8px;

border: 1px solid #555555;

background-color: #2c2c3e;

color: #ffffff;

}

.input-container button {

padding: 10px 20px;

background-color: #ff6b6b;

border: none;

color: white;

border-radius: 8px;

cursor: pointer;

font-weight: 600;

transition: background-color 0.3s, transform 0.3s;

}

.input-container button:hover {

background-color: #ff5b5b;

}

.header {

display: flex;

align-items: center;

justify-content: space-between;

margin-bottom: 1rem;

}

.menu-button {

background: none;

border: none;

color: white;

font-size: 1.5rem;

cursor: pointer;

padding: 0.5rem;

}

.dropdown-menu {

position: absolute;

top: 3.5rem;

left: 1rem;

background-color: #2c2c3e;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);

display: flex;

flex-direction: column;

width: 150px;

z-index: 1;

}

.dropdown-menu button {

background: none;

border: none;

color: white;

padding: 10px;

cursor: pointer;

font-size: 1rem;

text-align: left;

}

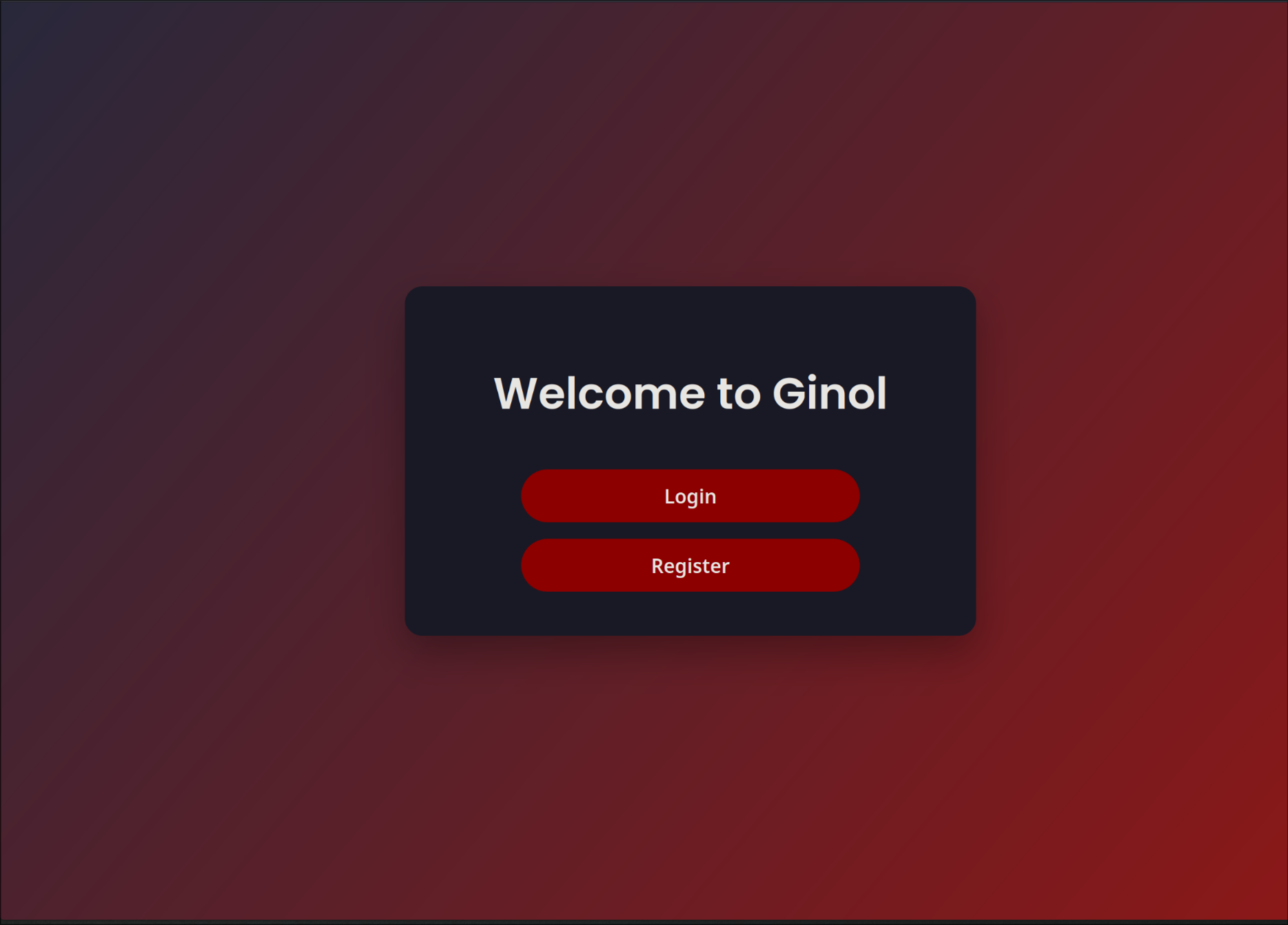
.dropdown-menu button:hover {

background-color: #3d3d4e;

}

Остальные файлы являются стандартными для фреймворков и не несут информации, полезной для понимания структуры проекта.

**Демонстрация работы приложения:**

****

