

Izveštaj o razvoju aplikacije Tressette

Teo Matošević

0036542778

Link od repozitorija: <https://github.com/TeoMatosevic/tressette>

1. Opis aplikacije i korištene tehnologije

Aplikacija **Tressette** je višekorisnička online implementacija klasične talijanske kartaške igre. Razvijena je koristeći sljedeće moderne web tehnologije:

- **Golang** - web server
- **WebSocket** za real-time komunikaciju
- **JavaScript** i **AJAX** za dinamično sučelje
- **REST API** za upravljanje rezultatima
- **CRUD operacije**
- **SQLite baza podataka** za perzistenciju podataka

2. WebSocket komunikacija i real-time interakcija

2.1 Struktura WebSocket Hub-a na backendu

Centralna komponenta za upravljanje WebSocket vezama implementirana je u `hub.go` :

```
type Hub struct {
    clients      map[*Client]bool
    lobbies      map[string][]*Client
    games        map[string]*game.Game
    clientToGame map[*Client]string
    processMessage chan clientMessage
    register     chan *Client
    unregister   chan *Client
    db           *database.Service
    clientMu     sync.RWMutex
    lobbyMu      sync.RWMutex
    gameMu       sync.RWMutex
    dbMu         sync.RWMutex
    rng          *rand.Rand
}
```

Ova struktura upravlja klijentima, sobama (lobby) i igrama, te osigurava sigurno ažuriranje stanja kroz `sync.RWMutex`.

2.2 Slanje i primanje poruka na backendu

Primanje poruka:

```
func (h *Hub) handleMessage(client *Client, msg protocol.Message) {
    switch msg.Type {
    case "create_game":
        h.handleCreateGame(client, msg)
    case "join_game":
        h.handleJoinGame(client, msg)
    case "play_card", "declare":
        h.handleGameAction(client, msg)
    case "ping":
        pongMsg, _ := protocol.NewMessage("pong", nil)
        client.send <- pongMsg
    default:
        log.Printf("Received unknown message type '%s' from client %s (%s)", msg.Type, client.ID, client.Name)
        h.sendErrorToClient(client, "Unknown message type.")
    }
}
```

Slanje poruka:

Postoji više varijanti slanja poruka ali priložen primjer je slanje poruke jednom klijentu.

```
func (h *Hub) sendMessageToClient(clientID string, message []byte) {
    h.clientMu.RLock()
    var targetClient *Client
    for client := range h.clients {
        if client.ID == clientID {
            targetClient = client
            break
        }
    }
    h.clientMu.RUnlock()

    if targetClient != nil {
        select {
        case targetClient.send <- message:
        default:
            log.Printf("Failed to send message to client %s (channel full or closed), initiating cleanup.", clientID)
            go func() {
                h.clientMu.RLock()
                _, stillConnected := h.clients[targetClient]
                h.clientMu.RUnlock()
                if stillConnected {
                    h.unregister <- targetClient
                }
            }()
        }
    } else {
        log.Printf("Could not find client %s to send message (already disconnected?).", clientID)
    }
}
```

3. REST API i CRUD operacije

3.1 Definiranje REST ruta

Definirane su samo rute za dohvat podataka, ostale rute bile bi nepotrebne u ovakvoj implementaciji.

```
func HandleRoutes(db *database.Service) {
    http.HandleFunc("/api/results/player/{name}", func(w http.ResponseWriter, r *http.Request) {
        GetResultsByPlayerHandler(db, w, r)
    })
    http.HandleFunc("/api/results", func(w http.ResponseWriter, r *http.Request) {
        GetResultsHandler(db, w, r)
    })
}
```

3.2 CRUD operacije nad bazom podataka

```
func (s *Service) GetAll() ([]GameResult, error) {
    s.m.Lock()
    defer s.m.Unlock()
    rows, err := s.db.Query("SELECT * FROM " + s.table_name)
    if err != nil {
        return nil, err
    }
    defer rows.Close()
    var results []GameResult
    for rows.Next() {
        var result GameResult
        err := rows.Scan(
            &result.ID, &result.CreatedAt, &result.Player1, &result.Player2,
            &result.Player3, &result.Player4, &result.Player1Team,
            &result.Player2Team, &result.Player3Team, &result.Player4Team,
            &result.Team1Score, &result.Team2Score)
        if err != nil {
            return nil, err
        }
        results = append(results, result)
    }
    return results, nil
}
```

4. Baza podataka i modeli

4.1 SQL shema

```
sqlStmt := `
CREATE TABLE IF NOT EXISTS tressette (
    id TEXT NOT NULL PRIMARY KEY,
    created_at TEXT,
    player1 TEXT,
    player2 TEXT,
    player3 TEXT,
    player4 TEXT,
    player1_team TEXT,
    player2_team TEXT,
    player3_team TEXT,
    player4_team TEXT,
    team1_score INTEGER,
    team2_score INTEGER
);`
```

5. Frontend implementacija

5.1 Uspostava WebSocket veze i slanje poruka

```

function connectWebSocket() {
  const wsProtocol = window.location.protocol === "https:" ? "wss:" : "ws:"
  const wsUrl = `${wsProtocol}://${window.location.host}/ws`

  ws = new WebSocket(wsUrl)

  ws.onopen = () => {
    console.log("WebSocket connection established")
    statusMessage.textContent = "Connected. Create or join a game."
  }

  ws.onmessage = (event) => {
    try {
      const message = JSON.parse(event.data)
      handleMessage(message)
    } catch (error) {
      console.error("Failed to parse message or handle:", error)
      statusMessage.textContent = "Error processing message from server."
    }
  }

  ws.onerror = (error) => {
    console.error("WebSocket error:", error)
    statusMessage.textContent = "WebSocket connection error."
    showSection("initial-section")
  }

  ws.onclose = () => {
    console.log("WebSocket connection closed")
    statusMessage.textContent = "Disconnected. Please refresh to reconnect."
    showSection("initial-section")
  }
}

```

5.2 Obrada dolaznih poruka

```

function handleMessage(message) {
  if (message.type !== "pong") {
    console.log("Handling message:", message)
  }
  switch (message.type) {
    case "game_created":
      handleGameCreated(message.payload)
      break
    case "lobby_update":
      handleLobbyUpdate(message.payload)
      break
    case "join_error":
      handleJoinError(message.payload)
      break
    case "game_start":
      handleGameStart(message.payload)
      break
    case "deal_hand":
      handleDealHand(message.payload)
      break
    case "your_turn":
      handleYourTurn()
      break
    case "game_state_update":
      handleGameState(message.payload)
      break
    case "you_played":
      handlePlayerPlayedCard(message.payload)
      break
    case "trick_end":
      handleTrickEnd(message.payload)
      break
    case "round_end":
      handleRoundEnd(message.payload)
      break
    case "game_over":
      handleGameOver(message.payload)
      break
    case "declaration_confirmation":
      handleDeclarationConfirmation(message.payload)
      break
    case "error":
      handleGenericError(message.payload)
      break
    case "pong":
      break
    default:
      console.warn("Received unhandled message type:", message.type)
  }
}

```

6. Sigurnosni mehanizmi i upravljanje greškama

6.1 Generiranje sigurnih game kodova

```

func (h *Hub) generateGameCode() string {
    const letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
    for {
        var sb strings.Builder
        for i := 0; i < gameCodeLength; i++ {
            sb.WriteByte(letters[h.rng.Intn(len(letters))])
        }
        code := sb.String()

        h.lobbyMu.RLock()
        _, lobbyExists := h.lobbies[code]
        h.lobbyMu.RUnlock()

        h.gameMu.RLock()
        _, gameExists := h.games[code]
        h.gameMu.RUnlock()

        if !lobbyExists && !gameExists {
            return code
        }
        log.Printf("Generated game code %s collided, retrying...", code)
    }
}

```

6.2 Obrada prekida veze

```

case client := <-h.unregister:
    h.clientMu.Lock()
    gameCode, inGameOrLobby := h.clientToGame[client]
    _, clientExists := h.clients[client]

    if clientExists {
        delete(h.clients, client)
        delete(h.clientToGame, client)
        close(client.send)
        log.Printf("Client %s (%s) disconnected", client.ID, client.Name)
    }
    h.clientMu.Unlock()

    if inGameOrLobby {
        h.lobbyMu.Lock()
        lobby, lobbyExists := h.lobbies[gameCode]
        if lobbyExists {
            newLobby := []*Client{}
            for _, c := range lobby {
                if c != client {
                    newLobby = append(newLobby, c)
                }
            }
            if len(newLobby) > 0 {
                h.lobbies[gameCode] = newLobby
                log.Printf("Client %s removed from lobby %s.", client.ID, gameCode)
                h.broadcastLobbyUpdate(gameCode, newLobby)
            } else {
                delete(h.lobbies, gameCode)
                log.Printf("Client %s left lobby %s. Lobby deleted.", client.ID, gameCode)
            }
            h.lobbyMu.Unlock()
        } else {
            h.lobbyMu.Unlock()

            h.gameMu.RLock()
            gameInstance, gameExists := h.games[gameCode]
            h.gameMu.RUnlock()

            if gameExists {
                log.Printf("Client %s was in game %s. Notifying game.", client.ID, gameCode)
                go gameInstance.HandlePlayerDisconnect(client.ID)
            } else {
                log.Printf("Client %s disconnected but was mapped to non-existent game/lobby code %s", client.ID, gameCode)
            }
        }
    } else if clientExists {
        log.Printf("Client %s disconnected before joining/creating a game.", client.ID)
    }
}

```

7. Prikaz dijelova sučelja i programskog koda

7.1 Primjer sučelja

- **Početni ekran:** Omogućuje unos imena i odabir radnje (kreiranje ili pridruživanje igri).

Create or Join a Game

Your Name:

Create Game

Points Goal:

Game Code (to Join):

Join Game

Choose Team:

Red

Blue

- **Igračka soba:** Prikazuje igrače u sobi i omogućuje početak igre.

Your turn!

Red: 0

0

Blue: 0

0

Points Goal: 51

test2

Show Rules

test4

test3

test1



Declare

8. Zaključak

Aplikacija **Tressette** integrira više modernih web tehnologija kako bi osigurala real-time interakciju i perzistenciju podataka. WebSocket protokol omogućuje brzu i dinamičnu komunikaciju između klijenata i servera, dok REST API i baza podataka omogućuju upravljanje rezultatima. Dinamičko sučelje, osigurano kroz JavaScript i AJAX, osigurava fluidno korisničko iskustvo. Sigurnosni mehanizmi i upravljanje greškama dodatno povećavaju pouzdanost sustava.