Sistem Distribuit Multiplayer

Gaming & Chat Platform

Documentație Tehnică

Pop Daniel

20 ianuarie 2026

Cuprins

1 Introducere

1.1 Prezentare generală

Acest proiect reprezintă o platformă distribuită de gaming și comunicare în timp real, implementată folosind arhitectura microserviciilor și tehnologii moderne de dezvoltare web. Sistemul permite utilizatorilor să joace jocuri multiplayer (Texas Hold'em Poker și Hangman), să comunice prin chat global, camere de chat și mesaje private, toate sincronizate în timp real între multiple instanțe de server.

1.2 Obiective

- Implementarea unei arhitecturi distribuite scalabile
- Sincronizare în timp real între clienți folosind Server-Sent Events (SSE)
- Distribuirea load-ului folosind Nginx ca load balancer
- Persistența datelor folosind MongoDB
- Comunicare asincronă prin RabbitMQ
- Pub/Sub pattern cu Redis pentru sincronizare distribuită

1.3 Tehnologii utilizate

1.3.1 Backend

- **Node.js** - Runtime environment pentru JavaScript
- **Koa.js** - Web framework modern și lightweight
- **MongoDB** - Bază de date NoSQL pentru persistența datelor
- **Redis** - In-memory data store pentru pub/sub
- **RabbitMQ** - Message broker pentru comunicare asincronă
- **JWT** - Autentificare bazată pe token-uri

1.3.2 Frontend

- **React 18** - Library pentru construirea interfețelor utilizator
- **React Router v6** - Routing pentru aplicație single-page
- **Bootstrap 5** - Framework CSS pentru design responsive
- **Vite** - Build tool și development server modern

1.3.3 Infrastructure

- **Docker** - Containerizare pentru portabilitate
- **Docker Compose** - Orchestrare multi-container
- **Nginx** - Load balancer și reverse proxy

2 Arhitectura sistemului

2.1 Diagrama de ansamblu

Sistemul este compus din mai multe componente care comunică între ele:

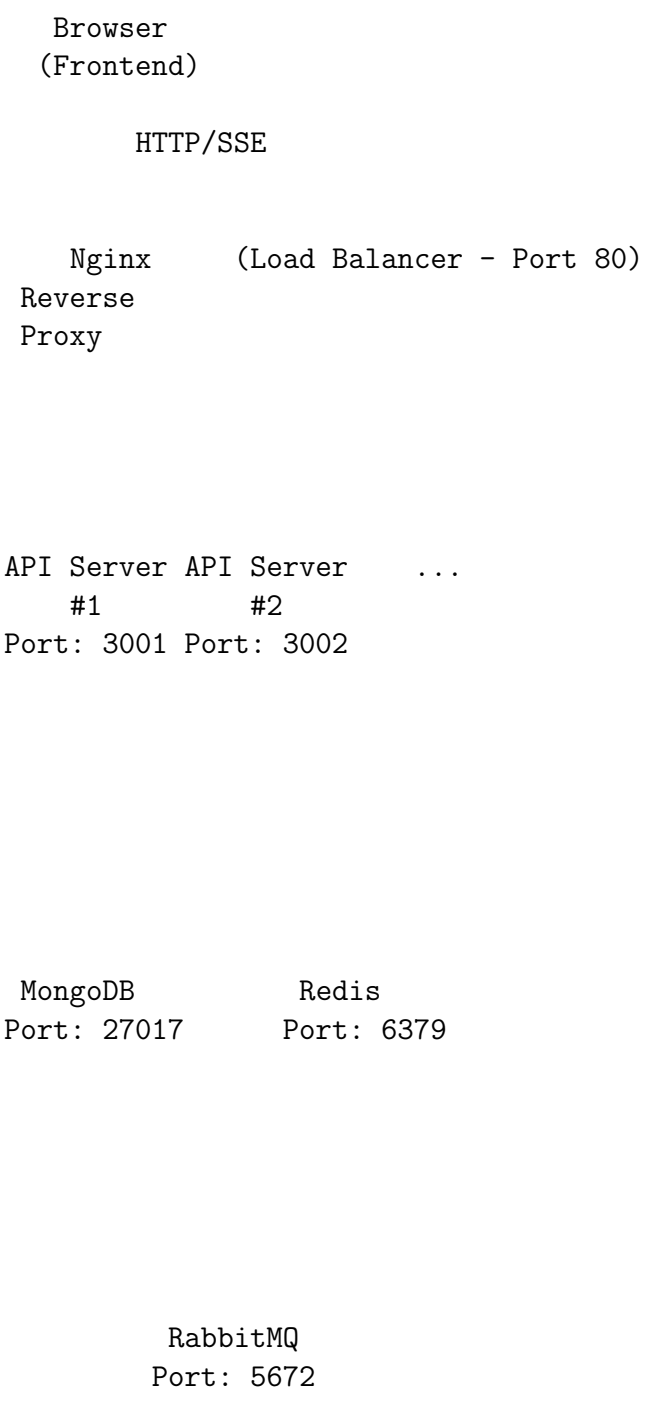

Figura 1: Arhitectura distribuită a sistemului

2.2 Componente principale

2.2.1 Load Balancer (Nginx)

Nginx funcționează ca reverse proxy și distribuie cererile HTTP între cele două instanțe de backend folosind algoritmul round-robin. Aceasta asigură:

- Distribuirea uniformă a load-ului
- High availability (dacă un server cade, celălalt preia)
- Single entry point pentru clienți

2.2.2 API Servers

Două instanțe Node.js/Koa care rulează în paralel și procesează cererile clienților. Fiecare server:

- Gestionează autentificarea utilizatorilor
- Menține conexiuni SSE cu clienții
- Procesează logica jocurilor (Poker, Hangman)
- Publică și consumă mesaje din RabbitMQ

2.2.3 MongoDB

Bază de date NoSQL care stochează:

- Conturi utilizatori (username, parolă hash)
- Istoricul mesajelor de chat
- Starea jocurilor (Poker, Hangman)

2.2.4 Redis

Folosit pentru:

- Pub/Sub între instanțele de server
- Cache pentru utilizatori online
- Sincronizare state jocuri între servere
- Gestionarea camerelor de chat

2.2.5 RabbitMQ

Message broker pentru:

- Distribuirea mesajelor de chat global
- Routing mesaje către camere specifice
- Mesaje private între utilizatori
- Decuplarea componentelor sistemului

3 Funcționalități

3.1 Autentificare și autorizare

3.1.1 Înregistrare utilizator

Utilizatorii se pot înregistra cu un username unic și o parolă. Parola este hash-uită folosind bcrypt înainte de stocare în MongoDB.

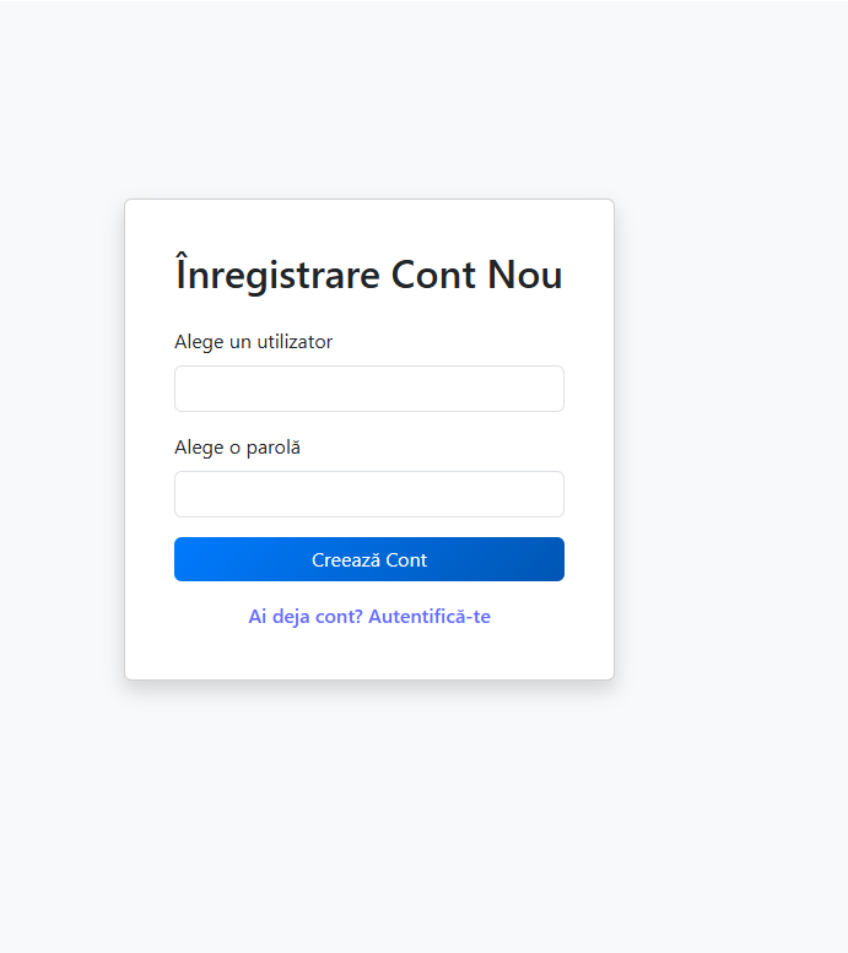

Figura 2: Ecran de înregistrare

3.1.2 Autentificare

La login, se generează un token JWT care este stocat într-un cookie HTTP-only pentru securitate. Token-ul conține username-ul și expiră după 24 de ore.

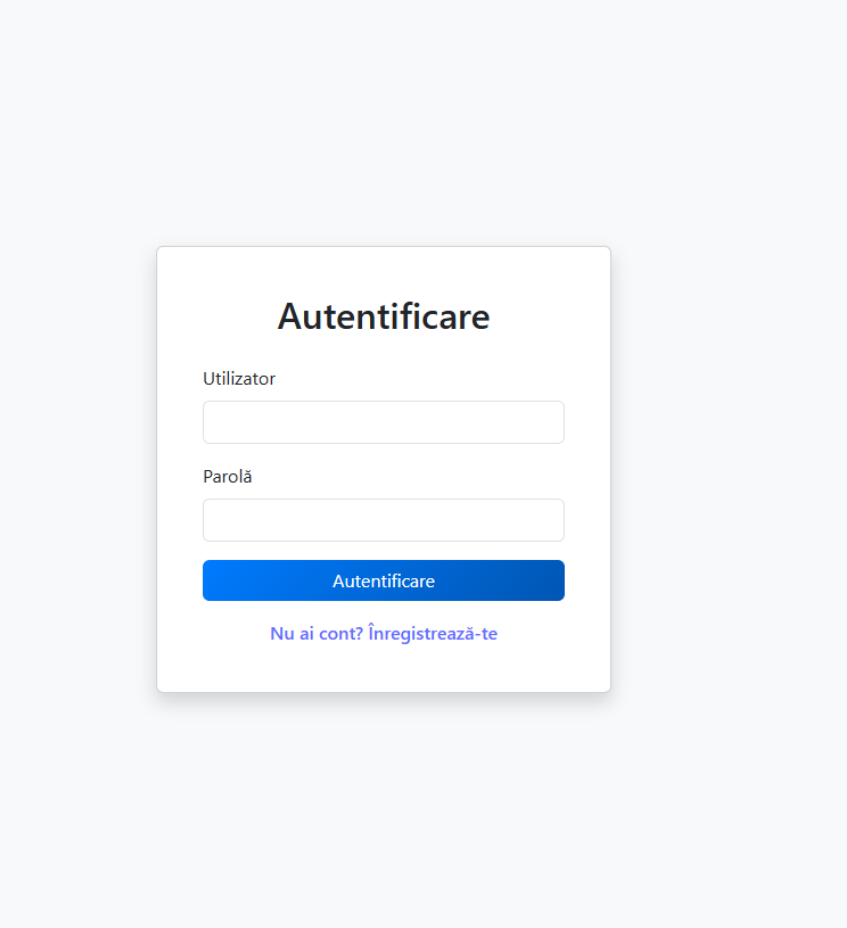

Figura 3: Ecran de autentificare

3.2 Chat în timp real

3.2.1 Chat global

Toți utilizatorii autentificați pot trimite și primi mesaje în chat-ul global. Mesajele sunt:

- Salvate în MongoDB pentru persistență
- Publicate în RabbitMQ exchange de tip fanout
- Distribuite către toți clienții conectați prin SSE

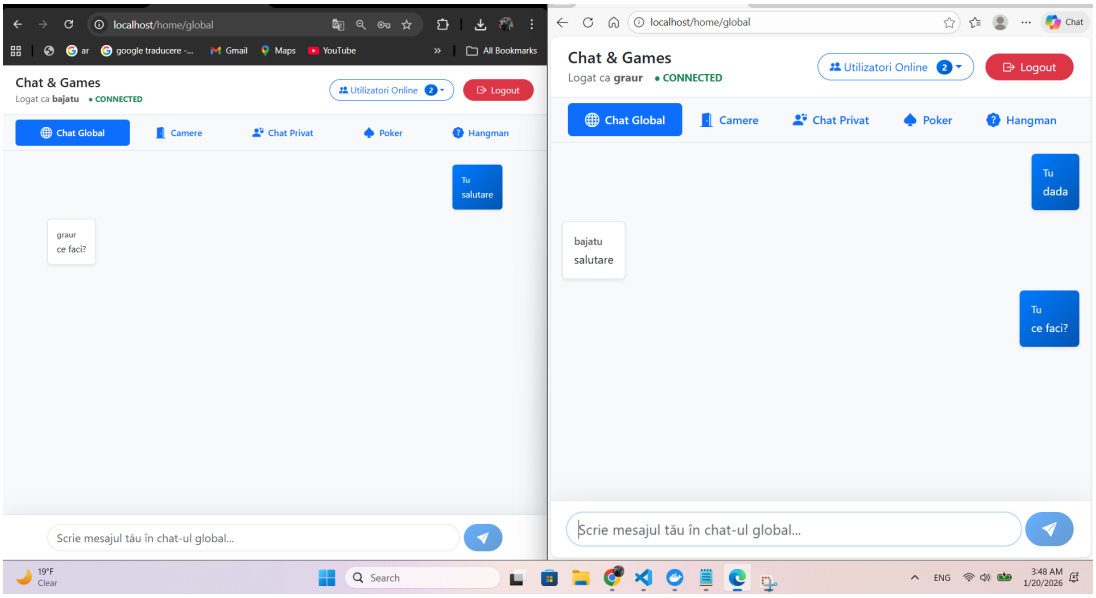

Figura 4: Interfață chat global

3.2.2 Camere de chat

Utilizatorii pot crea și se pot alătura camerelor de chat tematice:

- Creare cameră cu nume unic
- Join/Leave dinamic
- Mesaje vizibile doar membrilor camerei
- Ștergere automată când camera devine goală

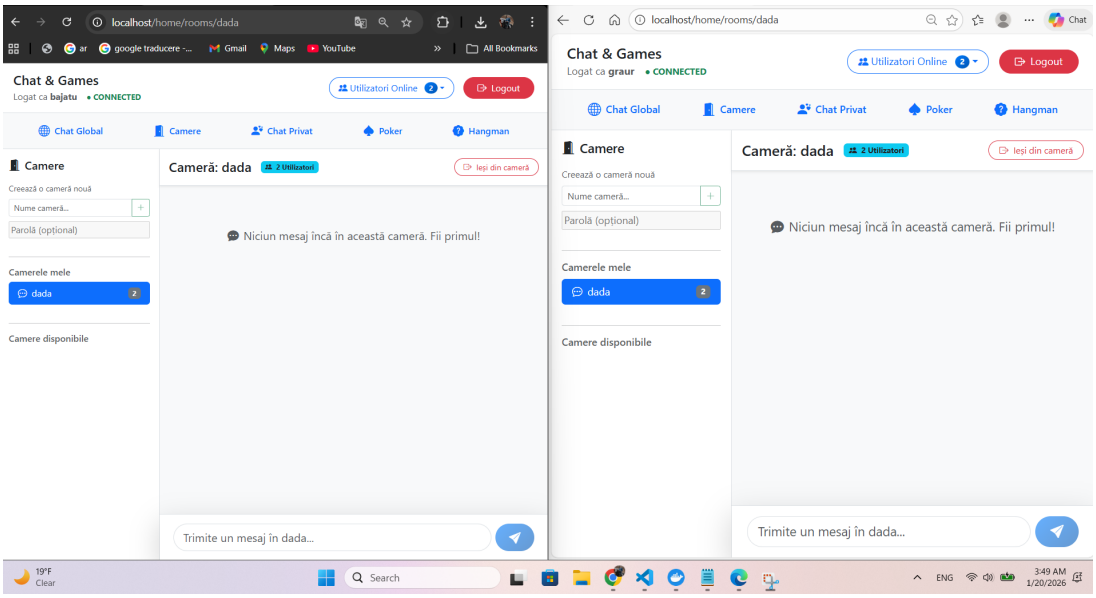

Figura 5: Sistem de camere de chat

3.2.3 Chat privat

Mesaje one-to-one între utilizatori:

- Selectare destinatar din lista de utilizatori online
- Routing prin RabbitMQ cu topic exchange
- Istoricul conversațiilor persistat în MongoDB

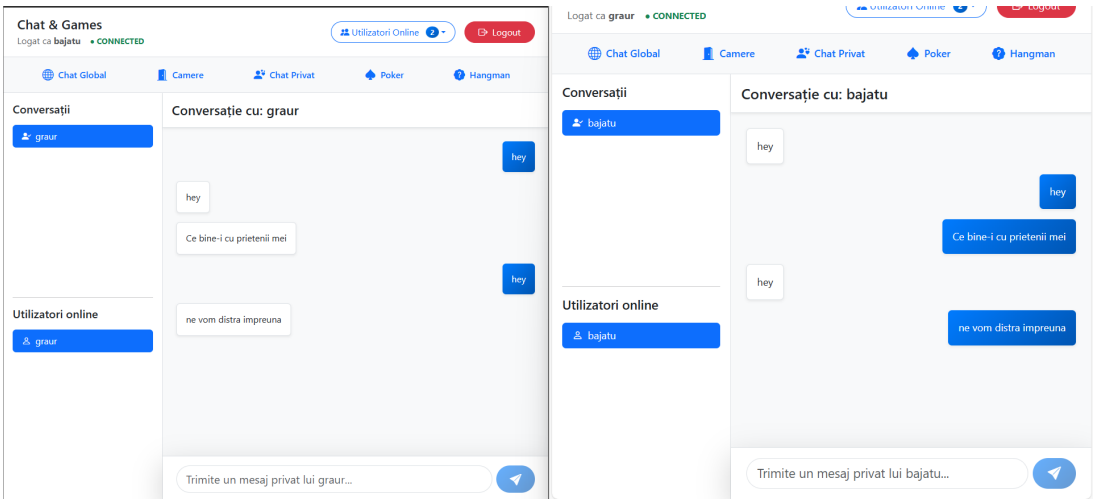

Figura 6: Chat privat între utilizatori

3.3 Jocuri multiplayer

3.3.1 Texas Hold'em Poker

Implementare completă a jocului de poker cu:

Caracteristici:

- Suport pentru 2-9 jucători
- Small blind și big blind configurabile
- Runde: Pre-flop, Flop, Turn, River, Showdown
- Acțiuni: Fold, Check, Call, Raise, All-in
- Evaluare automată a mâinilor folosind biblioteca holdem-poker
- Calculare pots, side-pots pentru all-in-uri multiple

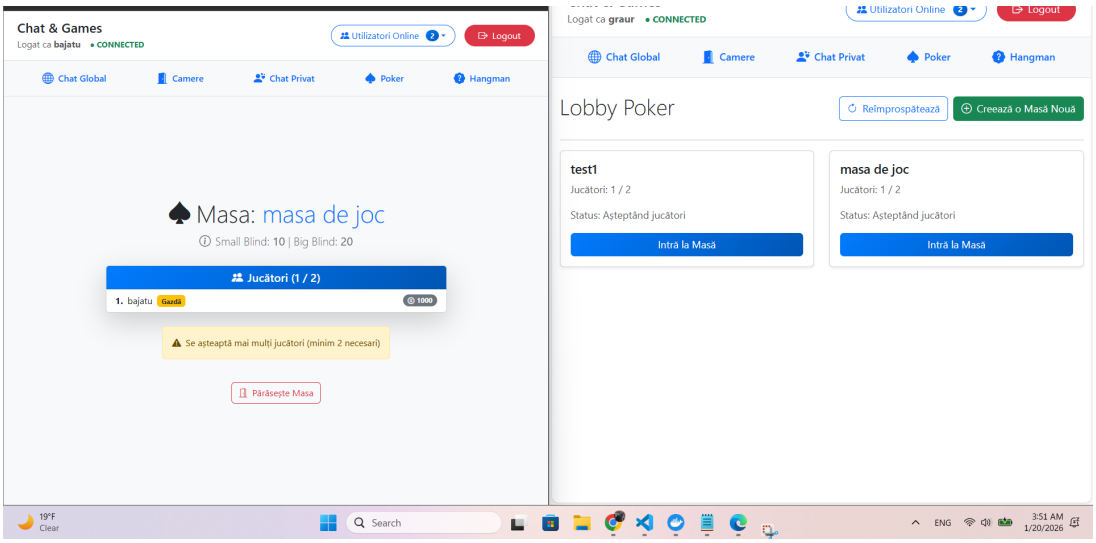

Figura 7: Lobby poker - lista jocurilor disponibile

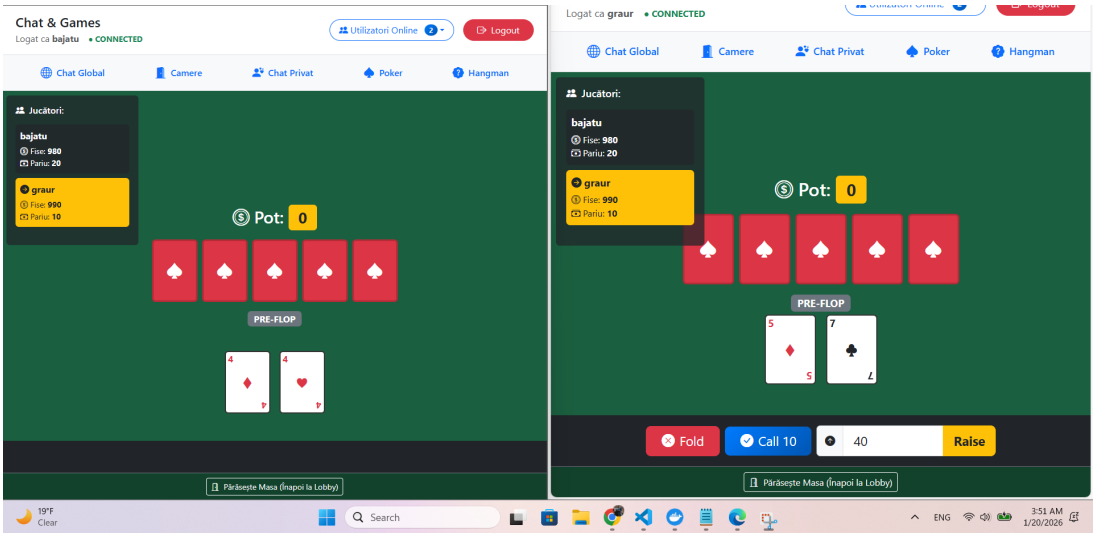

Figura 8: Masa de poker în timpul jocului

Flux de joc:

1. Creatorul configurează masa (blinds, număr maxim jucători, parolă opțională)
2. Jucătorii se alătură până la capacitate maximă
3. Creatorul începe jocul când sunt minim 2 jucători
4. Distribuire cărți, runde de pariuri
5. La showdown, se compară mâinile și se împarte pot-ul
6. Creatorul poate începe o mână nouă

3.3.2 Hangman (Spânzurătoarea)

Joc de ghicit cuvinte între doi jucători:

Caracteristici:

- Gazda setează cuvântul secret
- Ghicitorul încearcă să ghicească litera cu litera
- Număr limitat de încercări greșite (6)
- Feedback vizual pentru progres

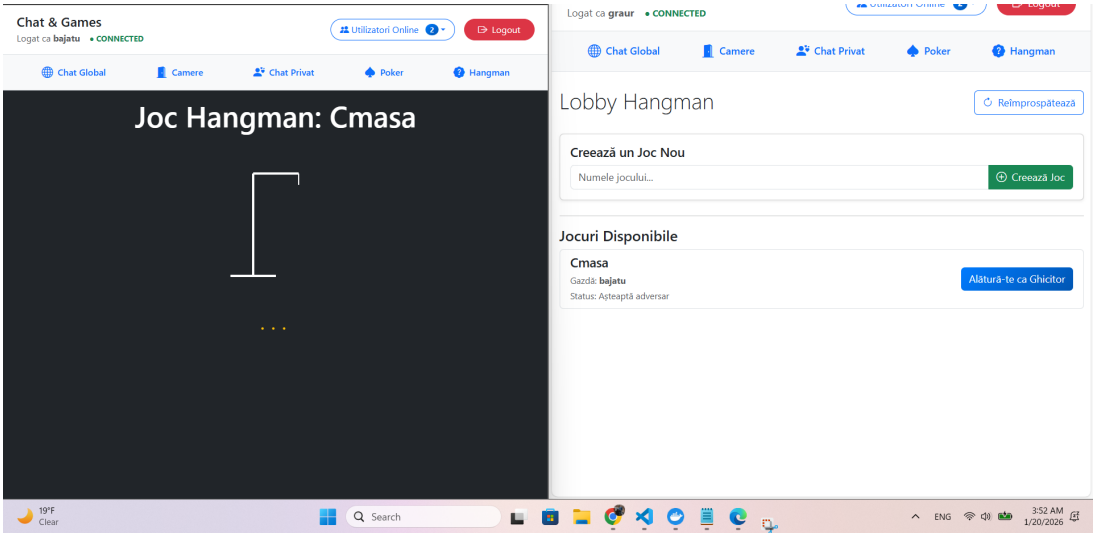

Figura 9: Lobby Hangman

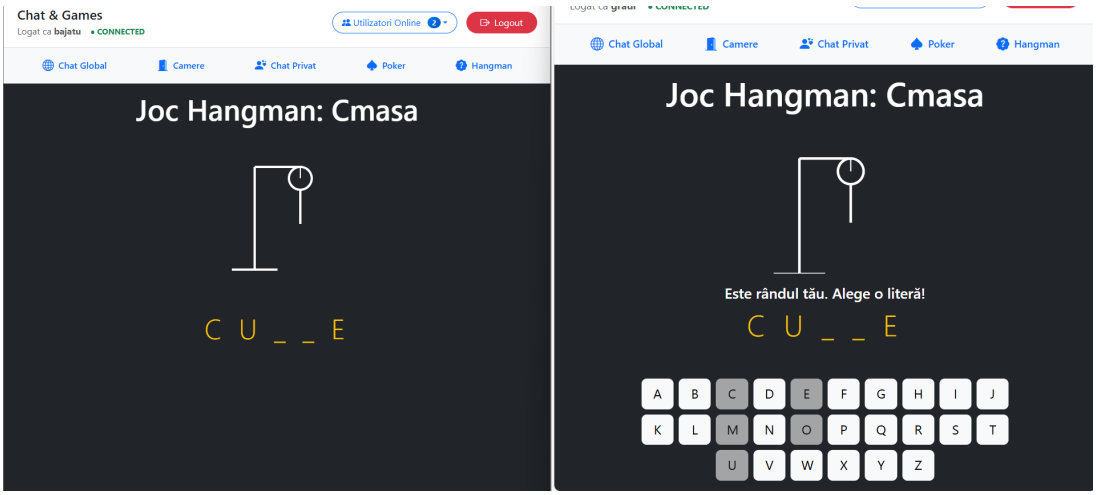

Figura 10: Joc Hangman în desfășurare

4 Implementare tehnică

4.1 Server-Sent Events (SSE)

SSE este utilizat pentru comunicare în timp real de la server către client. Implementarea asigură:

```
router.get('/api/events', authMiddleware, async (ctx) => {
    const username = ctx.state.username;

    ctx.set({
        'Content-Type': 'text/event-stream',
        'Cache-Control': 'no-cache',
        'Connection': 'keep-alive'
    });

    await sseManager.addClient(username, ctx, ctx.res);

    // Keep-alive ping
    const keepAliveInterval = setInterval(() => {
        ctx.res.write(': ping\n\n');
    }, 30000);

    // Cleanup handlers
    ctx.req.on('close', () => {
        clearInterval(keepAliveInterval);
        sseManager.removeClient(username);
    });

    // Mentine conexiunea deschisa
    await new Promise(() => {});
});
```

Listing 1: Conexiune SSE pe server

```
const connectSSE = () => {
    const es = new EventSource('/api/events');

    es.addEventListener('globalChatMessage', (event) => {
        const data = JSON.parse(event.data);
        setMessages(prev => [...prev, data]);
    });

    es.addEventListener('gameStateUpdate', (event) => {
        const gameState = JSON.parse(event.data);
        setCurrentPokerGame(gameState);
    });

    es.onerror = () => {
        es.close();
```

```
        setTimeout(connectSSE, 3000); // Retry
    };
};
```

Listing 2: Conexiune SSE pe client

4.2 Redis Pub/Sub

Redis este folosit pentru sincronizarea stării între instanțele de server:

```
// La fiecare update de joc
await redisPublisher.publish(
    `game-updates:${gameId}`,
    JSON.stringify(updatedGameState)
);
```

Listing 3: Publicare update joc în Redis

```
redisSubscriber.subscribe('game-updates:*');

redisSubscriber.on('message', (channel, message) => {
    const gameState = JSON.parse(message);
    const gameId = channel.split(':')[1];

    // Trimite update catre clienti prin SSE
    sseManager.sendEventToRoom(
        gameId,
        'gameStateUpdate',
        gameState
    );
});
```

Listing 4: Subscriber Redis pentru update-uri

4.3 RabbitMQ Message Routing

RabbitMQ gestionează distribuirea mesajelor de chat:

```
// Exchange pentru chat global (fanout)
await channel.assertExchange(
    'global_chat_exchange',
    'fanout',
    { durable: false }
);

// Exchange pentru chat camere (topic)
await channel.assertExchange(
    'room_chat_exchange',
    'topic',
    { durable: false }
```

```
);

// Queue pentru fiecare instanta de server
const { queue } = await channel.assertQueue('',
    { exclusive: true });

// Bind la exchanges
await channel.bindQueue(queue, 'global_chat_exchange', '');
await channel.bindQueue(queue, 'room_chat_exchange', '#');
```

Listing 5: Configurare RabbitMQ exchanges

```
// Chat global
channel.publish(
    'global_chat_exchange',
    '',
    Buffer.from(JSON.stringify({
        username: 'john',
        content: 'Hello everyone!',
        timestamp: new Date().toISOString()
    }))
);

// Chat camera
channel.publish(
    'room_chat_exchange',
    `room.gaming`,  // routing key
    Buffer.from(JSON.stringify({
        sender: 'john',
        text: 'Hi gamers!',
        room: 'gaming'
    }))
);
```

Listing 6: Publicare mesaj chat

4.4 Autentificare JWT

```
// La login - generare token
const token = jwt.sign(
    { username },
    process.env.SECRET_KEY,
    { expiresIn: '24h' }
);

ctx.cookies.set('token', token, {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'strict',
```

```
    maxAge: 24 * 60 * 60 * 1000 // 24 ore
});

// Middleware autentificare
const authMiddleware = async (ctx, next) => {
    const token = ctx.cookies.get('token');

    if (!token) {
        ctx.status = 401;
        return;
    }

    const decoded = jwt.verify(token, process.env.SECRET_KEY);
    ctx.state.username = decoded.username;
    await next();
};
```

Listing 7: Generare și validare JWT

4.5 Modele de date (MongoDB)

```
const userSchema = new mongoose.Schema({
    username: {
        type: String,
        required: true,
        unique: true
    },
    password: {
        type: String,
        required: true
    },
    createdAt: {
        type: Date,
        default: Date.now
    }
});

// Hash password inainte de salvare
userSchema.pre('save', async function(next) {
    if (!this.isModified('password')) return next();
    this.password = await bcrypt.hash(this.password, 10);
    next();
});
```

Listing 8: Schema User

```
const pokerGameSchema = new mongoose.Schema({
    gameId: { type: String, required: true, unique: true },
    creatorUsername: { type: String, required: true },
```

```
    password: { type: String, select: false },
    options: {
        smallBlind: { type: Number, default: 10 },
        bigBlind: { type: Number, default: 20 },
        maxPlayers: { type: Number, default: 9 },
        minPlayers: { type: Number, default: 2 }
    },
    players: [{
        username: String,
        stack: Number,
        currentBet: Number,
        hand: [String],
        status: String,
        hasActed: Boolean
    }],
    pot: { type: Number, default: 0 },
    board: [String],
    round: String,
    currentPlayerIndex: Number,
    inProgress: { type: Boolean, default: false }
});
```

Listing 9: Schema PokerGame

5 Deployment

5.1 Docker Compose

Aplicația este containerizată folosind Docker Compose pentru portabilitate și ușurință în deployment:

```
version: '3.8'

services:
  # MongoDB
  mongo:
    image: mongo:latest
    volumes:
      - mongo-data:/data/db
    ports:
      - "27017:27017"
    healthcheck:
      test: echo 'db.runCommand("ping").ok'
      interval: 10s

  # Redis
  redis:
    image: redis:latest
    ports:
      - "6379:6379"
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s

  # RabbitMQ
  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"
    healthcheck:
      test: rabbitmq-diagnostics -q ping
      interval: 10s

  # API Server 1
  api-server-1:
    build: .
    environment:
      - NODE_ENV=production
      - PORT=3001
      - MONGODB_URI=mongodb://mongo:27017/gaming
      - REDIS_HOST=redis
      - RABBITMQ_HOST=rabbitmq
    depends_on:
```

```
      - mongo
      - redis
      - rabbitmq

  # API Server 2
  api-server-2:
    build: .
    environment:
      - NODE_ENV=production
      - PORT=3002
      - MONGODB_URI=mongodb://mongo:27017/gaming
      - REDIS_HOST=redis
      - RABBITMQ_HOST=rabbitmq
    depends_on:
      - mongo
      - redis
      - rabbitmq

  # Nginx Load Balancer
  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    depends_on:
      - api-server-1
      - api-server-2

volumes:
  mongo-data:
```

Listing 10: docker-compose.yml

5.2 Configurare Nginx

```
upstream backend {
    server api-server-1:3001;
    server api-server-2:3002;
}

server {
    listen 80;

    location / {
        proxy_pass http://backend;
        proxy_http_version 1.1;

        # Headers pentru SSE
```

```
        proxy_set_header Connection '';
        proxy_set_header Cache-Control 'no-cache';

        # Headers standard
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For
           $proxy_add_x_forwarded_for;

        # Timeouts pentru SSE
        proxy_read_timeout 86400s;
        proxy_send_timeout 86400s;
    }
}
```

Listing 11: nginx.conf - Load balancing

5.3 Comenzi deployment

```
# Build si pornire toate serviciile
docker-compose up --build

# Pornire in background
docker-compose up -d --build

# Verificare status
docker-compose ps

# Vizualizare logs
docker-compose logs -f

# Oprire servicii
docker-compose down

# Oprire si stergere volume-uri
docker-compose down -v
```

Listing 12: Pornire aplicație

6 Testare și debugging

6.1 Testare funcționalități

6.1.1 Testare Chat

1. Deschide aplicația în două browsere diferite
2. Autentifică-te cu utilizatori diferiți
3. Trimite mesaje în chat global - ar trebui să apară instant în ambele browsere
4. Creează o cameră de chat și alătură-te din ambele browsere
5. Testează chat privat între cei doi utilizatori

6.1.2 Testare Poker

1. User 1 creează o masă de poker
2. User 2 se alătură mesei
3. User 1 (creator) începe jocul
4. Jucătorii fac acțiuni (fold, call, raise)
5. Verifică că starea jocului se sincronizează în timp real
6. Testează showdown și împărțirea pot-ului

6.2 Debugging

6.2.1 Logs în Docker

```
# Vezi logs pentru un serviciu specific
docker-compose logs -f api-server-1

# Vezi logs pentru toate serviciile
docker-compose logs -f

# Ultimele 100 linii
docker-compose logs --tail=100
```

6.2.2 Verificare conexiuni

```
# Verifica MongoDB
docker exec -it mongo mongosh
> use gaming
> db.users.find()

```

```
# Verifica Redis
docker exec -it redis redis-cli
> KEYS *
> SMEMBERS online_users

# Verifica RabbitMQ Management UI
# Browser: http://localhost:15672
# User: guest, Pass: guest
```

6.3 Probleme comune

6.3.1 SSE se deconectează continuu

Cauză: Promise-ul din ruta SSE nu așteaptă indefinit.
Soluție: Adaugă `await new Promise(() => {})` la sfârșitul rutei SSE.

6.3.2 Mesajele nu apar în chat

Cauză: RabbitMQ nu este conectat sau queue-urile nu sunt bind-uite corect.
Soluție: Verifică logs RabbitMQ și asigură-te că exchanges și queues sunt create.

6.3.3 Starea jocurilor nu se sincronizează

Cauză: Redis pub/sub nu funcționează între servere.
Soluție: Verifică că subscriber-ul Redis ascultă pe channel-ul corect și că publisher-ul publică mesaje.

7 Concluzii

7.1 Realizări

Proiectul a reușit să implementeze cu succes:

- Arhitectură distribuită scalabilă cu load balancing
- Comunicare în timp real folosind Server-Sent Events
- Sincronizare între multiple instanțe de server prin Redis și RabbitMQ
- Două jocuri multiplayer complet funcționale
- Sistem complet de chat (global, camere, privat)
- Autentificare securizată cu JWT
- Containerizare cu Docker pentru deployment facil

7.2 Provocări întâmpinate

- Gestionarea conexiunilor SSE și prevenirea deconectărilor
- Sincronizarea stării jocurilor între servere distribuite
- Routing corect al mesajelor prin RabbitMQ
- Debugging în mediu distribuit cu multiple containere

7.3 Îmbunătățiri viitoare

- Implementare sistem de matchmaking automat pentru jocuri
- Adăugare mai multe variante de poker (Omaha, Stud)
- Sistem de ranking și statistici jucători
- Notificări push pentru evenimente importante
- Suport pentru video/audio chat
- Implementare sistem de tournaments
- Optimizări de performanță și caching
- Teste automate (unit tests, integration tests)

7.4 Lecții învățate

- Importanța arhitecturii bine planificate în sisteme distribuite
- Provocările sincronizării stării în sisteme real-time
- Avantajele containerizării pentru development și deployment
- Necesitatea unui sistem robust de logging și monitoring
- Importanța testării în condiții de producție

8 Anexe

8.1 Anexa A - Structura proiectului

```
project/
          backend/
                src/
                      models/
                            user.model.js
                            chatMessage.model.js
                            pokerGame.model.js
                            hangmanGame.model.js
                      services/
                            poker.service.js
                            hangman.service.js
                      config.js
                      routes.js
                      sseManager.js
                      redisClient.js
                      rabbitClient.js
                      server.js
                Dockerfile
                docker-compose.yml
                nginx.conf
                package.json
          frontend/
                src/
                      components/
                            chat/
                                  GlobalChat.jsx
                                  RoomChat.jsx
                                  PrivateChat.jsx
                            poker/
                                  PokerLobby.jsx
                                  PokerTable.jsx
                            hangman/
                                  HangmanLobby.jsx
                                  HangmanGame.jsx
                            Header.jsx
                            Login.jsx
                            Register.jsx
                      App.jsx
                      main.jsx
                package.json
          README.md
```

8.2 Anexa B - Variabile de mediu

```
# Server Configuration
NODE_ENV=production
PORT=3001

# MongoDB
MONGODB_URI=mongodb://localhost:27017/gaming

# Redis
REDIS_HOST=localhost
REDIS_PORT=6379

# RabbitMQ
RABBITMQ_HOST=localhost
RABBITMQ_PORT=5672

# JWT Secret
SECRET_KEY=your-secret-key-here-change-in-production

# Cookie Settings
COOKIE_DOMAIN=localhost
COOKIE_SECURE=false
```

Listing 13: .env.example

8.3 Anexa C - API Endpoints

Method	Endpoint	Descriere
POST	/api/auth/register	Înregistrare utilizator nou
POST	/api/auth/login	Autentificare utilizator
POST	/api/auth/logout	Deconectare
GET	/api/auth/verify	Verificare token valid
GET	/api/events	Conexiune SSE pentru real-time
POST	/api/chat/global	Trimite mesaj chat global
GET	/api/chat/global/history	Istoric mesaje globale
POST	/api/chat/rooms/create	Creează cameră chat
POST	/api/chat/rooms/join	Alătură-te camerei
POST	/api/chat/rooms/leave	Părăsește camera
GET	/api/chat/rooms	Lista camere disponibile
POST	/api/chat/room/:name	Trimite mesaj în cameră
POST	/api/chat/private	Trimite mesaj privat
GET	/api/poker/games	Lista jocuri poker
POST	/api/poker/create	Creează joc poker
POST	/api/poker/join	Alătură-te jocului
POST	/api/poker/start	Începe jocul
POST	/api/poker/action	Acțiune joc (fold/call/raise)
POST	/api/poker/newhand	Mână nouă
POST	/api/poker/leave	Părăsește jocul
GET	/api/hangman/games	Lista jocuri hangman
POST	/api/hangman/create	Creează joc hangman
POST	/api/hangman/join	Alătură-te jocului
POST	/api/hangman/setword	Setează cuvânt secret
POST	/api/hangman/guess	Ghicește literă

Tabela 1: API Endpoints disponibile

9 Bibliografie

1. Node.js Documentation - https://nodejs.org/docs/
2. Koa.js Documentation - https://koajs.com/
3. React Documentation - https://react.dev/
4. MongoDB Manual - https://docs.mongodb.com/
5. Redis Documentation - https://redis.io/documentation
6. RabbitMQ Tutorials - https://www.rabbitmq.com/tutorials
7. Docker Documentation - https://docs.docker.com/
8. Nginx Documentation - https://nginx.org/en/docs/
9. Server-Sent Events Specification - https://html.spec.whatwg.org/multipage/server-sent-events.html
10. JWT Introduction - https://jwt.io/introduction