# Tic-tac-toe Arena

**A Modern, Real-Time Multiplayer TicTacToe Game with Competitive Matchmaking**

---

---

## Overview

**TicTacToe Arena** is a full-stack, real-time multiplayer online game platform that brings the classic TicTacToe game into the modern era. Built with a **microservices architecture**, the platform provides:

- **Real-time gameplay** with WebSocket communication
- **Competitive matchmaking** with ELO rating system
- **User authentication** and profile management
- **Global leaderboards** and player rankings
- **In-game chat** functionality
- **Responsive design** for all devices

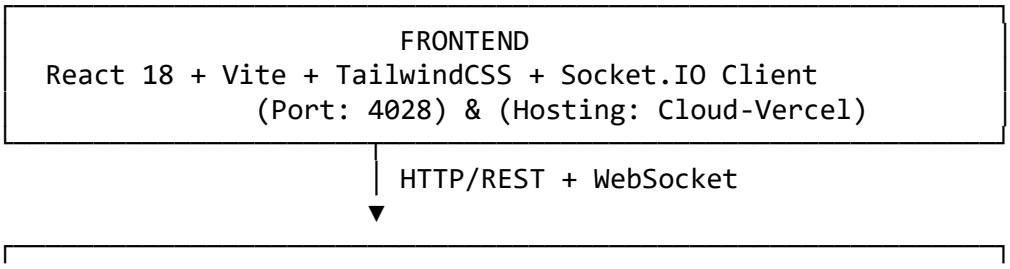
The project demonstrates enterprise-grade software development practices including: - Distributed microservices architecture - RESTful API design - Real-time bidirectional communication - Secure authentication with JWT - Database design with PostgreSQL/MySQL - Modern frontend with React and TailwindCSS

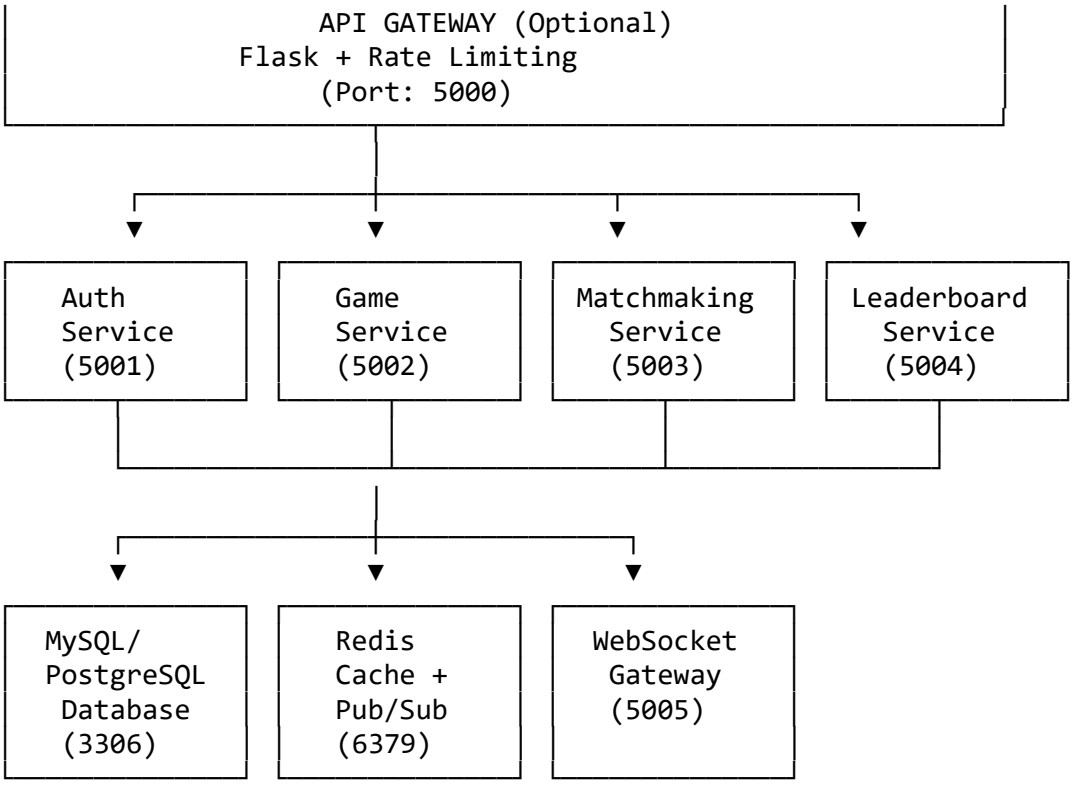
---

## Architecture

This project follows a **microservices architecture** pattern, where the application is decomposed into small, independent services that communicate with each other.

### Architecture Diagram






## Key Architectural Benefits

- **Independent Deployment** - Each service can be deployed separately
- **Scalability** - Scale services independently based on load
- **Fault Isolation** - Service failures don't cascade
- **Technology Flexibility** - Use best tools for each service
- **Team Independence** - Multiple teams can work in parallel
- **Maintainability** - Smaller codebases, easier to understand

---

# Features

## Core Gameplay

- Real-time multiplayer TicTacToe
- Move validation and game state management
- Win/Draw/Loss detection
- Game timer and move history
- Game abandonment handling

## User Management

- Email/Password authentication
- Social login options (Google, Facebook)
- User profiles with avatars
- Password reset functionality
- Session management with JWT

## Matchmaking

- Intelligent opponent matching
- ELO-based rating system (starts at 1200)
- Matchmaking preferences (rating range)
- Queue management
- Match history tracking

## Leaderboards

- Global rankings
- Friends leaderboard
- Player statistics (W/L/D, win streak)
- ELO progression tracking
- Player search

## Real-Time Features

- Live game updates via WebSocket
- Opponent move notifications
- In-game chat
- Connection status indicators
- Matchmaking status updates

## UI/UX

- Responsive design (mobile, tablet, desktop)
- Modern, intuitive interface
- Dark/Light mode support
- Smooth animations with Framer Motion
- Accessibility features

---

# Technology Stack

## Backend (Python)

| Category | Technology | Version | Purpose |
|---|---|---|---|
| **Framework** | Flask | 3.0.0 | Web application framework |
| **API** | Flask-RESTX | 1.3.0 | REST API + Swagger docs |
| **Database** | SQLAlchemy | 2.0.23 | ORM for database operations |
| | PostgreSQL | 14+ | Primary database (or MySQL) |
| | Flask-Migrate | 4.0.5 | Database migrations |
| **Authentication** | Flask-JWT-Extended | 4.6.0 | JWT token management |
| | Argon2-cffi | 23.1.0 | Secure password hashing |
| | Flask-Bcrypt | 1.0.1 | Alternative password hashing |
| **Real-Time** | Flask-SocketIO | 5.3.6 | WebSocket support |
| | Redis | 5.0.1 | Pub/Sub + caching |
| | Eventlet | 0.35.2 | Async server |
| **Validation** | Marshmallow | 3.20.1 | Request/response validation |
| **Security** | Flask-CORS | 4.0.0 | CORS handling |
| | Flask-Limiter | 3.5.0 | Rate limiting |
| | Flask-Talisman | 1.1.0 | Security headers |
| **Background Tasks** | Celery | 5.3.4 | Distributed task queue |
| | APScheduler | 3.10.4 | Scheduled tasks |
| **Testing** | Pytest | 7.4.3 | Test framework |
| | Pytest-Flask | 1.3.0 | Flask testing utilities |
| **Deployment** | Gunicorn | 21.2.0 | Production WSGI server |

## Frontend (JavaScript)

| Category | Technology | Version | Purpose |
|---|---|---|---|
| **Framework** | React | 18.2.0 | UI library |

| Category | Technology | Version | Purpose |
|---|---|---|---|
| **Build Tool** | Vite | Latest | Fast build tool |
| **Routing** | React Router | 6.0.2 | Client-side routing |
| **State Management** | Redux Toolkit | 2.6.1 | Global state management |
| **Styling** | TailwindCSS | 3.x | Utility-first CSS |
| **HTTP Client** | Axios | 1.8.4 | API requests |
| **WebSocket** | Socket.IO Client | Latest | Real-time communication |
| **Forms** | React Hook Form | 7.55.0 | Form management |
| **Animation** | Framer Motion | 10.16.4 | UI animations |
| **Icons** | Lucide React | 0.484.0 | Icon library |
| **Data Visualization (Future Use)** | Recharts | 2.15.2 | Charts and graphs |
| | D3.js | 7.9.0 | Advanced visualizations |
| **Date/Time** | date-fns | 4.1.0 | Date utilities |
| **Testing** | Jest | Latest | Test framework |
| | React Testing Library | Latest | Component testing |

## Infrastructure

| Technology | Purpose |
|---|---|
| **Docker** | Containerization |
| **Docker Compose** | Multi-container orchestration |
| **MySQL/PostgreSQL** | Relational database |
| **Redis** | Caching + message broker |
| **Nginx** | Reverse proxy (production) |

---

# Project Structure

```
TikTacToe-Arena/
│
├── be/                                 # Backend (Python/Flask)
│   ├── main.py                         # Main entry point & DB test
│   ├── requirements.txt                # Python dependencies
│   ├── README.md                       # Backend documentation
│   │
│   ├── services/                       # Microservices
│   │   ├── authServices/              # Authentication Service
│   │   │   ├── main.py                # Auth service entry
```

```
│   │   │   └── db/                 # Auth database layer
│   │   │       └── models/         # User models
│   │   │
│   │   └── gameServices/           # Game Service
│   │       ├── main.py             # Game service entry
│   │       └── db/                 # Game database layer
│   │
│   └── docs/                        # Documentation
│       ├── ARCHITECTURE.md          # Architecture guide
│       ├── MICROSERVICES_GUIDE.md  # Microservices details
│       ├── MIGRATION_GUIDE.md      # Supabase to Flask migration
│       ├── LIBRARIES_GUIDE.md      # Library explanations
│       └── SUMMARY.md              # Project summary
│
├── fe/                              # Frontend (React/Vite)
│   ├── index.html                  # HTML entry point
│   ├── package.json                # Node dependencies
│   ├── vite.config.mjs            # Vite configuration
│   ├── tailwind.config.js         # Tailwind configuration
│   ├── README.md                   # Frontend documentation
│   │
│   ├── public/                     # Static assets
│   │   ├── manifest.json
│   │   ├── robots.txt
│   │   └── assets/images/
│   │
│   ├── src/                        # Source code
│   │   ├── App.jsx                # Main app component
│   │   ├── index.jsx              # React entry point
│   │   ├── Routes.jsx             # Application routes
│   │   │
│   │   ├── components/            # Reusable components
│   │   │   ├── ui/                # UI components
│   │   │   │   ├── Button.jsx
│   │   │   │   ├── Input.jsx
│   │   │   │   ├── GameContextHeader.jsx
│   │   │   │   └── BottomTabNavigation.jsx
│   │   │   ├── ErrorBoundary.jsx
│   │   │   └── ScrollToTop.jsx
│   │   │
│   │   ├── pages/                 # Page components
│   │   │   ├── user-login/        # Login page
│   │   │   ├── user-registration/ # Registration page
│   │   │   ├── game-dashboard/    # Main dashboard
│   │   │   ├── matchmaking-game-lobby/  # Matchmaking
│   │   │   ├── active-game-board/ # Game board
│   │   │   ├── rankings-leaderboard/    # Leaderboards
│   │   │   └── NotFound.jsx
│   │   │
│   │   ├── contexts/              # React contexts
```

```
│   │   │   └── AuthContext.jsx   # Authentication context
│   │   │
│   │   ├── utils/                 # Utility functions
│   │   │   ├── authService.js    # Auth API calls
│   │   │   ├── gameService.js    # Game API calls
│   │   │   ├── leaderboardService.js
│   │   │
│   │   └── styles/                # Global styles
│   │       ├── index.css
│   │       └── tailwind.css
│   └── docker-compose.yml              # Docker services config
├── prompts.txt                         # Development prompts/notes
└── README.md                           # This file
```

---

# Prerequisites

Before you begin, ensure you have the following installed:

## Required Software

- **Python** 3.10 or higher
- **Node.js** 14.x or higher
- **npm** or **yarn**
- **MySQL** 8.0 or **PostgreSQL** 14+ (or Docker)
- **Redis** 6.0+ (or Docker)
- **Git**

## Optional (Recommended)

- **Docker** & **Docker Compose** (for easy setup)
- **VS Code** or your preferred IDE
- **Postman** or **Insomnia** (for API testing)

---

# Installation & Setup

## Option 1: Docker Setup (Recommended)

1. **Clone the repository**

   ```
   git clone https://github.com/yourusername/TikTacToe-Arena.git
   cd TikTacToe-Arena
   ```

2. **Start the database**

   ```
   docker-compose up -d
   ```

This will start MySQL on port 3306.

3. **Set up the backend**

```
cd be
python -m venv venv
.\venv\Scripts\Activate.ps1  # Windows PowerShell
pip install -r requirements.txt
```

4. **Configure environment variables**

Create `be/.env`:

```
# Flask Configuration
FLASK_APP=main.py
FLASK_ENV=development
SECRET_KEY=your-secret-key-here
JWT_SECRET_KEY=your-jwt-secret-here

# Database Configuration
DB_DRIVER=mysqlconnector
DB_USER=admin
DB_PASSWORD=admin
DB_HOST=localhost
DB_PORT=3306
DB_NAME=mysql_db

# Redis Configuration
REDIS_URL=redis://localhost:6379/0

# Frontend URL
FRONTEND_URL=http://localhost:4028
```

5. **Set up the frontend**

```
cd ../fe
npm install
```

6. **Configure frontend environment**

Create `fe/.env`:

```
VITE_API_URL=http://localhost:5000
VITE_WS_URL=ws://localhost:5005
```

## Option 2: Manual Setup (Without Docker)

1. **Install MySQL**
   - Download and install MySQL 8.0+
   - Create database: `CREATE DATABASE mysql_db;`
   - Create user and grant permissions

2. **Install Redis**
    - Download and install Redis
    - Start Redis server: `redis-server`
3. **Follow steps 1, 3-6 from Docker setup above**

---

# Running the Application

## Backend Services

The backend consists of multiple microservices. You can run them individually or use a process manager.

### *Individual Services*

**Terminal 1 - Auth Service:**

```
cd be/services/authServices
.\..\..\venv\Scripts\Activate.ps1
python main.py
# Runs on http://localhost:5001
```

**Terminal 2 - Game Service:**

```
cd be/services/gameServices
.\..\..\venv\Scripts\Activate.ps1
python main.py
# Runs on http://localhost:5002
```

**Terminal 3 - WebSocket Gateway:**

```
cd be
.\venv\Scripts\Activate.ps1
python websocket_gateway.py
# Runs on http://localhost:5005
```

### *Test Database Connection*

```
cd be
.\venv\Scripts\Activate.ps1
python main.py
```

## Frontend

**Terminal 4 - React Development Server:**

```
cd fe
npm start
# Runs on http://localhost:4028
```

## Access the Application

- **Frontend**: http://localhost:4028
- **Auth Service API**: http://localhost:5001
- **Game Service API**: http://localhost:5002
- **WebSocket Gateway**: http://localhost:5005
- **API Documentation**: http://localhost:5000/doc (if using API Gateway)

---

# Microservices

## 1. Authentication Service (Port 5001)

**Responsibilities:** - User registration and login - JWT token generation and validation - Password hashing (Argon2) - Password reset functionality - User profile management

**Key Endpoints:** - `POST /auth/register` - Create new user - `POST /auth/login` - User login - `POST /auth/refresh` - Refresh JWT token - `POST /auth/logout` - User logout - `GET /auth/profile` - Get user profile - `PUT /auth/profile` - Update profile - `POST /auth/password-reset` - Request password reset - `POST /auth/password-reset/confirm` - Confirm password reset

**Database Tables:** - `user_profiles` - User accounts and statistics

## 2. Game Service (Port 5002)

**Responsibilities:** - Game creation and management - Move validation - Game state tracking - Win/Draw/Loss detection - Game history

**Key Endpoints:** - `POST /games` - Create new game - `GET /games` - List user's games - `GET /games/:id` - Get game details - `POST /games/:id/move` - Make a move - `GET /games/:id/history` - Get move history - `DELETE /games/:id` - Abandon game

**Database Tables:** - `games` - Game records and states - `game_results` - Game outcomes and ELO changes

## 3. Matchmaking Service (Port 5003)

**Responsibilities:** - Queue management - ELO-based opponent matching - Match creation - Queue position tracking

**Key Endpoints:** - `POST /matchmaking/queue` - Join matchmaking queue - `DELETE /matchmaking/queue` - Leave queue - `GET /matchmaking/status` - Get queue status

**Database Tables:** - `matchmaking_queue` - Active matchmaking requests

## 4. Leaderboard Service (Port 5004)

**Responsibilities:** - Global rankings - Player statistics - ELO calculations - Player search

**Key Endpoints:** - `GET /leaderboard/global` - Global leaderboard - `GET /leaderboard/friends` - Friends leaderboard - `GET /leaderboard/search` - Search players - `GET /leaderboard/stats/:userId` - Player statistics

**Caching:** - Redis cache with 5-minute TTL for leaderboard data

## 5. WebSocket Gateway (Port 5005)

**Responsibilities:** - Real-time game updates - Move broadcasting - Chat messages - Connection management

**Events:** - `join_game` - Join game room - `leave_game` - Leave game room - `make_move` - Broadcast move to opponent - `chat_message` - Send chat message - `game_update` - Notify game state change

---

# Database Schema

## User Profiles

```
CREATE TABLE user_profiles (
    id UUID PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    full_name VARCHAR(100),
    avatar_url TEXT,
    password_hash VARCHAR(255) NOT NULL,
    role ENUM('admin', 'player', 'moderator') DEFAULT 'player',
    elo_rating INTEGER DEFAULT 1200,
    games_played INTEGER DEFAULT 0,
    games_won INTEGER DEFAULT 0,
    games_lost INTEGER DEFAULT 0,
    games_drawn INTEGER DEFAULT 0,
    win_streak INTEGER DEFAULT 0,
    best_win_streak INTEGER DEFAULT 0,
    last_active TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

## Games

```
CREATE TABLE games (
    id UUID PRIMARY KEY,
```

```
    player1_id UUID REFERENCES user_profiles(id),
    player2_id UUID REFERENCES user_profiles(id),
    current_player_id UUID REFERENCES user_profiles(id),
    winner_id UUID REFERENCES user_profiles(id),
    game_state JSON DEFAULT '[]',
    status ENUM('waiting', 'active', 'completed', 'abandoned'),
    moves_count INTEGER DEFAULT 0,
    started_at TIMESTAMP,
    ended_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### Game Results

```
CREATE TABLE game_results (
    id UUID PRIMARY KEY,
    game_id UUID REFERENCES games(id),
    player_id UUID REFERENCES user_profiles(id),
    result ENUM('win', 'loss', 'draw'),
    elo_before INTEGER NOT NULL,
    elo_after INTEGER NOT NULL,
    elo_change INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### Matchmaking Queue

```
CREATE TABLE matchmaking_queue (
    id UUID PRIMARY KEY,
    player_id UUID REFERENCES user_profiles(id),
    elo_rating INTEGER NOT NULL,
    preferences JSON DEFAULT '{}',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

# API Documentation

## Authentication

#### *Register User*

```
POST /auth/register
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "SecurePass123!",
  "username": "player1",
```

```
  "full_name": "John Doe"
}

Response: 201 Created
{
  "message": "User created successfully",
  "user_id": "uuid-here"
}
```

### *Login*

```
POST /auth/login
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "SecurePass123!"
}

Response: 200 OK
{
  "access_token": "eyJhbGciOiJIUzI1NiIs...",
  "refresh_token": "eyJhbGciOiJIUzI1NiIs...",
  "user": {
    "id": "uuid",
    "email": "user@example.com",
    "username": "player1",
    "elo_rating": 1200
  }
}
```

## Get Profile

```
GET /auth/profile
Authorization: Bearer <access_token>

Response: 200 OK
{
  "id": "uuid",
  "email": "user@example.com",
  "username": "player1",
  "full_name": "John Doe",
  "elo_rating": 1200,
  "games_played": 10,
  "games_won": 7,
  "games_lost": 2,
  "games_drawn": 1
}
```

## Game Operations

### *Create Game*

```
POST /games
Authorization: Bearer <access_token>
Content-Type: application/json

{
  "player2_id": "opponent-uuid" // optional
}

Response: 201 Created
{
  "game_id": "game-uuid",
  "status": "waiting",
  "game_state": [null, null, null, null, null, null, null, null, null]
}
```

### *Make Move*

```
POST /games/:gameId/move
Authorization: Bearer <access_token>
Content-Type: application/json

{
  "position": 4  // 0-8
}

Response: 200 OK
{
  "success": true,
  "game_state": ["X", null, null, null, "X", null, null, null, null],
  "current_player_id": "opponent-uuid",
  "status": "active",
  "winner_id": null
}
```

For complete API documentation, visit `/doc` endpoint when services are running.

---

# Security

## Authentication & Authorization

- **JWT Tokens**: Stateless authentication with access and refresh tokens
- **Password Hashing**: Argon2 algorithm (stronger than bcrypt)
- **Token Expiration**: Access tokens expire in 15 minutes, refresh tokens in 7 days
- **Password Requirements**: Minimum 8 characters, mix of letters and numbers

## Security Headers

- **CORS**: Configured for specific frontend origin
- **CSP**: Content Security Policy headers
- **HSTS**: HTTP Strict Transport Security
- **X-Frame-Options**: Prevent clickjacking
- **X-Content-Type-Options**: Prevent MIME sniffing

## Rate Limiting

- **Global**: 100 requests per minute per IP
- **Auth Endpoints**: 5 login attempts per minute
- **Game Moves**: 10 moves per minute per game

## Data Protection

- **SQL Injection**: Protected by SQLAlchemy ORM
- **XSS**: Input sanitization and output encoding
- **CSRF**: Token-based protection
- **Sensitive Data**: Passwords never logged or exposed

---

# Development

## Running Tests

**Backend Tests:**

```
cd be
.\venv\Scripts\Activate.ps1
pytest
pytest --cov=. --cov-report=html  # With coverage
```

**Frontend Tests:**

```
cd fe
npm test
npm run test:coverage
```

## Code Quality

**Backend Linting:**

```
flake8 .
black . --check
pylint **/*.py
mypy .
```

**Frontend Linting:**

```
npm run lint
npm run format
```

## Database Migrations

**Create Migration:**

```
flask db migrate -m "Add new column"
```

**Apply Migration:**

```
flask db upgrade
```

**Rollback Migration:**

```
flask db downgrade
```

## Adding a New Microservice

1. Create service directory in `be/services/`
2. Create `main.py` with Flask app
3. Define service-specific models
4. Implement REST endpoints
5. Add inter-service communication
6. Update documentation
7. Add tests
8. Update docker-compose if needed

---

# Deployment

## Production Considerations

1. **Environment Variables**
    - Use strong random secrets
    - Configure production database credentials
    - Set `FLASK_ENV=production`
    - Configure Redis with password
2. **Database**
    - Use managed database services (AWS RDS, Google Cloud SQL)
    - Enable SSL connections
    - Set up automated backups
    - Configure connection pooling
3. **Web Server**
    - Use Gunicorn for WSGI serving
    - Configure Nginx as reverse proxy

    - Enable HTTP/2 and compression
    - Set up SSL certificates (Let's Encrypt)
4. **Caching**
    - Use Redis cluster for high availability
    - Configure cache TTLs appropriately
    - Implement cache warming strategies
5. **Monitoring**
    - Set up application monitoring (Sentry, New Relic)
    - Configure log aggregation (ELK Stack, Datadog)
    - Set up uptime monitoring
    - Create dashboards for key metrics
6. **Scaling**
    - Horizontal scaling with load balancer
    - Database read replicas
    - CDN for static assets
    - Auto-scaling based on metrics

## Docker Deployment

```
# Build production images
docker-compose -f docker-compose.prod.yml build

# Start all services
docker-compose -f docker-compose.prod.yml up -d

# View logs
docker-compose logs -f

# Stop services
docker-compose down
```

## Cloud Deployment

The application will be deployed on: - **AWS**: EC2, ECS, RDS, ElastiCache - **Google Cloud**: GKE, Cloud SQL, Memorystore - **Azure**: App Service, Azure Database, Redis Cache - **Heroku**: Web dynos, Heroku Postgres, Heroku Redis

---

## Acknowledgments


- React and Vite teams for excellent tooling
- Flask community for comprehensive documentation
- TailwindCSS for beautiful utility classes
- Socket.IO for real-time communication
- All contributors and testers

## Project Status

**Current Version**: 1.0.1 (Development)

**Status**: Active Development

**Roadmap**: - ✓Database setup and connection - ✓Frontend UI components - ✓Project architecture design - ✓Authentication service implementation - ✓Game service implementation - ✓User Profile service - ✓Matchmaking service - ✓WebSocket integration - ✓Leaderboard service - □ Testing suite - □ Production deployment