Step-by-Step Guide: Matchmaking System, Tournament Brackets, and Backend Deployment with Docker

This guide will walk you through:

1. Building a Matchmaking System.

2. Creating Tournament Brackets.

3. Deploying the Backend with Docker.

---

1. Matchmaking System

Overview

The matchmaking system pairs players for a game. It uses a queue to manage waiting players and notifies them when they are matched.

Implementation

Add Matchmaking Routes

Update your app.rb:

require 'json'

# Queue to store waiting players

queue = []

# Add player to matchmaking queue

post '/matchmaking' do

data = JSON.parse(request.body.read)

player = { id: SecureRandom.uuid, username: data['username'] }

queue << player

if queue.size >= 2

# Match the first two players in the queue

player1, player2 = queue.shift(2)

{ status: 'Matched', players: [player1, player2] }.to\_json

else

{ status: 'Waiting', message: 'Waiting for another player...' }.to\_json

end

end

Test the Matchmaking Endpoint

1. Add a player to the queue:

curl -X POST -H "Content-Type: application/json" -d '{"username":"player1"}' http://localhost:4567/matchmaking

2. Add a second player to see them matched:

curl -X POST -H "Content-Type: application/json" -d '{"username":"player2"}' http://localhost:4567/matchmaking

---

2. Tournament Brackets

Overview

Tournament brackets organize players into a series of matches. Each round eliminates players until a winner is determined.

Implementation

Bracket Logic

1. Create a helper function to generate brackets:

def create\_brackets(players)

players.shuffle.each\_slice(2).to\_a

end

2. Add a route to handle tournament setup:

post '/tournament' do

data = JSON.parse(request.body.read)

players = data['players']

if players.size.even?

brackets = create\_brackets(players)

{ status: 'Tournament Created', brackets: brackets }.to\_json

else

{ status: 'Error', message: 'Number of players must be even.' }.to\_json

end

end

Test the Tournament Endpoint

1. Create a tournament with players:

curl -X POST -H "Content-Type: application/json" -d '{"players":["Alice","Bob","Charlie","David"]}' http://localhost:4567/tournament

---

3. Deploying the Backend with Docker

Create a Dockerfile

1. Add a Dockerfile to your project:

# Base image

FROM ruby:3.2-slim

# Install dependencies

RUN apt-get update && apt-get install -y build-essential libpq-dev

# Set the working directory

WORKDIR /app

# Copy Gemfile and install dependencies

COPY Gemfile Gemfile.lock ./

RUN bundle install

# Copy the application code

COPY . .

# Expose the port

EXPOSE 4567

# Start the server

CMD ["ruby", "app.rb"]

2. Build the Docker image:

docker build -t ft\_transcendance-backend .

3. Run the container:

docker run -p 4567:4567 --env-file .env ft\_transcendance-backend

---

Set Up Docker Compose

For multi-container deployment (e.g., with PostgreSQL), use Docker Compose.

1. Create a docker-compose.yml file:

version: '3.8'

services:

backend:

build: .

ports:

- "4567:4567"

env\_file:

- .env

depends\_on:

- db

db:

image: postgres:15

environment:

POSTGRES\_USER: ft\_user

POSTGRES\_PASSWORD: securepassword

POSTGRES\_DB: ft\_transcendance

volumes:

- db\_data:/var/lib/postgresql/data

volumes:

db\_data:

2. Build and start the services:

docker-compose up --build

3. Access your backend at http://localhost:4567.

---

Testing and Next Steps

Test Matchmaking and Tournament in Docker

1. Run the same curl commands from earlier to test the endpoints inside the container.

2. Ensure PostgreSQL data persists using the db\_data volume.

Next Steps

Add WebSocket Notifications to notify players when they’re matched.

Integrate HTTPS to secure communication.

Expand tournament logic to handle multiple rounds.

Would you like detailed guidance on adding WebSocket support to the tournament system or optimizing the Docker setup?