Detailed Guidance: Adding WebSocket Support and Optimizing Docker Setup

This section will cover:

1. Adding WebSocket Support to the Tournament System.

2. Optimizing the Docker Setup for Development and Production.

---

1. Adding WebSocket Support to the Tournament System

WebSockets enable real-time updates to notify players when they are matched or when a tournament progresses.

Update Your Dependencies

1. Add the faye-websocket gem:

gem 'faye-websocket'

Install the gem:

bundle install

WebSocket Integration

Basic WebSocket Setup

1. Add WebSocket support in app.rb:

require 'faye/websocket'

clients = [] # Store connected WebSocket clients

# WebSocket endpoint

get '/ws' do

if Faye::WebSocket.websocket?(env)

ws = Faye::WebSocket.new(env)

clients << ws

ws.on :open do |\_event|

puts "WebSocket connection opened"

end

ws.on :message do |event|

# Example: Broadcast messages to all connected clients

clients.each { |client| client.send(event.data) }

end

ws.on :close do |\_event|

puts "WebSocket connection closed"

clients.delete(ws)

end

ws.rack\_response

else

halt 400, 'WebSocket endpoint only'

end

end

Notify Players in Matchmaking

Update the /matchmaking route to send notifications:

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

player1, player2 = queue.shift(2)

# Notify players via WebSocket

clients.each do |client|

client.send("Matched: #{player1[:username]} vs #{player2[:username]}")

end

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

else

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

end

end

Notify Tournament Updates

Update the /tournament route:

post '/tournament' do

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

players = data['players']

if players.size.even?

brackets = create\_brackets(players)

# Notify clients about the tournament brackets

clients.each do |client|

client.send("Tournament Brackets: #{brackets}")

end

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

else

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

end

end

---

Testing WebSocket Integration

Test the WebSocket

1. Use a WebSocket client like websocat:

websocat ws://localhost:4567/ws

2. Open multiple websocat instances and send matchmaking or tournament requests to see real-time updates.

Test Matchmaking Notifications

1. Start the backend with WebSocket support.

2. Add players to matchmaking using:

curl -X POST -H "Content-Type: application/json" -d '{"username":"Player1"}' http://localhost:4567/matchmaking

curl -X POST -H "Content-Type: application/json" -d '{"username":"Player2"}' http://localhost:4567/matchmaking

3. Watch WebSocket clients receive updates about the matched players.

---

2. Optimizing Docker Setup

For a smoother workflow, configure Docker for both development and production environments.

Development vs. Production

Dockerfile

Modify the Dockerfile to handle both environments:

# Base image

FROM ruby:3.2-slim

# Install dependencies

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

# Set working directory

WORKDIR /app

# Copy Gemfile and install dependencies

COPY Gemfile Gemfile.lock ./

RUN bundle install

# Copy application code

COPY . .

# Expose port

EXPOSE 4567

# Default to development

ENV RACK\_ENV development

# Start the server

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

---

docker-compose.yml

1. Define separate configurations for development and production:

version: '3.8'

services:

backend:

build:

context: .

ports:

- "4567:4567"

volumes:

- .:/app # Mount local files for live updates in development

env\_file:

- .env

depends\_on:

- db

command: ["ruby", "app.rb"]

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. For production:

Remove the volume mount (- .:/app) to ensure only the built code is deployed.

---

Add HTTPS Support

1. Update docker-compose.yml to mount SSL certificates:

services:

backend:

build: .

ports:

- "4567:4567"

volumes:

- ./ssl:/ssl

env\_file:

- .env

command: ["ruby", "app.rb"]

2. Update app.rb to enable HTTPS:

require 'webrick/https'

set :server\_settings, {

SSLEnable: true,

SSLVerifyClient: OpenSSL::SSL::VERIFY\_NONE,

SSLPrivateKey: OpenSSL::PKey::RSA.new(File.open('ssl/key.pem')),

SSLCertificate: OpenSSL::X509::Certificate.new(File.open('ssl/cert.pem'))

}

---

Testing Dockerized Backend

1. Start the containers:

docker-compose up --build

2. Test all endpoints:

Matchmaking: /matchmaking.

Tournament: /tournament.

WebSocket: /ws.

---

Next Steps

Expand the WebSocket integration for in-game communication (e.g., ball position in Pong).

Add monitoring tools to your Docker setup (e.g., Prometheus/Grafana for logs and metrics).

Consider using reverse proxies like NGINX for routing HTTPS traffic to your container.

Would you like help with advanced monitoring tools or integrating WebSocket-driven in-game communication?