Advanced Ruby Techniques and Backend Enhancements

This guide will provide detailed examples for:

1. WebSocket Integration (for real-time updates like matchmaking and live chat).

2. SSL Setup for HTTPS Support (using a self-signed certificate).

3. Ruby Best Practices for Backend Development.

---

1. WebSocket Integration

WebSockets enable real-time communication, making them ideal for matchmaking, live chat, or game state synchronization.

Setup

1. Add the faye-websocket gem to your Gemfile:

gem 'faye-websocket'

Install the gem:

bundle install

Simple WebSocket Example

Add this to your app.rb:

require 'faye/websocket'

require 'json'

clients = [] # Store connected WebSocket clients

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|

# Broadcast message 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'

end

end

Testing WebSocket

Use a WebSocket client like websocat:

websocat ws://localhost:4567/ws

Open multiple websocat connections and send messages to see real-time updates.

---

Matchmaking System with WebSocket

Here’s how you can use WebSockets for matchmaking:

Code Example

require 'faye/websocket'

clients = {} # Track clients and their state

get '/matchmaking' do

if Faye::WebSocket.websocket?(env)

ws = Faye::WebSocket.new(env)

client\_id = SecureRandom.uuid

clients[client\_id] = ws

ws.on :open do |\_event|

puts "Client #{client\_id} connected"

if clients.size >= 2

# Match two players

players = clients.keys.sample(2)

players.each do |id|

clients[id].send("Matched! Opponent: #{(players - [id]).first}")

end

# Remove matched players from the pool

players.each { |id| clients.delete(id) }

end

end

ws.on :close do |\_event|

puts "Client #{client\_id} disconnected"

clients.delete(client\_id)

end

ws.rack\_response

else

halt 400, 'WebSocket endpoint'

end

end

---

2. SSL Setup for HTTPS Support

Ensure all communications are secure by enabling HTTPS with a self-signed SSL certificate.

Generate a Self-Signed Certificate

1. Run the following command to create a certificate:

openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes

key.pem: Private key file.

cert.pem: Certificate file.

2. Move the files to your project directory:

mv key.pem cert.pem ft\_transcendance-backend/

Enable HTTPS in Sinatra

Update app.rb:

require 'webrick/https'

# Create an HTTPS server

set :server\_settings, {

SSLEnable: true,

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

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

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

}

get '/' do

'Secure connection over HTTPS'

end

Run the Server

Run the app:

ruby app.rb

Visit your app at https://localhost:4567.

> Note: Your browser will warn about the self-signed certificate. You can proceed safely for development purposes.

---

3. Ruby Best Practices for Backend Development

Organizing Your Code

1. Folder Structure:

Separate your code into logical modules:

ft\_transcendance-backend/

├── app.rb # Main app file

├── routes/ # Route files

│ ├── users.rb # User-related endpoints

│ └── games.rb # Game-related endpoints

├── models/ # Models (e.g., database interactions)

│ ├── user.rb

│ └── game.rb

├── helpers/ # Helper functions

│ └── auth.rb

├── public/ # Frontend assets

│ └── index.html

├── config/ # Configuration files

│ ├── database.yml

│ └── environment.rb

└── Gemfile

2. Example Route File (routes/users.rb):

require 'sinatra'

get '/users' do

# Fetch all users

end

post '/register' do

# User registration logic

end

3. Load All Routes in app.rb:

Dir['./routes/\*.rb'].each { |file| require file }

---

Error Handling

Handle errors gracefully to improve user experience:

error 400 do

{ error: 'Bad Request' }.to\_json

end

error 404 do

{ error: 'Not Found' }.to\_json

end

error 500 do

{ error: 'Internal Server Error' }.to\_json

end

---

Environment Management

Use dotenv to manage environment variables:

1. Create a .env file:

DB\_NAME=ft\_transcendance

DB\_USER=ft\_user

DB\_PASSWORD=securepassword

2. Load variables in app.rb:

require 'dotenv/load'

db\_user = ENV['DB\_USER']

db\_password = ENV['DB\_PASSWORD']

---

Testing

Add basic tests using the minitest gem:

1. Add minitest to your Gemfile:

gem 'minitest'

2. Create a test file (test/test\_app.rb):

require 'minitest/autorun'

require\_relative '../app'

class AppTest < Minitest::Test

def test\_homepage

response = get '/'

assert\_equal 200, response.status

assert\_equal 'Welcome to ft\_transcendance', response.body

end

end

3. Run tests:

ruby test/test\_app.rb

---

Would you like further assistance with advanced Ruby techniques, such as creating APIs or additional examples for WebSocket use cases?