Backend - Ruby

Backend Roadmap for ft\_transcendance with Ruby

Phase 1: Preparation and Learning (Week 1)

1. Learn Ruby Basics (2-3 Days)

Understand syntax and basic concepts:

Variables, data types, control flow (if, else, case).

Loops (while, each, for).

Methods and blocks.

Explore resources:

Ruby Official Documentation.

FreeCodeCamp's Ruby tutorial or Codecademy’s Ruby course.

Build a simple Ruby script (e.g., a calculator).

2. Learn Ruby-Specific Features

Object-Oriented Programming:

Classes, modules, inheritance, and mixins.

Understand how Ruby's object model works.

Key libraries (Standard Library):

net/http for basic HTTP requests.

json for parsing and generating JSON.

---

Phase 2: Setup and Basic Backend Implementation (Week 2–3)

1. Project Setup

Install Ruby (latest stable version).

Initialize a new Ruby project:

mkdir ft\_transcendance-backend

cd ft\_transcendance-backend

bundle init

Add essential gems to Gemfile:

gem 'sinatra' # Lightweight Ruby web framework

gem 'json' # JSON parsing

gem 'bcrypt' # For password hashing

gem 'dotenv' # Environment variables

2. Build a Simple Web Server

Use Sinatra for a lightweight backend:

require 'sinatra'

require 'json'

get '/' do

{ message: 'Welcome to ft\_transcendance' }.to\_json

end

Run the server:

ruby app.rb

Test by visiting http://localhost:4567.

3. User Registration and Authentication

Create endpoints for user management:

POST /register: Register a user (store hashed passwords with bcrypt).

POST /login: Validate credentials and return a session token.

Use a simple data structure (like a hash or a text file) for now.

Example user registration:

post '/register' do

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

hashed\_password = BCrypt::Password.create(data['password'])

# Store `data['username']` and `hashed\_password` securely.

{ status: 'User registered' }.to\_json

end

4. Database Integration

Use PostgreSQL for user storage (if the Database module is chosen):

Install and set up PostgreSQL.

Add the pg gem to Gemfile for database interaction.

Learn basic SQL queries (INSERT, SELECT, UPDATE, DELETE).

Example database configuration:

require 'pg'

conn = PG.connect(dbname: 'ft\_transcendance', user: 'your\_user', password: 'your\_password')

conn.exec("CREATE TABLE users (id SERIAL PRIMARY KEY, username VARCHAR(50), password VARCHAR(255));")

---

Phase 3: Implement Matchmaking and Tournament Logic (Week 4)

1. Matchmaking System

Create an endpoint to queue players for matchmaking:

POST /matchmaking: Add a user to the queue.

Match players in pairs when at least two players are available.

2. Tournament Bracket

Create a system to manage tournament brackets:

Store player aliases and match order.

Announce next matches via an API endpoint (e.g., GET /tournament).

3. Real-Time Updates

Learn and implement WebSockets for real-time updates:

Use the faye-websocket gem for server-side WebSocket support.

Push notifications to connected clients about matchmaking and tournaments.

---

Phase 4: Security and Optimization (Week 5)

1. Secure User Data

Hash all passwords using bcrypt before storing them in the database.

Implement form validation for all inputs to prevent SQL injection and XSS.

2. Add HTTPS Support

Use the webrick server with an SSL certificate (self-signed for development).

3. Environment Variables

Store sensitive information like database credentials in a .env file:

require 'dotenv/load'

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

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

4. Optimize Code

Refactor your backend for modularity:

Separate concerns into files (e.g., user.rb, matchmaking.rb).

---

Phase 5: Bonus Features and Modules (Week 6+)

1. Remote Players Module

Extend matchmaking to handle remote players.

Use WebSockets for game synchronization between two players.

2. AI Opponent

Add an endpoint to generate AI moves based on game state:

POST /ai-move: Accept the current game state and return the AI’s move.

3. User and Game Stats Dashboards

Add endpoints for stats:

GET /stats/user/:id: Fetch user stats (e.g., wins, losses).

GET /stats/game/:id: Fetch game stats (e.g., scores, durations).

---

Ruby Strengths, Weaknesses, and Use Cases

1. Strengths:

Simple and elegant syntax, beginner-friendly.

Excellent for rapid prototyping and web applications.

Strong community support and many gems for common tasks.

2. Weaknesses:

Slower performance compared to languages like C++ or Go.

Limited for high-performance, CPU-intensive tasks.

3. Use Cases:

Web development (popular frameworks: Rails, Sinatra).

Scripting and automation.

Prototyping APIs and applications.

Would you like assistance with specific Ruby resources, detailed examples, or database setup instructions?