Project Specification: PokéForum SaaS

Course: CS-UY 4513 - Software Engineering
Professor: Dr. DePasquale
Project Title: PokellForum SaaS — A Community for Pokémon Enthusiasts
Date: September 28, 2025

1.0 Project Overview

PokéForum is a multi-user web platform on which Pokémon fans post forum threads, post team buildings, rate/review other user's teams or favorite Pokémon, and discuss meta-strategies. APIs and databases handling users and post data will be the backbone of the system. The system will be developed for user role-based moderation and reputation/voting (similar to Reddit karma). Users will have a large range of customization options and organizations (e.g., clubs, streamers, tournament orgs) can receive their own branded spaces which have exclusive customizable policies and themes.

2.0 Core Requirements

2.1 User-Based System

For auth: Email/password; JWT sessions (session cookies) with two-factor authentication

Optional Discord and Google logins (OAuth)

Personalization: user profiles with avatars and global-region; per-user preferences; saved/favorited Pokémon and teams; active posts.

2.2 User Roles (Minimum 3)

- **Guest**: Browse public forums/teams, view ratings/META. **CANNOT COMMENT OR RATE**

- **Member**: Customizable profile, create posts and teams, review/rate other user's teams, follow users/tags, can report content.

- **Moderator**: Pin/lock threads, remove content, act on reports made by members, manage tags within a user.

- **Admin/Owner**: Manage role assignments, feature toggles, API keys, analytics. *(We include Admin for SaaS completeness.)*

2.3 Persistent Storage

We will use MySQL (or whatever we are told to use by Prof) for relational data

Proposed Database Schema:

1. **users** — account, profile, auth identifiers.
    a. Separates identity from content so everything (posts, teams, reviews) can belong to users
    b. Makes it easy to add authentication (2FA secrets), etc.

2. **roles** — role catalog (guest/member/moderator/admin).
    a. A place to define/modify roles easily without touching user data, especially good if we plan to expand roles
    b. Easy to attach role data (perms, color, display)

3. **Threads** — forum threads, strategy notes, and comments (type: thread/review/strategy/announcement/comment).
    a. Dependant on forum tags to identify posting type
    b. Dependant on users to post

4. **Forum tags** – form catalog (thread/review/strategy/announcement/comment, etc)
    a. A list of the different types of threads that our database supportslll
    b. Maintains consistent typings among threads

5. **Teams** — user-submitted teams (title, format, description, favorites, public/private).
    a. Dependable on Pokémon to form valid team compositions

6. **Pokémon** — Pokémon entries (species, item, ability, EVs/IVs/moves).
    a. Sourced from Pokéapi

2.4 Modular Architecture (3–5 Modules)

1. **Identity**

    - Auth (JWT), user profiles, role/permission checks, user routing & theming.
    - **Dependency:** none; provides user context to all modules.

2. **Pokémon**
    - Name, typing, moves, etc

    - Delivers necessary pokemon data to teams
    - Sourced from pokéAPI

3. **Teams**

    - Team CRUD, team_members builder, import/export, reviews/ratings, aggregations
    - **Dependency: Identity** (ownership) for team to user mapping and **Pokemon** for team member data

4. **Forum & Content**

    - Posts, comments, reviews, rich text, tagging, search, pin/lock, pagination.
    - **Dependency:** requires **Identity** for authorship/permissions and **Teams** for review matching.

5. **Social & Notifications necessary?**

    - Real-time notifications for interactions with other users (comments, posts, etc)
    - **Dependency: Forum & Content** to know when user was interacted with each other

2.5 API Interfaces (REST)

Each module exposes RESTful endpoints (JSON). Versioning via `/api/`.

Identity API

- `POST /api/auth/register`
- `POST /api/v1/auth/login`
- `GET /api/v1/users/:id`
- `POST /api/v1/users` *(Admin)*

Forum & Content API

- `GET /api/v1/posts/:id`
- `POST /api/v1/posts/:id/comments`
- `POST /api/v1/posts/:id/lock` *(Moderator)*

Teams & Reviews API

- `POST /api/v1/teams`
- `GET /api/v1/teams/:id`

- `POST /api/v1/teams/:id/reviews` (score, text)
- `GET /api/v1/teams/:id/reviews`
- `POST /api/v1/teams/:id/export` (downloadable set text)

Meta & Strategy API

- `GET /api/v1/meta?format=&since=`
- `POST /api/v1/meta` *(Member+; flagged for Moderator review)*
- `GET /api/v1/meta/:id`
- `POST /api/v1/meta/:id/publish` *(Moderator/Admin)*

Social & Notifications API

- `POST /api/v1/favorites` (target_type: pokemon|team|post, target_id)
- `GET /api/v1/notifications`
- `POST /api/v1/follow` (user_id or tag)

3.0 Technical Stack

Language & Framework:

- Ruby 3.3, Ruby on Rails 7.x (API + server-rendered views)

Frontend:

- Rails Views + Turbo for forum, posts, teams, reviews, and live updates
- Tailwind CSS via `tailwindcss-rails`

Auth, Roles, Multi-tenancy:

- Devise for authentication (email/password)
- Simple role column on `users` (e.g., `guest`, `member`, `moderator`, `admin`) and basic before_actions for checks

Database & Storage:

- MySQL (primary relational store)
- Redis for caching, sessions, rate limiting

API:

- Rails controllers in API mode (versioned under `/api/v1`)

- Jbuilder or jsonapi-serializer for JSON response

Development Stack:

- Docker for dev/prod parity
- RSpec for development testing
- GitHub Actions for CI (RSpec, Rubocop, Brakeman), build & deploy
- Deploy to Heroku

4.0 Deliverables

1. **Complete Source Code** — monorepo (apps: web, api; packages: ui, types, config) with environment templates and seed scripts.

2. **Project Documentation** — PDF detailing features, admin tools, and contributors

3. **API Documentation** — Auto-generated Swagger (auto-api documentation software)

4. **Presentation** — 8–10 minute demo: creating a user, posting a META update, building a team, receiving reviews and live notifications.