Project Specification: TutorMe

Course: CS-UY 4513 - Software Engineering

Professor: Dr. DePasquale

Project Title: TutorMe

Date: September 28, 2025

1.0 Project Overview

The goal of this project is to design and develop a multi-module Software-as-a-Service (SaaS) system using Ruby on Rails. The system, named “TutorMe,” will centralize peer-to-peer academic support by matching students seeking help with approved student tutors, coordinating online session scheduling, and collecting post-session feedback. It is intended for learners, tutors, and program coordinators.

The system will require a robust user management system with

- distinct roles and permissions,
- persistent data storage, and
- a clear, modular architecture.

Each module must expose a well-defined API for inter-module communication.

Notes: The platform supports online meetings only. Sessions must include a meeting link. Learners select from tutor-posted availability slots. One session equals one posted availability slot.

2.0 Core Requirements

The project must adhere to the following mandatory characteristics:

2.1 User-Based System

The application must be a user-based system, requiring a secure user authentication and authorization framework. User accounts will be managed via standard email/password registration and login.

2.2 User Roles (Minimum 3)

The system will support three distinct user roles, each with specific permissions:

- **Learner (Tutee)**: Can sign up/sign in, browse tutors by subject/course, book into posted availability slots (1:1 or group), view upcoming sessions, cancel their own bookings, and submit ratings/comments after a session. Learners cannot edit tutor/student profiles, or view other students’ data.

● **Tutor:** May apply and be approved to tutor; can create/edit their tutor profile, publish availability slots with capacity, manage scheduled sessions, cancel sessions when necessary, and mark attendance after the meeting, and review feedback they received. They cannot edit tutor/student profiles. They cannot modify administrative settings.

● **Admin (Coordinator):** Oversees the program. Can approve/reject tutor applications, manage subject tags, view and modify any session, and review analytics (session counts, popular subjects, average ratings). Has full system access.

2.3 Persistent Storage

The application must use a relational database to persist all data. The database schema must be well-designed and contain **no more than 10 tables.**

Proposed Database Schema (max 10 tables):

● **learner:** Stores learners' credentials and core profile fields (learner_id, email, password, first_name, last_name).

● **tutor:** Stores tutor's credentials, details, approval state, and cached ratings (tutor_id, email, password, first_name, last_name, bio, qualifications, approval_status, photo_url, rating_avg, rating_count).

● **admin:** Stores admin credentials and core profile fields (admin_id, email, password, first_name, last_name)

● **subjects:** Canonical list of subjects/courses (id, code, name, description).

● **teaches:** Join tables linking tutors to subjects they offer (id, tutor_id, subject_id).

● **availability_slots:** Tutor-posted bookable windows; capacity defines 1:1 vs groups; supports recurrence (id, tutor_id, start_at, end_at, capacity, recurrence_rule).

● **sessions:** Booked online tutoring events created from a single posted slot; holds lifecycle and meeting link (id, tutor_id, subject_id, availability_slot_id, start_at, end_at, meeting_link, status, capacity)

● **sessions_attendees:** Roster of learners for each session; tracks join/attendance/feedback status. (id, session_id, learner_id, joined_at, attended, feedback_submitted, cancelled_at)

● **feedback:** Post-session ratings and comments (id, session_id, learner_id, tutor_id, score, comment).

2.4 Modular Architecture (3-5 Modules)

The system must be logically divided into a minimum of 3 and a maximum of 5 modules. These modules must demonstrate clear dependencies and rely on each other for full functionality.

Proposed Modules:

1. **User & Identity Management:**
    - ○ Manages user registration, login, and profile information
    - ○ Defines user roles (Learner, Tutor, Admin)

○ Maintains tutor applications status and approvals via linked tutor profiles.

○ **Dependency:** Core authentication/authorization service required by all other modules.

2. **Tutor & Subject Catalog:**
   ○ CRUD for tutor profiles and the subjects/coures offered
   ○ Allows learners to browse/filter tutors
   ○ Surfaces tutor availability snapshots to learners; bookings are initiated from these posted slots.
   ○ **Dependency:** Requires User & Identity (role check, tutor ownership). Feeds scheduling.

3. **Scheduling & Session Management:**
   ○ Instant-book from posted slots (one session = one slot); enforce non-overlap for tutors; support one-one-one and group sessions.
   ○ Updates calendars and session status; sessions are online-only and must include a meeting link.
   ○ Attendance: Tutor marks attendance after the meeting for each learner on the roster.
   ○ Cancellations: Learners can cancel their own reservations; Tutors can cancel sessions when needed; Admins may override.
   ○ **Dependency:** Use Tutor & Subject data (offerings, slots) and User & Identity (participant permissions)

4. **Feedback & Reporting:**
   ○ Feedback time rule: learners may submit ratings/comments only after the session has ended and the tutor has marked them as attended.
   ○ Gathers ratings and comments after sessions and produces summaries for coordinators.
   ○ Aggregates analytics for coordinators.
   ○ **Dependency:** Requires completed Sessions and attendance status; consults User & Identify for access control.

2.5 API Interfaces

Each module must expose a RESTful API interface. This is crucial for enabling the modules to communicate with each other and for potential future integrations. The API should follow standard REST conventions.

Example Endpoints:

- **User & Identity Management API**
  - POST /api/users/register (Create a new user account.)
  - POST /api/users/login (Log in and start a session/token.)
  - GET /api/me (Get the current user's profile and roles.)
  - POST /api/tutors/apply (Submit an application to become a tutor.)
  - POST /api/tutors/login (Submit an application to become a tutor.)
  - PATCH /api/tutors/:tutorId (Admin updates a tutor to be approved.)

- **Tutor & Subject Catalog**
    - GET /api/tutors?subject=Calculus&available_after=2025-10-01T10:00Z (Search tutors with filters and see their open times.)
    - GET /api/tutors/:id (View a tutor's profile, subjects, and rating summary.)
    - PATCH /api/tutors/:id/profile (Update tutor bio, qualifications, subject, and photo.)
    - POST /api/admin/subjects (Admin adds a new subject/course tag.)
    - POST /api/tutors/:id/availability (Post or update the tutor's available time slots in one request.)
    - GET /api/tutors/:id/availability?date<=>YYYY-MM-DD (List a tutor's bookable times for a specific day or range of dates.)
- **Scheduling & Session Management**
    - POST /api/sessions/:session_id/join (Learner joins an existing group session if seats remain; prevents duplicate joins.)
    - DELETE /api/sessions/:id/:learner_id (Learner leaves a session or is removed by Tutor/Admin.)
    - PUT /api/sessions/:id/attendance (Tutor marks which learners attended after the meeting.)
    - DELETE /api/sessions/:id (Cancel a session and record the reason.)
- **Feedback & Reporting**
    - POST /api/sessions/:id/feedback (Learner leaves a rating/comment only if marked attended and the session ended.)
    - GET /api/tutors/:id/feedback (View a tutor's reviews and average scores.)
    - GET /api/admin/analytics?range=start..end (Admin reports: sessions by subject, average ratings, tutor workload, etc.)

3.0 Technical Stack

- **Language**
    - **Ruby**
    - **JavaScript**
    - **HTML**
    - **CSS**
    - **YAML / Markdown**
- **Framework: Ruby on Rails** (Provides MVC, routing, validations, and Active Record to build a clean REST API and enforce booking rules like capacity and one session per slot.)
- **Database: PostgreSQL** (Offers reliable transactions and strong constraints to prevent double-booking, with fast indexing for date/subject searches and solid support for concurrent updates.)
- **Testing**
    - **RSpec** (Backend unit and API tests enforcing booking rules and lifecycle policies, for example rejecting double bookings, requiring cancellation reasons, and limiting attendance marking to tutors)
    - **Cucumber and Capybara** (End to end browser tests confirming full user journeys, for example a learner browses tutors and slots, books, sees the meeting link, the tutor marks attendance, and the learner submits feedback afterward.)

4.0 Deliverables

The final submission must include:

1. **Complete Source Code:** A well-structured and commented codebase, including all necessary configuration files.
2. **Project Documentation:** A README.md file that explains the system's architecture, setup instructions, and how to run the application.
3. **API Documentation:** A separate document or file detailing all API endpoints, their expected parameters, and response formats.
4. **Presentation:** A brief presentation and demonstration of the application's core features.