Project Specification: Clinic Appointment & Prescription System

Course: CS-UY 4513 - Software Engineering
Professor: Dr. DePasquale
Project Title: Clinic Appointment/Prescription System
Date: 9/25/25

1.0 Project Overview

Our project is to build a modular SaaS system that provides appointment and prescription services for a medical clinic. Users of the system will be divided into one of three roles: patients can locate doctors, schedule appointments, and view their prescriptions, doctors can sign up for clinic positions, manage their own appointments, and prescribe medication to their patients, and administrators have full access to the system's data for maintenance purposes.

To achieve this, the system will have access to persistent data storage in the form of a MySQL database, along with a variety of interconnected modules to handle user permissions and all of the actual functionality required to manage users, appointment and prescription records, and payments.

Each module will expose a well-defined RESTful API for easy communication and data transfer.

2.0 Core Requirements

The project must adhere to the mandatory characteristics outlined in the project specification document.

2.1 User-Based System

This is a user-based system, and it makes use of a secure authentication and authorization framework. Users accounts are managed by standard username/password registration and login.

2.2 User Roles (Minimum 3)

- Admin: Full system access, can manage user accounts and all stored data
- Doctor: Can sign up to work at a clinic, can specify availability for appointments, can view their list of upcoming appointments, and can prescribe things to any patient who they have had an appointment with. Cannot modify clinic data or other administrative settings
- Patient: Can view list of clinics/doctors and sign up for appointments with a doctor. Can view their own list of prescriptions. Cannot modify any other data

2.3 Persistent Storage

The application will use a relational database to store all of its data.
Proposed Database Schema (max 10 tables):

- patients: Stores credentials and profile info for patients (age, height, weight, gender, doctor notes)
- doctors: Stores credentials and profile info for doctors (salary, qualifications)
- admins: Stores credentials for admins
- clinics: Stores details about each clinic
- employments: Links doctors with clinics
- appointments: Stores records of appointments between a patient and a doctor
- time_slots: Stores appointment slots linked to specific doctor
- prescriptions: Stores prescriptions linked to a specific patient
- bills: Stores billing statements, dates, and amounts for each appointments record

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 management/authentication:
    a. Tracks user identities, handles login and permissions
    b. Dependency: used by all other modules
2. Clinic management
    a. Allows patients to view clinics, view doctors at a clinic, and view time slots for a doctor
    b. Allows doctors to start or stop working at a specific clinic
    c. Allows doctors to configure their available time slots
    d. Dependency: relies on user auth (to determine who the user is)
3. Appointment management
    a. Allows patients to sign up for an appointment with a doctor
    b. Allows doctors to view their upcoming appointments with patients
    c. Dependency: relies on user auth (to determine who the user is) and clinic info (to obtain employment lists and time slots)
4. Prescription management
    a. Allows doctors to prescribe things to patients after an appointment
    b. Allows patients to view their own prescriptions
    c. Dependency: relies on user auth (to determine who the user is) and appointments (to determine who is a valid patient to prescribe to)
5. Payment system
    a. Allows patients to make a payment for an appointment
    b. Dependency: relies on user auth and appointments to know which appointment to pay for

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_patient
  - POST /api/users/register_doctor
  - POST /api/users/register_admin
  - POST /api/users/login
  - GET /api/patient/{patient} (must be admin or **that** patient)
  - GET /api/doctor/{doctor} (must be admin or **that** doctor)
- Clinic Management API:
  - GET /api/clinics
  - GET /api/clinic/{clinic}/doctors
  - GET /api/doctor/{doctor}/time_slots
  - POST /api/clinic/{clinic}/employment (must be admin or doctor)
  - DELETE /api/clinic/{clinic}/employment (must be admin or doctor)
  - POST /api/doctor/{doctor}/time_slot (must be admin or **that** doctor)
  - DELETE /api/doctor/{doctor}/time_slot (must be admin or **that** doctor)
- Appointment Management API:
  - POST /api/appointment
  - DELETE /api/appointment
  - POST /api/appointment/{appt}/complete (must be admin or **assigned** doctor)
  - GET /api/patient/{patient}/appointments (must be admin or **that** patient)
  - GET /api/doctor/{doctor}/appointments (must be admin or **that** doctor)
- Prescription Management API
  - GET /api/patient/{patient}/prescriptions (must be admin, doctor or **that** patient)
  - POST /api/patient/{patient}/prescription (must be admin or doctor)
- Payment API
  - GET /api/patient/{patient}/bills (must be admin or **that** patient)
  - POST /api/bills/{bill}/pay (must be admin or **that** patient)

3.0 Technical Stack

- Language: Ruby
- Framework: Ruby on Rails
- Database: MySQL
- Testing: rSpec

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.