Hospital Management System (HMS)

Modern Application Development - I

Author: Venkatesh Ramakrishnan

Date: November 30, 2025

Institution: Indian Institute of Technology Madras

Abstract

This report documents the design and implementation of a Hospital Management System (HMS) developed as part of the Modern Application Development - I course. The system implements role-based access (admin, doctor, nurse, triage, patient) using Flask and SQLite, a calendar-driven appointment booking UI, and administrative and clinical workflows. This document explains the architecture (MVC mapping), data model (ER diagram), key features, testing performed, deployment instructions, and known limitations and future work.

Acknowledgements

The author thanks the course instructors and peers for guidance and the project rubric provided in the course materials.

Revision History

Date	Version	Notes
2025-11-30	1.0	Initial complete submission with ER diagram, code, and documentation

Introduction

This project implements a simplified Hospital Management System (HMS) focusing on out-patient clinic workflows: user management, role-based dashboards, appointment booking via a weekly calendar grid, doctor availability management, triage-assisted patient creation and booking, and basic treatment recording. The system's goals align with course deliverables: Flask back-end, Jinja2 templates, SQLite database, and a responsive UI using Bootstrap and FullCalendar.

Technologies & Tools

- Python 3.12, Flask (microframework), Flask-Login
- Jinja2 templates for server-side rendering
- SQLite for persistent storage (file: `hospital.db`)
- Bootstrap for responsive UI and FullCalendar for calendar UI

- PlantUML for ER diagram generation (source in repository)
- Tooling: virtualenv, pip, Graphviz (optional for PlantUML)

High-level Architecture and MVC mapping

The application follows an MVC-like pattern:

- Models: Python ORM-style classes under `models/` (e.g., `models/user.py`, `models/appointment.py`). They map directly to database tables.
- Views: Jinja2 templates stored under `templates/` (e.g., `templates/login.html`, `templates/patient/book_appointment.html`).
- Controllers: Flask blueprints in `routes/` (e.g., `routes/auth.py`, `routes/admin.py`, `routes/doctor.py`, `routes/patient.py`). These handle request routing, form processing, data validation and redirects.

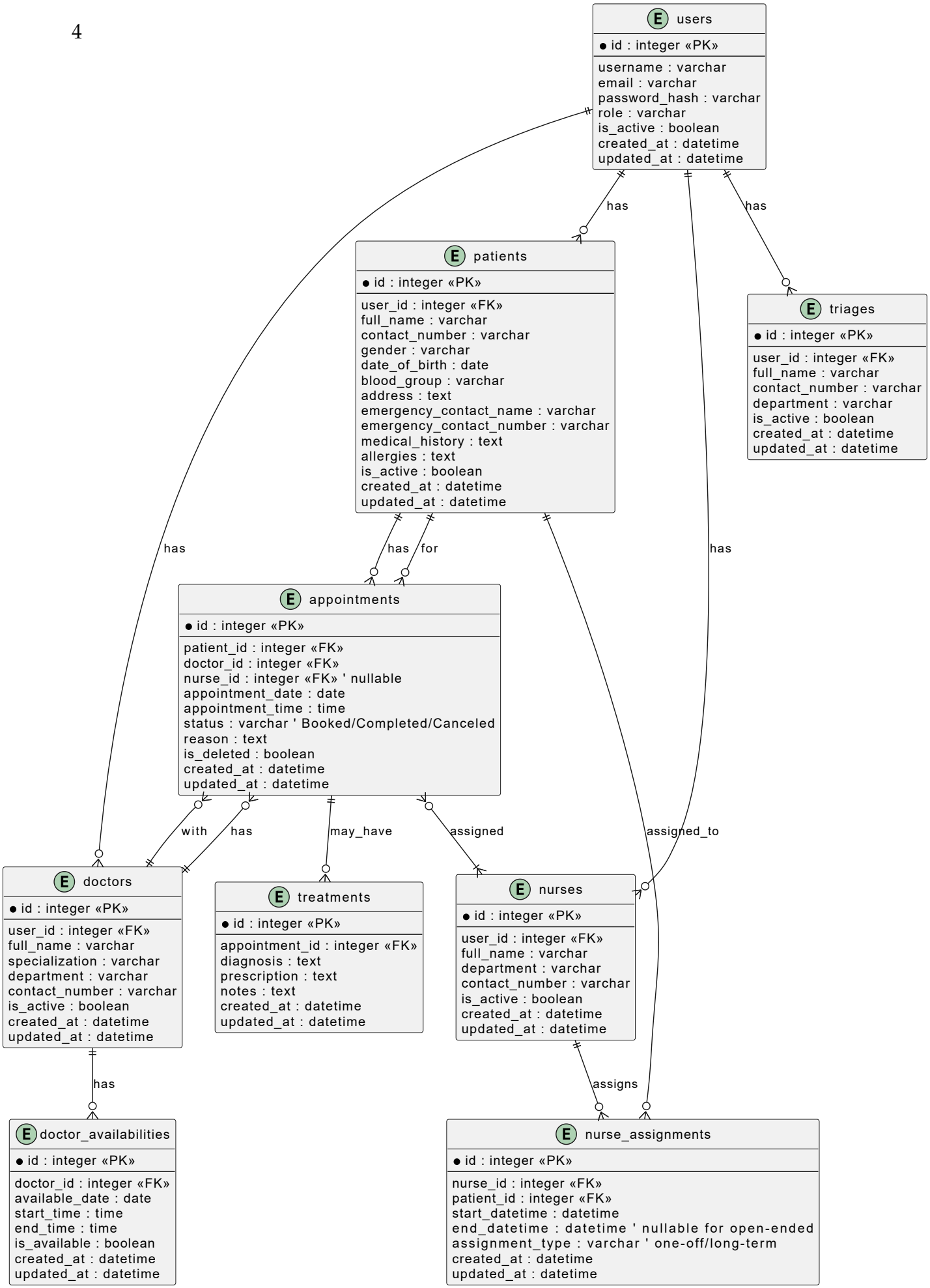
Extensions and helpers:

- `extensions.py` configures shared services (database, login manager).
- `utils/decorators.py` and `utils/helpers.py` provide role-check decorators and utility functions.

Database Design and ER Diagram

The ER diagram is included below. It shows the central `users` table and role-specific profile tables (`patients`, `doctors`, `nurses`, `triages`), and transactional tables such as `appointments`, `doctor_availabilities`, `treatments` and `nurse_assignments`.

The ER diagram is included on the following page.


users
id : integer «PK»
username : varchar
email : varchar
password_hash : varchar
role : varchar
is_active : boolean
created_at : datetime
updated_at : datetime
has
has
patients
id : integer «PK»
user_id : integer «FK»
full_name : varchar
contact_number : varchar
gender : varchar
date_of_birth : date
blood_group : varchar
address : text
emergency_contact_name : varchar
emergency_contact_number : varchar
medical_history : text
allergies : text
is_active : boolean
created_at : datetime
updated_at : datetime
triages
id : integer «PK»
user_id : integer «FK»
full_name : varchar
contact_number : varchar
department : varchar
is_active : boolean
created_at : datetime
updated_at : datetime
has
has
for
has
appointments
id : integer «PK»
patient_id : integer «FK»
doctor_id : integer «FK»
nurse_id : integer «FK» ' nullable
appointment_date : date
appointment_time : time
status : varchar ' Booked/Completed/Canceled
reason : text
is_deleted : boolean
created_at : datetime
updated_at : datetime
with
has
may_have
assigned
assigned_to
doctors
id : integer «PK»
user_id : integer «FK»
full_name : varchar
specialization : varchar
department : varchar
contact_number : varchar
is_active : boolean
created_at : datetime
updated_at : datetime
treatments
id : integer «PK»
appointment_id : integer «FK»
diagnosis : text
prescription : text
notes : text
created_at : datetime
updated_at : datetime
nurses
id : integer «PK»
user_id : integer «FK»
full_name : varchar
department : varchar
contact_number : varchar
is_active : boolean
created_at : datetime
updated_at : datetime
has
assigns
doctor_availabilities
id : integer «PK»
doctor_id : integer «FK»
available_date : date
start_time : time
end_time : time
is_available : boolean
created_at : datetime
updated_at : datetime
nurse_assignments
id : integer «PK»
nurse_id : integer «FK»
patient_id : integer «FK»
start_datetime : datetime
end_datetime : datetime ' nullable for open-ended
assignment_type : varchar ' one-off/long-term
created_at : datetime
updated_at : datetime

Data Models — mapping to code

Below are short descriptions of the principal model classes and their responsibilities.

User model (models/user.py)

Purpose: central authentication and role storage. Key attributes: `id`, `username`, `email`, `password_hash`, `role`, `is_active`. Role helper methods include `is_admin()`, `is_doctor()`, `is_patient()`, `is_nurse()`, `is_triage()`.

Example (short snippet — explanatory only):

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String, unique=True, index=True)
    email = db.Column(db.String, unique=True, index=True)
    password_hash = db.Column(db.String)
    role = db.Column(db.String)  # admin/doctor/patient/nurse/triage
    is_active = db.Column(db.Boolean, default=True)

    def is_triage(self):
        return getattr(self, 'role', None) == 'triage'
```

Appointment model (models/appointment.py)

Purpose: stores appointment records with references to patient, doctor, optional nurse, date/time and status.

Key columns: `patient_id`, `doctor_id`, `nurse_id` (nullable), `appointment_date`, `appointment_time`, `status`.

Functional Requirements & Feature List

- Role-based login and dashboards: admin, doctor, nurse, triage, patient.
- Admin features: add/edit/delete doctors, nurses, patients; view and search appointments; assign staff.
- Doctor features: set availability, view weekly schedule, view patient history, complete appointments and record treatments.
- Patient features: register, login, book appointments via calendar grid, reschedule, view history and profile.
- Triage features: create patient records (one-off), and book appointments directly for patients; triage users have specific landing pages and workflows.

- Nurse management: assign nurses to appointments or patients (time-bound nurse assignments supported).
- Calendar integration: weekly grid with FullCalendar, displays availability, allows slot selection; non-availability drawn with diagonal shading.
- Conflict checking: backend checks when booking/rescheduling to prevent double-booking and notifies via flash messages.

Detailed Implementation Notes

Login flow and redirects

Login happens at `routes/auth.py::login`. After successful authentication, the application redirects based on role:

- admin → `admin.dashboard`
- doctor → `doctor.dashboard`
- triage → triage landing page (ensure `is_triage` exists on User)
- patient → `patient.dashboard`

A common cause of redirect loops is protecting the login route with a login-required decorator — ensure the login route has no access-control decorator.

Appointment booking

Booking relies on doctor availability records (`doctor_availabilities`) and the calendar UI. The front-end requests slots and the backend returns available times (taking into account existing appointments). On selection, the POST handler validates conflicts before creating an `Appointment` record.

Reschedule flow

Rescheduling uses the calendar UI (or fallback select lists). The backend checks for conflicts and ensures the new slot matches doctor availability.

Nurse assignment

Appointments include nullable `nurse_id`. A separate `nurse_assignments` table supports long-term assignments with `start_datetime` and `end_datetime`. Admin, triage, and doctors can assign nurses.

Key templates and controllers referenced

- `routes/auth.py` — login, register, logout
- `routes/main.py` — root route and landing redirects
- `routes/admin.py` — admin CRUD and dashboard
- `routes/doctor.py` — availability and doctor dashboard
- `routes/patient.py` — patient features and booking
- `templates/login.html`, `templates/register.html`, `templates/patient/book_appointment.html`, `templates/patient/reschedule_appointment.html`

Testing

Manual test cases performed:

- Admin login: verified
- Patient registration + login: verified
- Triage: created and tested triage user using `create_triage.py`
- Booking: calendar displays doctor availability; able to book and view appointments
- Conflict handling: attempted to double-book a slot to verify flash message appears
- Reschedule: verified UI flow and server-side conflict rejection where applicable

Deployment & Run Instructions

```
# Activate virtual environment
# Unix
source venv/bin/activate

# Windows (PowerShell)
venv\Scripts\Activate.ps1

# Set environment variables and run
export FLASK_APP=app.py
```

```
export FLASK_ENV=development
flask run

# Create triage user if needed
python create_triage.py
```

Security & Privacy Considerations

- Passwords are stored hashed (do not store plain text passwords).
- Sessions rely on Flask session cookies — set secure cookie flags in production.
- Role checks implemented in decorators — ensure those decorators are applied correctly on sensitive routes.
- Sanitize any user-supplied text that is displayed back in templates to avoid injection.

Usability & UI Notes

FullCalendar provides a familiar weekly view for booking. Non-availability shading and diagonal patterns were added in CSS/JS. The calendar allows click-to-select an available slot, and the backend verifies availability at the time of booking.

Known Issues & Future Work

- Improve automated tests and add unit tests for booking logic and conflict checks.
- Add migration support (Flask-Migrate) for schema changes.
- Email notifications for appointment confirmations and reminders.
- Audit logging for admin actions.
- Background worker (RQ/Celery) for heavy tasks and email sending.

User Guide (step-by-step)

Admin

- Login as admin (default demo: admin / admin123).
- Use Admin Dashboard to add doctors/nurses/patients.
- Set doctor availability via Doctor Availability pages.
- View appointments and assign nurses if required.

Doctor

- Login with doctor credentials.
- Set weekly availability and view scheduled appointments.
- Mark appointment as complete and add treatment details.

Triage

- Triage users can create a patient record and book an appointment for the patient.
- Triage landing page allows immediate booking into available slots.

Patient

- Register or login, view dashboard, book/reschedule appointments via calendar.