Hospital Management System

MAD-1 Project Report

Student Name: [To be filled]

Roll Number: 2KXXXX

Date: [To be filled]

Course: Modern Application Development - I

Indian Institute of Technology Madras

1. Project Overview

Introduction

The Hospital Management System (HMS) is a comprehensive web-based application designed to streamline hospital operations by digitizing and automating key processes such as patient registration, doctor management, appointment scheduling, and treatment record maintenance. This system addresses the inefficiencies of manual record-keeping and disconnected software systems prevalent in many healthcare facilities.

Objectives

- **Centralized Data Management:** Create a unified platform for managing patient records, doctor profiles, and appointments
- **Role-Based Access Control:** Implement secure, role-specific interfaces for Admins, Doctors, and Patients
- **Efficient Scheduling:** Prevent appointment conflicts through automated slot validation and availability management
- **Complete Medical Records:** Maintain comprehensive treatment history with diagnosis, prescriptions, and follow-up information
- **Enhanced User Experience:** Provide intuitive interfaces with real-time data visualization and responsive design

Problem Statement

Hospitals currently face challenges with manual registers prone to errors, scheduling conflicts, difficulty tracking patient medical history, limited search capabilities, lack of centralized access, and inefficient communication. This HMS provides a modern, efficient solution through a unified web application.

2. Technology Stack & Frameworks

Backend Technologies

Flask 3.0.0 | SQLAlchemy 2.0.23 | Flask-Login 0.6.3 | Flask-Mail 0.9.1 | Werkzeug 3.0.1

Frontend Technologies

HTML5 | CSS3 | Bootstrap 5.3 | JavaScript | Chart.js | Jinja2

Database

SQLite - Relational database programmatically created using SQLAlchemy models with ACID transaction support.

3. System Architecture & Approach

MVC Architecture

The application follows the **Model-View-Controller (MVC)** design pattern:

Models (models/)

Define database schema, relationships, data validation, and business logic. Implement soft delete pattern.

Views (templates/)

Jinja2 templates for dynamic HTML rendering, organized by role with reusable base template.

Controllers (routes/)

Flask blueprints handling HTTP requests, form submissions, business logic, and authorization.

Project Folder Structure

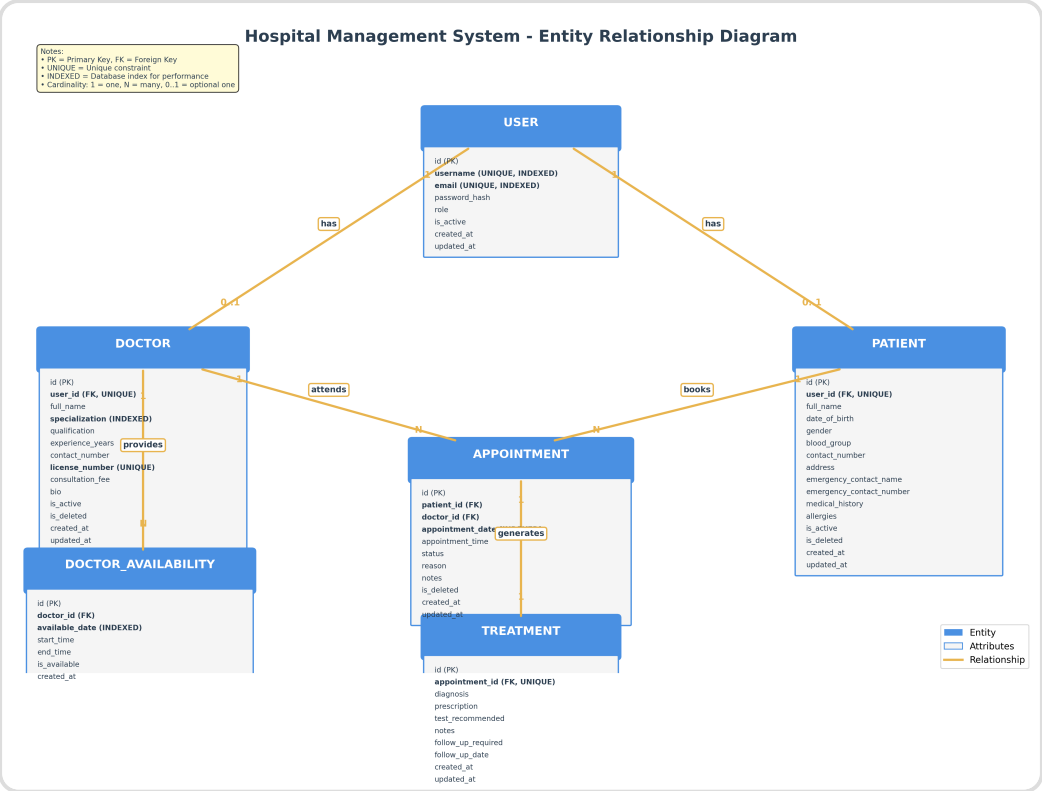
```
hospital_management_system/
├── app.py                      # Application entry point
├── config.py                   # Configuration settings
├── extensions.py               # Flask extensions
├── requirements.txt            # Dependencies
├── models/                     # Database models
│   ├── user.py
│   ├── doctor.py
│   ├── patient.py
```

```
│   ├── appointment.py
│   ├── treatment.py
│   └── doctor_availability.py
├── routes/                     # Controllers
│   ├── auth.py
│   ├── admin.py
│   ├── doctor.py
│   ├── patient.py
│   ├── api.py
│   └── main.py
├── templates/                  # Views
│   ├── admin/
│   ├── doctor/
│   ├── patient/
│   └── errors/
├── static/                     # Static assets
│   ├── css/
│   ├── js/
│   └── images/
└── utils/                      # Utilities
    ├── decorators.py
    └── helpers.py
```

Key Design Decisions

- **Soft Delete Pattern:** Records marked as deleted instead of permanent deletion for audit trails
- **Role-Based Access Control:** Three roles with custom decorators for authorization
- **Indexed Fields:** Optimized queries with indexes on frequently searched columns
- **RESTful API:** Stateless JSON endpoints with standard HTTP methods
- **Responsive Design:** Mobile-first Bootstrap approach for all devices

4. Database Design - ER Diagram



Entity Relationships

Relationship	Cardinality	Description
USER ↔ DOCTOR	1:1	One user account for one doctor profile
USER ↔ PATIENT	1:1	One user account for one patient profile
PATIENT ↔ APPOINTMENT	1:N	One patient can have multiple appointments
DOCTOR ↔ APPOINTMENT	1:N	One doctor can handle multiple appointments
APPOINTMENT ↔ TREATMENT	1:1	One appointment generates one treatment record
DOCTOR ↔ AVAILABILITY	1:N	One doctor can have multiple availability slots

5. Features Implemented

Admin Portal Features

- ✓ Dashboard with real-time statistics
- ✓ Pre-existing admin account (programmatic)
- ✓ CRUD operations for doctors and patients
- ✓ View all appointments (past and upcoming)
- ✓ Advanced search by name/specialization/contact
- ✓ Edit and manage profiles
- ✓ Soft delete/deactivate users

Doctor Portal Features

- ✓ Dashboard with upcoming appointments
- ✓ Assigned patient list
- ✓ Mark appointments as completed/canceled
- ✓ Set availability for next 7 days
- ✓ Provide diagnosis and prescriptions
- ✓ View complete patient medical history
- ✓ Profile management

Patient Portal Features

- ✓ Self-registration system
- ✓ Browse doctors by specialization
- ✓ View doctor availability
- ✓ Book and cancel appointments
- ✓ View appointment history
- ✓ Access treatment records

✓ Update personal profile

Optional Features

- ✓ Complete REST API with CRUD operations
- ✓ Chart.js data visualizations
- ✓ HTML5 and backend form validation
- ✓ Bootstrap 5.3 responsive design
- ✓ Flask-Login authentication system
- ✓ Soft delete data preservation
- ✓ Advanced search and filtering

6. API Endpoints Documentation

Doctor Endpoints

Method	Endpoint	Auth	Description
GET	/api/doctors	None	Get all active doctors (with filters)
GET	/api/doctors/<id>	None	Get specific doctor details
POST	/api/doctors	Admin	Create new doctor profile
PUT	/api/doctors/<id>	Admin	Update doctor information
DELETE	/api/doctors/<id>	Admin	Soft delete doctor

Patient Endpoints

Method	Endpoint	Auth	Description
GET	/api/patients	Required	Get all patients
POST	/api/patients	None	Register new patient
PUT	/api/patients/<id>	Patient/ Admin	Update patient profile

Appointment Endpoints

Method	Endpoint	Auth	Description
GET	/api/appointments	Required	Get appointments (with filters)
POST	/api/appointments	Required	Book new appointment
PUT	/api/appointments/<id>	Required	Update appointment status
DELETE	/api/appointments/<id>	Required	Cancel appointment

Statistics Endpoint

Method	Endpoint	Auth	Description
GET	/api/stats	Admin	Get system statistics

7. Setup & Installation

Prerequisites

- Python 3.8 or higher
- pip (Python package manager)

Installation Steps

```
# 1. Extract project unzip hospital_management_system.zip cd
hospital_management_system # 2. Create virtual environment
python -m venv venv source venv/bin/activate # On Windows:
venv\Scripts ctivate # 3. Install dependencies pip install -r
requirements.txt # 4. Run application python app.py # 5.
Access at http://127.0.0.1:5000
```

Default Credentials

> **Admin Account:**
> Username: `admin`
> Password: `admin123`
> Note: Created programmatically on first run

8. AI/LLM Usage Disclosure

To be filled by student during submission

This section must contain detailed description of AI/LLM assistance used during development:

1. AI Tools Used:

_____

2. Extent of Use:

_____

3. Specific Areas:

- ☐ Authentication system
- ☐ Database models
- ☐ API endpoints
- ☐ Frontend templates
- ☐ Styling/CSS
- ☐ Documentation
- ☐ None - completed without AI

4. Percentage of AI contribution:

_____%

5. Declaration:

I declare that I have clearly demarcated areas where AI/LLM tools were used. I understand the business logic and can explain all code.

Student Signature: _____

Date: _____

9. Video Presentation Link

Video Submission Details

Google Drive Link: [To be filled by student]

Access: Ensure link is set to "Anyone with the link"

Video Requirements:

- Duration: 5-10 minutes
- Introduction (30 seconds)
- Problem approach (30 seconds)
- Key features demo (90 seconds)
- Optional features (30 seconds)

Content to Cover:

- Architecture explanation
- Admin portal walkthrough
- Doctor portal demonstration
- Patient portal features
- API testing (optional)
- Code highlights

10. Conclusion

This Hospital Management System successfully implements all core requirements and several optional features, providing a comprehensive solution for hospital administration, doctor workflows, and patient care management.

Key Achievements

- Full-stack web development using Flask framework
- Database design and ORM implementation with SQLAlchemy
- RESTful API development with proper authentication
- Role-based access control system
- Responsive frontend with Bootstrap 5.3
- Data visualization using Chart.js
- Modern software engineering practices (MVC, separation of concerns)

Future Enhancements

- Payment gateway integration
- SMS notifications
- Telemedicine video consultation
- Prescription print/download
- Advanced analytics dashboards
- Mobile application
- Multi-hospital support
- Pharmacy management integration

Hospital Management System - MAD-1 Project

Indian Institute of Technology Madras

Modern Application Development - I

End of Report