Author

Yellapu Gyan Sagar
24F2005346
24f2005346@ds.study.iitm.ac.in

Description

This project is a Flask-based Hospital Management System that provides role-based access for Admins, Doctors, and Patients. The Admin can manage departments, doctors, and hospital records; Doctors can access patient details and update medical records; Patients can view their profiles and appointments. The system uses SQLite for data storage, Flask Blueprints for modular API design, and a simple HTML/JS + Bootstrap frontend for dashboards. All database tables are created programmatically through `db_init.py`, and API documentation is provided through a YAML specification.

Technologies used

- **Python (Flask)** – Core backend framework used for routing, API handling, and modular architecture via Blueprints.
- **SQLite** – File-based database storing departments, doctors, patients, users, appointments, and medical records.
- **Flask-Login** – Handles role-based authentication for Admin, Doctor, and Patient.
- **Flask Blueprints** – Organizes backend into modular route files (`admin_routes`, `doctor_routes`, etc.).
- **VueJS (CDN)** – Used in the frontend for reactive UI updates, API fetching, and rendering dashboard components without page reloads.
- **Bootstrap (HTML/CSS)** – Provides responsive layouts for all dashboards.
- **Jinja2 (minimal)** – Only used for initial entry rendering; actual pages run on static HTML + VueJS.

DB Schema Design

Department: Stores hospital departments.

- **DepartmentID** (PK, AUTO)
- **Name, Location**

User: Stores login credentials.

- **UserID** (PK, AUTO)
- **Username** (Unique), **Password**
- **Role** (Admin/Doctor/Patient)
- **LinkedID** (FK to DoctorID/PatientID)

Doctor: Stores doctor details.

- **DoctorID** (PK, AUTO)
- **Name, Email, Phone**
- **DepartmentID** (FK)

Patient: Stores patient details.

- **PatientID** (PK, AUTO)
- **Name, Age, Gender**
- **DepartmentID** (FK)

Appointment: Manages doctor–patient appointments.

- **AppointmentID** (PK, AUTO)
- **DoctorID** (FK), **PatientID** (FK)
- **Date, Status**

MedicalRecord: Stores patient's diagnostic and treatment history.

- **RecordID** (PK, AUTO)
- **PatientID** (FK), **DoctorID** (FK)
- **Diagnosis, Treatment, Date**

API Design

The implemented APIs manage departments, users, doctors, patients, appointments, and medical records. Each entity includes endpoints for retrieving (GET), adding (POST), updating (PUT), and deleting (DELETE) the respective data. Role-based access is enforced using Flask-Login, and all responses are returned in JSON format to ensure smooth interaction between the frontend (VueJS + JS) and the backend.

admin_routes.py
Handles CRUD operations for Departments, Doctors, and system-wide administration.

auth_routes.py
Provides user registration and login functionality, returning session details and role information.

doctor_routes.py
Allows Doctors to view assigned patients, update medical records, and manage appointments.

patient_routes.py
Enables Patients to view their profiles, check appointments, and access their medical history.

The API structure is documented in `config.yaml`, which defines the available routes and expected request/response formats.

Architecture and Features

The project follows a modular Blueprint-based architecture, where `app.py` initializes the Flask application, registers Blueprints, and manages overall configuration. All backend logic is separated into the `routes/` folder, with dedicated files for Admin, Doctor, Patient, and Authentication functionalities. The database is structured in SQLite and initialized automatically through `db_init.py`, ensuring all tables and sample data are created before use.

The frontend uses static HTML pages enhanced with VueJS (CDN) and JavaScript (`api.js`, `session.js`) to interact with backend APIs without page reloads. Bootstrap is used for styling, while minimal Jinja2 is used only for initial HTML loading.

Key features include secure authentication for Admin, Doctor, and Patient roles, Department and Doctor management (Admin), patient profiles and appointments, and full medical record handling. Each role has its own dedicated dashboard with role-specific actions. The system also includes structured API endpoints defined via Flask Blueprints and documented using a YAML configuration for seamless integration with external clients.

Video

https://drive.google.com/file/d/1OYr7uPtLsn6WQ5sBsSTJN4hPh1I6HfeD/view?usp=sharing