Project Title: Hospital Management System (H.M.S)

Authors: Kundan Santosh Bhatkar, Gourang Waikar, Sanket Markad

Date: October 28, 2025

Supervisor: (Add supervisor name)

Institution: (Add your institution)

Abstract -------- This project implements a small Hospital Management System (H.M.S) using Flask (Python) and MySQL. The system provides user authentication, doctor management, patient booking and appointment management, and basic trigger-based auditing (via SQL triggers in the provided SQL dump). The application demonstrates a typical web-based CRUD workflow, integration with a relational database, and templated UI pages for user interactions.

Keywords: Flask, MySQL, SQLAlchemy, Jinja2, Hospital Management

---

1. Introduction ---------- Hospitals need efficient systems to manage patients, doctors, and appointments. This mini-project builds a web application to manage those workflows with a lightweight architecture: a Flask backend, Jinja2 templates for views, and MySQL as the persistent store. The main goals are ease of use, clear data modeling, and a simple UI for booking and administration.

2. Objectives ---------- - Provide login/signup for users (Doctors and Patients). - Allow Doctors to view and manage bookings. - Allow Patients to book, edit and delete appointment slots. - Store data in a MySQL database with appropriate tables and triggers to capture changes. - Provide a minimal, responsive UI and documentation for setup and usage.

3. Tools and Technologies ---------- - Python 3.8+ - Flask 1.1.2 - Flask-SQLAlchemy - MySQL / MariaDB - Jinja2 templates - Bootstrap (CSS framework) - Git & GitHub for version control

Dependencies are listed in `requirements.txt`.

4. System Design ---------- 4.1 High-level architecture - Single-process Flask app (development server) serves dynamic pages and interacts with MySQL via SQLAlchemy. - Templates are under `PROJECT/templates/` and static assets under `PROJECT/static/`.

4.2 Database schema (summary) The SQL dump `hms.sql` contains the schema and sample data. Key tables: - `user` (id, username, usertype, email, password) — stores account information. - `doctors` (did, email, doctorname, dept) — doctors list and departments. - `patients` (pid, email, name, gender, slot, disease, time, date, dept, number) — patient bookings. - `trigr` (tid, pid, email, name, action, timestamp) — audit log populated by triggers. - `test` — sample table used for testing connectivity.

Triggers capture insert/update/delete events on the `patients` table and write entries to `trigr` for auditing.

4.3 ER overview (textual) - A `User` may be a Doctor or Patient. Patients create bookings (records in `patients`). Doctors are referenced by department fields in patients and by their own `doctors` table.

5. Implementation ---------- 5.1 Code layout - `PROJECT/main.py` — application entry point,

route handlers, models, and DB setup. - `PROJECT/templates/` — Jinja2 templates (base.html, index.html, patient.html, booking.html, doctor.html, login.html, signup.html, edit.html, trigers.html, etc). - `PROJECT/static/css/virtualregister.css` — custom CSS for visuals. - `hms.sql` — SQL dump to create the database and sample data.

5.2 Major routes and functionality - `/` — homepage with carousel and search. - `/signup` — sign up new users. - `/login` — login page. - `/logout` — log out users. - `/patients` — patient booking (requires login). - `/bookings` — list bookings (Doctors can see all; Patients see their own). - `/doctors` — add doctors (POST) and view form. - `/edit/<pid>` — edit a booking. - `/delete/<pid>` — delete a booking. - `/details` — view trigger log (audit trail).

5.3 Security notes - Password storage in the provided code may not consistently use hashing. For production, use `generate_password_hash` and `check_password_hash` from Werkzeug. Also, don't commit secrets; set secret keys via environment variables.

6. Testing ---------- - A simple connectivity endpoint `/test` was included to confirm DB connectivity. - Manual testing has been performed by running the app locally and exercising signup/login, booking creation, edit, and delete flows.

7. Installation and running (Windows PowerShell) ---------- 1. Clone the repo and change directory into `hospital system`.

2. Create a virtual environment and activate it:

```powershell python -m venv venv .\venv\Scripts\Activate ```

3. Install dependencies:

```powershell pip install -r requirements.txt ```

4. Create the MySQL database and import `hms.sql` (or create schema manually). Update the connection string in `PROJECT/main.py` if needed (line with `SQLALCHEMY_DATABASE_URI`).

5. Run the app:

```powershell python PROJECT/main.py ```

Visit http://127.0.0.1:5000

8. Screenshots & UI ---------- (Referenced templates are under `PROJECT/templates/`. You can capture screenshots of the running app for: homepage, login, signup, patient booking form, bookings table, and trigger logs.)

9. Known limitations and future work ---------- - Password hashing and stronger authentication flows need to be enforced. - Replace development server with production WSGI (Gunicorn + nginx) for deployment. - Add unit and integration tests and CI pipeline enhancements. - Improve responsive layout and accessibility. - Add role-based access control and input validation hardening.

10. Conclusion ---------- This mini H.M.S demonstrates a simple, functioning web application with database integration and template-driven pages. It is a good base for further expansion into a production-ready system with improved security, testing, and UX.

Appendix A — Important files - `PROJECT/main.py` — main app and models - `PROJECT/templates/*` — HTML templates - `PROJECT/static/css/virtualregister.css` — custom CSS - `hms.sql` — database schema and sample data - `requirements.txt` — dependencies

Appendix B — Backup branch A backup branch named `backup/pre-reset-8c47079` was created and pushed
```

to the remote before updating the remote `main` branch. If you need the previous versions, view or checkout that branch on GitHub.

If you'd like I can: - Export this Markdown to PDF (requires Pandoc or other tooling), - Add ER diagram images (I can generate a schema diagram and add to `PROJECT/static/images/`), - Expand any report section with more detail or add screenshots.