**Hospital Management System**

**Author**: Afreen Sultana

**Overview**

A Python-based Hospital Management System built with **MS SQL Server** to streamline patient, doctor, and appointment management.  
This project handles patient records, doctor information, and appointment scheduling with robust exception handling and unit testing.

**Features**

* **Patient Management**: Register and maintain patient profiles with contact details and demographics.
* **Doctor Management**: Track doctors’ personal details and specializations.
* **Appointment Scheduling**: Book, update, or cancel appointments, and associate each with the correct patient and doctor.
* **Error Handling**: Custom exceptions to manage invalid lookups or missing records.

**Technology Stack**

* **Backend Language**: Python
* **Database**: Microsoft SQL Server
* **Database Connectivity**: pyodbc
* **Testing**: Python unittest framework

**Database Schema**

1. **Patient**
   * patient\_id, first\_name, last\_name, date\_of\_birth,  
     gender, contact\_number, address
2. **Doctor**
   * doctor\_id, first\_name, last\_name,  
     specialization, contact\_number
3. **Appointment**
   * appointment\_id, patient\_id, doctor\_id,  
     appointment\_date, description

**Project Structure**

HospitalManagement/

│

├─ EntityModel/ # Core entity classes: Patient, Doctor, Appointment

├─ DAO/ # Data Access Objects & service interfaces/abstract classes

├─ Exception/ # Custom exceptions

│ ├─ PatientNotFoundException

│ └─ AppointmentNotFoundException

├─ Util/ # Database context and utility classes

├─ Main/ # Application entry point (menu-driven)

└─ Tests/ # Unit test cases

**Key Python Classes**

* **Entities**: Patient, Doctor, Appointment
* **Services**: PatientService, DoctorService, AppointmentService
* **Interfaces**: IPatientService, IDoctorService, IAppointmentService
* **Utilities**: DatabaseContext, AuthenticationService (if used)
* **Exceptions**: PatientNotFoundException, AppointmentNotFoundException

**Unit Testing**

Implemented with Python’s unittest to ensure correctness and reliability.  
Important scenarios include:

* Creating and updating patient records
* Adding and retrieving doctor details
* Scheduling, updating, and canceling appointments
* Handling cases where patient or appointment IDs are not found

**How to Run**

1. **Database Setup**
   * Create the database and run SQL scripts to create the Patient, Doctor, and Appointment tables.
   * Update db\_util.py with your SQL Server connection details.
2. **Install Dependencies**
3. pip install pyodbc
4. **Run Application**
5. python main.py
6. **Run Tests**
7. python -m unittest discover Tests

**Key Learning Outcomes**

* Designed normalized relational tables and relationships in SQL Server.
* Applied Python OOP principles with clear separation of layers (Entity, DAO, Service).
* Implemented custom exception handling for reliable, user-friendly error management.
* Practiced unit testing to maintain code quality.