**HOSPITAL PATIENT RECORD & BILLING SYSTEM**

**DOCUMENTATION**

**-By**

Kadali Harshavardhan

2633272

**Table of Contents**

1. [System Overview](#system-overview)
2. [Architecture](#architecture)
3. [Integration & Dependencies](#integration--dependencies)
4. [Database Schema Overview](#database-schema-overview)
5. Setup Instructions
6. [Main Modules](#main-modules)
   * [Patient Records (patient.py)](#patient-records-patientpy)
   * [Doctor Records (doctor.py)](#doctor-records-doctorpy)
   * [Service Records (service.py)](#service-records-servicepy)
   * [Service Usage Tracking (ServiceUsageDB)](#service-usage-tracking-serviceusagedb)
   * [Appointment Management (appointment.py)](#appointment-management-appointmentpy)
   * [Billing Management (billing.py)](#billing-management-billingpy)
7. [Core Workflows](#core-workflows)
   * [Appointment Workflow](#appointment-workflow)
   * [Billing Workflow](#billing-workflow)
8. [Error Handling & Validation](#error-handling--validation)
9. [Reporting & Export](#reporting--export)
10. [Extensibility](#extensibility)
11. CLI Workflow Images
12. [Conclusion](#conclusion)

**HOSPITAL PATIENT RECORD & BILLING SYSTEM**

1. **SYSTEM OVERVIEW**

The Hospital Patient Record & Billing System is designed to streamline the management of patient appointments and billing in a healthcare setting. It leverages a MySQL database for persistent storage and provides robust validation, reporting, and error handling to ensure data integrity and operational efficiency.

1. **ARCHITECTURE**

* **Programming Language**: Python
* **Database**: MySQL
* **Modularity**: The system is organized into separate modules for appointment and billing management.
* **Database Connectivity**: All database operations are performed using a get\_connection() function from a shared db\_config.py module.

1. **DEPENDENCIES**

* Python 3.x
* MySQL database
* mysql-connector-python for database connectivity
* CSV module for export functionality

**Notes**

* The db\_config.py file must be present and correctly configured for database access.
* The ServiceUsageDB class (used in billing) is expected to be defined in a separate service.py module.

1. **DATABASE SCHEMA OVERVIEW**

The system expects the following tables (not exhaustive):

* appointments: Stores appointment details (ID, patient, doctor, date, diagnosis, charge).
* patients: Patient master records.
* doctors: Doctor master records.
* billing: Bill master records (ID, patient, amount, date).
* services: Hospital services available.
* billed\_services: Services billed under each bill.
* temp\_service\_usage: Temporary records of services used by patients before billing.

A screenshot of a computer

AI-generated content may be incorrect.

1. **SETUP INSTRUCTIONS**

Follow the steps below to install and run the Hospital Patient Record & Billing System on your local machine.

**5.1. Prerequisites**

Ensure the following are installed on your system:

* **Python 3.7+**
* **MySQL Server**
* **pip** (Python package manager)

**5.2. Clone the Repository**

If you are using version control (e.g., GitHub), clone the project:

*```*

*git clone https://github.com/your-username/hospital-mgmt.git*

*cd hospital-mgmt*

*```*

**5.3. Install Required Python Packages**

Install all dependencies listed in the requirements.txt file:

*pip install -r requirements.txt*

Required packages include:

* mysql-connector-python
* tabulate

**5.4. Configure the Database Connection**

Open the db\_config.py file and update it with your MySQL credentials:

*# db\_config.py*

*import mysql.connector*

*def get\_connection():*

*return mysql.connector.connect(*

*host="localhost",*

*user="your\_mysql\_user",*

*password="your\_mysql\_password",*

*database="your\_database\_name"*

*)*

**5.5. Create the Database & Tables**

Start your MySQL server, and create the required database and tables. Use the provided schema or SQL script, or manually create these tables:

* patients
* doctors
* appointments
* services
* billing
* billed\_services
* temp\_service\_usage

Optionally, create indexes or foreign key constraints as needed.

**5.6. Run the Application**

Start the application using the main script:

*python hospital\_main.py*

Follow the on-screen instructions to manage patients, appointments, billing, and services.

**Notes**

* Ensure MySQL is running before launching the app.
* Use the tabulate library for better output formatting.
* Regularly export your data for backup and reporting.

1. **Main Modules**
   1. **Patient Records (patient.py)**

The patient.py module is responsible for managing patient records in the Hospital Appointment & Billing Management System. It provides functionality to add, update, delete, and retrieve patient information, ensuring data integrity and seamless integration with other modules such as appointments and billing.

**Class: Patient**

**Purpose**

Encapsulates all attributes and operations related to a patient’s record.

**Attributes**

* patient\_id: Unique identifier for the patient (string or integer, usually auto generated).
* name: Full name of the patient (string).
* age: Age of the patient (integer).
* gender: Gender of the patient (string, e.g., 'Male', 'Female', 'Other').
* address: Residential address (string).
* phone: Contact phone number (string).
* email: Email address (string, optional).

**Methods**

**1. add ()**

* Purpose: Adds a new patient record to the database.
* Validation:
  + Checks for valid name, age, gender, and contact details.
  + Ensures no duplicate patient ID or phone number.
* Database Operation: Inserts a new row into the patient’s table.

**2. update ()**

* Purpose: Updates details for an existing patient.
* Validation:
  + Checks if the patient exists.
  + Validates updated fields.
* Database Operation: Updates the row in the patients table where patient\_id matches.

**3. delete(patient\_id)**

* Purpose: Deletes a patient record by ID.
* Validation:
  + Checks if the patient exists.
  + Ensures no dependent records in appointments or billing (or handles cascading).
* Database Operation: Deletes the row from the patient’s table.

**4. get\_by\_id(patient\_id)**

* Purpose: Retrieves a patient's details with their ID.
* Database Operation: Fetches the row from the patient’s table.

**5. view ()**

* Purpose: Lists all patients in the system.
* Database Operation: Selects all rows from the patients table and displays them in a readable format.

**6. search\_by\_name(name)**

* Purpose: Finds patients by partial or full name.
* Database Operation: Performs a LIKE query on the name field.

**7. export\_patients\_to\_csv(filename="patients.csv")**

* Purpose: Exports all patient records to a CSV file for reporting or backup.

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| patient\_id | varchar/int | Primary Key, unique ID |
| name | varchar | Patient’s full name |
| age | int | Age |
| gender | varchar | Gender |
| address | varchar | Address |
| phone | varchar | Contact number (unique) |
| email | varchar | Email address |

**Error Handling & Validation**

* Ensure all mandatory fields are provided and valid.
* Handles database integrity errors (e.g., duplicate phone numbers).
* Provides clear error messages for user actions.

**Integration with Other Modules**

* Appointments: Patient ID is referenced in the appointments table.
* Billing: Patient ID is referenced in the billing table.
* Services: Patient service usage may be tracked for billing.

**Best Practices**

* Always validate user input before database operations.
* Use parameterized queries to prevent SQL injections.
* Handle exceptions and provide user-friendly error messages.
  1. **. Doctor Records (doctor.py)**

The doctor.py module is dedicated to managing doctor records within the Hospital Appointment & Billing Management System. It provides all necessary functionality to add, update, delete, and retrieve doctor information, and ensures smooth integration with appointment scheduling and billing modules.

**Class: Doctor**

**Purpose**

Encapsulates all attributes and operations related to a doctor's record.

**Attributes**

* doctor\_id: Unique identifier for the doctor (string, e.g., "D001", auto-generated or user-defined).
* name: Full name of the doctor (string).
* specialization: Medical specialty (string, e.g., "Cardiology", "Pediatrics").
* phone: Contact phone number (string).
* email: Email address (string, optional).
* department: Department or unit in the hospital (string, optional).

**Methods**

**1. add()**

* Purpose: Adds a new doctor record to the database.
* Validation:
  + Ensures all required fields are present and valid.
  + Checks for duplicate doctor ID or phone/email.
* Database Operation: Inserts a new row into the doctors table.

**2. update()**

* Purpose: Updates details for an existing doctor.
* Validation:
  + Checks if the doctor exists.
  + Validates updated fields.
* Database Operation: Updates the row in the doctors table where doctor\_id matches.

**3. delete(doctor\_id)**

* Purpose: Deletes a doctor record by ID.
* Validation:
  + Checks if the doctor exists.
  + Ensures no dependent records in appointments (or handles cascading).
* Database Operation: Deletes the row from the doctors table.

**4. get\_by\_id(doctor\_id)**

* Purpose: Retrieves a doctor's details by their ID.
* Database Operation: Fetches the row from the doctors table.

**5. view ()**

* Purpose: Lists all doctors in the system.
* Database Operation: Selects all rows from the doctors table and displays them in a readable format.

**6. search\_by\_specialization(specialization)**

* Purpose: Finds doctors by their specialization.
* Database Operation: Performs a LIKE query on the specialization field.

**7. export\_doctors\_to\_csv(filename="doctors.csv")**

* Purpose: Exports all doctorrecords to a CSV file for reporting or backup.

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| doctor\_id | varchar | Primary Key, unique ID |
| name | varchar | Doctor’s full name |
| specialization | varchar | Medical specialty |
| phone | varchar | Contact number (unique) |
| email | varchar | Email address |
| department | varchar | Department/unit |

**Error Handling & Validation**

* Ensure all mandatory fields are provided and valid.
* Handles database integrity errors (e.g., duplicate phone numbers or emails).
* Provides clear error messages for user actions.

**Integration with Other Modules**

* Appointments: Doctor ID is referenced in the appointments table.
* Billing/Invoices: Doctor details are included in invoices for patient consultations.
* Reporting: Doctor data can be exported and used in hospital analytics.

**Best Practices**

* Always validate user input before database operations.
* Use parameterized queries to prevent SQL injections.
* Handle exceptions and provide user-friendly error messages.
  1. **Service Records(service.py)**

The service.py module manages hospital services and tracks their usage by patients. It allows administrators to add, update, delete, and view available services, as well as record and retrieve which services have been used by which patients. This module is critical for accurate billing and reporting.

**Main Components**

**Class: Service**

**Purpose**

Represents a single hospital service (e.g., X-ray, Blood Test, MRI) and provides methods for CRUD (Create, Read, Update, Delete) operations on the services table.

**Attributes**

* service\_id: Unique identifier for the service (string, e.g., "S001", auto-generated or user-defined).
* service\_name: Name of the service (string).
* cost: Cost of the service (float).

**Methods**

1. **add()**
   * Adds a new service to the database after validating inputs.
2. **update()**
   * Updates the details (name, cost) of an existing service.
3. **delete(service\_id)**
   * Deletes a service from the database by its ID.
4. **get\_by\_id(service\_id)**
   * Retrieves details of a specific service.
5. **view()**
   * Displays all available services in a readable format.
6. **search\_by\_name(service\_name)**
   * Finds services by partial or full name.
7. **export\_services\_to\_csv(filename="services.csv")**
   * Exports all service records to a CSV file.

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| service\_id | varchar | Primary Key, unique ID |
| service\_name | varchar | Name of the service |
| cost | float | Cost of the service |

**Class: ServiceUsageDB**

**Purpose**

Tracks which services have been used by each patient (before billing). This is referenced in the billing process to generate accurate bills.

**Methods**

1. **add\_service\_for\_patient(patient\_id, service\_id, service\_name, cost)**
   * Records that a patient has used a particular service. Adds an entry to the temp\_service\_usage table.
2. **get\_services\_for\_patient(patient\_id)**
   * Retrieves all unbilled services used by a patient (returns a list of tuples: (service\_id, service\_name, cost)).
3. **clear\_services\_for\_patient(patient\_id)**
   * Removes all unbilled service usage records for a patient (typically called after billing is complete).
4. **view\_service\_usage()**
   * Displays all current (unbilled) service usages for all patients.

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| **patient\_id** | varchar | Foreign Key to patients |
| **service\_id** | varchar | Foreign Key to services |
| **service\_name** | varchar | Name of the service |
| **cost** | float | Cost of the service |

**Error Handling & Validation**

* Ensures all service fields are valid before database operations.
* Handles duplicate service IDs and integrity errors gracefully.
* Provides clear error messages for invalid operations.

**Integration with Other Modules**

* Billing: The billing module fetches unbilled services for a patient from ServiceUsageDB and clears them after billing.
* Appointments: Services may be associated with appointments for certain workflows.

**Best Practices**

* Always validate input data for service name and cost.
* Use parameterized queries to prevent SQL injection.
* Handle exceptions and provide user-friendly error messages.
* Regularly export service data for backup and reporting.
  1. **Appointment Management (appointment.py)**

The appointment.py module manages all operations related to patient appointments in the Hospital Management System. It provides functionalities for creating, updating, deleting, viewing, filtering, and exporting appointments. The module ensures data validation and integrates with a MySQL database for persistent storage.

**Function: auto\_appt\_id()**

**Purpose:**  
Automatically generates the next available unique appointment ID in the format A### (e.g., A001, A002).

**How it works:**

* Connects to the database.
* Retrieves all appointment IDs starting with 'A'.
* Extracts the numeric part, finds the maximum, and increments it.
* Returns the next appointment ID as a string.

**Class: Appointment**

**Attributes**

* appt\_id: Appointment ID (string, e.g., "A001").
* patient\_id: Patient ID (integer as string).
* doctor\_id: Doctor ID (string).
* date: Appointment date (string, format: YYYY-MM-DD).
* diagnosis: Diagnosis description (string).
* consulting\_charge: Consultation charge (float).

**Methods**

**1. add(self)**

* Purpose: Adds a new appointment to the database.
* Validation:
  + Checks for valid patient ID (numeric), doctor ID (string), date (YYYY-MM-DD), diagnosis (string), and consulting charge (positive float).
* Operation:
  + Inserts a new record into the appointments table.
  + Handles duplicate appointment IDs and other integrity errors.

**2. update(self)**

* Purpose: Updates an existing appointment in the database.
* Validation:
  + Similar to add, but also ensures consulting charge is within a reasonable range (0–10,000).
* Operation:
  + Updates the record in the appointments table for the given appt\_id.

**3. get\_by\_id(appt\_id) *(staticmethod)***

* Purpose: Fetches a single appointment record by its ID.
* Operation:
  + Returns the appointment as a dictionary, or None if not found.

**4. delete(appt\_id) *(staticmethod)***

* Purpose: Deletes an appointment by its ID.
* Operation:
  + Removes the record from the appointments table.
  + Returns True if successful, False otherwise.

**5. view() *(staticmethod)***

* Purpose: Displays all appointments in the system.
* Operation:
  + Prints all appointments in a tabular format.

**6. filter\_appointments() *(staticmethod)***

* Purpose: Filters and displays appointments within a user-specified date range.
* Operation:
  + Prompts for start and end dates.
  + Displays matching appointments.

**7. days\_between\_appointments(patient\_id) *(staticmethod)***

* Purpose: Calculates the number of days between consecutive appointments for a given patient.
* Operation:
  + Fetches all appointment dates for the patient, ordered chronologically.
  + Calculates and prints the days between each appointment.

**8. export\_appointment\_summary\_to\_csv(filename="appointment\_summary.csv") *(staticmethod)***

* Purpose: Exports all appointment records to a CSV file.
* Operation:
  + Writes appointment data to the specified CSV file.

**Error Handling & Validation**

* Validates all input fields before performing database operations.
* Handles database integrity errors, such as duplicate IDs.
* Provides clear, user-friendly error messages for invalid input or failed operations.
* Ensures database connections and cursors are closed after each operation.

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| appt\_id | varchar | Appointment ID (Primary Key) |
| patient\_id | int | Patient ID (Foreign Key) |
| doctor\_id | varchar | Doctor ID (Foreign Key) |
| date | date | Appointment date |
| diagnosis | varchar | Diagnosis description |
| consulting\_charge | float | Consultation charge |

**Integration**

* **Patient and Doctor Modules**: Uses patient\_id and doctor\_id as foreign keys; these must exist in their respective tables.
* **Billing Module**: Appointment data (including consulting charge) is referenced during invoice generation.

**Best Practices**

* Always use the auto\_appt\_id() function to generate new appointment IDs.
* Validate all user input before adding or updating records.
* Regularly export appointment data for backup and compliance.
* Handle exceptions gracefully and log errors for troubleshooting.
  1. **. Billing Management (billing.py)**

The billing.py module manages all billing operations in the Hospital Management System. It provides functionalities to create, update, delete, view, and retrieve bills, as well as generate detailed invoices for patients. The module integrates with the services and appointments modules, ensuring that all billed services and consultation charges are accurately recorded and reported.

**Function: auto\_bill\_id()**

**Purpose:**Automatically generates the next available unique bill ID in the format B### (e.g., B001, B002).

**How it works:**

* Connects to the database.
* Retrieves all bill IDs starting with 'B' followed by digits.
* Extracts the numeric part, finds the maximum, and increments it.
* Returns the next bill ID as a string.

**Class: Bill**

**Attributes**

* bill\_id: Bill ID (string, e.g., "B001").
* patient\_id: Patient ID (string).
* billing\_date: Billing date (string, format: YYYY-MM-DD, defaults to today if not provided).

**Methods**

**1. add(self)**

* Purpose: Adds a new bill to the database for a patient.
* Validation:
  + Checks for the presence of patient ID.
  + Validates billing date format.
  + Ensures the patient exists in the database.
  + Fetches all unbilled services for the patient from temp\_service\_usage.
* Operation:
  + Calculates the total amount for all services.
  + Inserts a new record into the billing table.
  + Inserts each billed service into the billed\_services table.
  + Clears temporary service usage for the patient after billing.
  + Handles duplicate bill IDs and other integrity errors.

**2. update(self)**

* Purpose: Updates an existing bill in the database.
* Validation:
  + Checks for the presence of bill ID and patient ID.
  + Validates billing date format.
  + Ensures the patient exists in the database.
  + Fetches all unbilled services for the patient from temp\_service\_usage.
* **Operation:**
  + Recalculates the total amount for all services.
  + Updates the record in the billing table for the given bill\_id.
  + Clears temporary service usage for the patient after updating the bill.

**3. get\_by\_id(bill\_id) *(staticmethod)***

* Purpose: Fetches a single bill record by its ID.
* Operation:
  + Returns the bill as a dictionary, or None if not found.

**4. delete(bill\_id) *(staticmethod)***

* Purpose: Deletes a bill by its ID.
* Operation:
  + Removes the record from the billing table.
  + Returns True if successful, False otherwise.

**5. view() *(staticmethod)***

* Purpose: Displays all bills in the system.
* Operation:
  + Prints all bills in a tabular format.

**6. generate\_invoice(self)**

* Purpose: Generates and prints a detailed invoice for the bill, including patient, doctor, appointment, and service details.
* Operation:
  + Fetches patient details.
  + Fetches the latest appointment and doctor details for the patient.
  + Fetches all billed services for the bill.
  + Calculates totals and formats the invoice for display.

**Database Tables:**

1. Table: billing

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| bill\_id | varchar | Bill ID (Primary Key) |
| patient\_id | varchar | Patient ID (Foreign Key) |
| total\_amount | float | Total billed amount |
| billing\_date | date | Billing date |

1. Table: billed\_services (Optional)

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| bill\_id | varchar | Bill ID (Foreign Key) |
| patient\_id | varchar | Patient ID (Foreign Key) |
| service\_id | varchar | Service ID (Foreign Key) |
| service\_name | varchar | Name of the service |
| cost | float | Cost of the service |

1. Table: temp\_service\_usage (Optional)

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| patient\_id | varchar | Patient ID |
| service\_id | varchar | Service ID |
| service\_name | varchar | Name of the service |
| cost | float | Cost of the service |

**Error Handling & Validation**

* Validates all input fields before performing database operations.
* Handles database integrity errors, such as duplicate IDs.
* Provides clear, user-friendly error messages for invalid input or failed operations.
* Ensures database connections and cursors are closed after each operation.

**Integration**

* **Service Module**: Uses ServiceUsageDB to fetch and clear unbilled services for a patient.
* **Patient and Doctor Modules**: References patient and doctor details for invoice generation.
* **Appointment Module**: Fetches latest appointment and consulting charge for invoice.

**Best Practices**

* Always use the auto\_bill\_id() function to generate new bill IDs.
* Validate all user input before adding or updating records.
* Regularly review and export billing data for backup and compliance.
* Handle exceptions gracefully and log errors for troubleshooting.

**7. CORE WORKFLOWS**

**7.1. Appointment Workflow**

* **Create Appointment**
  + System generates a unique appointment ID.
  + User provides patient ID, doctor ID, date, diagnosis, and consulting charge.
  + Input is validated and stored in the database.
* **Update Appointment**
  + User specifies the appointment ID and updated details.
  + System validates and updates the record.
* **Delete Appointment**
  + User specifies the appointment ID to remove.
  + System deletes the record if it exists.
* **View & Export**
  + All appointments can be viewed in a tabular format or exported as a CSV for reporting.
* **Date Filtering & Analysis**
  + Appointments can be filtered by date range.
  + Days between a patient's appointments can be calculated for analysis.
  1. **Billing Workflow**
* **Create Bill**
  + System generates a unique bill ID.
  + User provides patient ID and (optionally) billing date.
  + System fetches all unbilled services used by the patient.
  + Total amount is calculated and stored.
  + Each service is recorded in the billed\_services table.
  + Temporary service usage records are cleared.
* **Update Bill**
  + Bill details and amount can be updated as needed.
* **Delete Bill**
  + Bill can be removed by specifying its ID.
* **View Bills**
  + All bills can be listed for administrative review.
* **Generate Invoice**
  + Produces a formatted invoice showing:
    1. Bill and patient details
    2. Doctor and appointment information
    3. Itemized list of services and charges
    4. Consultation charge and total amount

1. **Error Handling & Validation**

* All user inputs are validated for correct format (IDs, dates, charges).
* Database integrity is enforced via exception handling.
* Duplicate IDs and missing records are reported with clear messages.
* Temporary service usage is cleared after successful billing to prevent duplicate charges.

1. **Reporting & Export**

* **Appointment Summaries** can be exported as CSV files for external analysis or compliance.
* **Invoices** are generated in a clear, human-readable format for printing or digital sharing.

1. **Extensibility**

* The system is modular and can be extended to include more features such as:
  + Patient registration and management
  + Advanced reporting and analytics
  + User authentication and access control

1. **CLI WORFLOW IMAGES**

Main Menu of each record

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

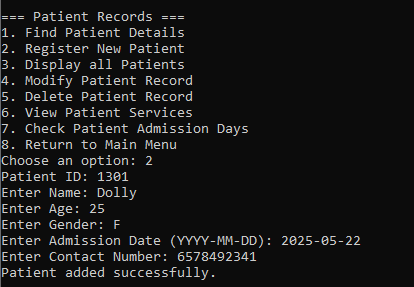
A screenshot of a computer screen

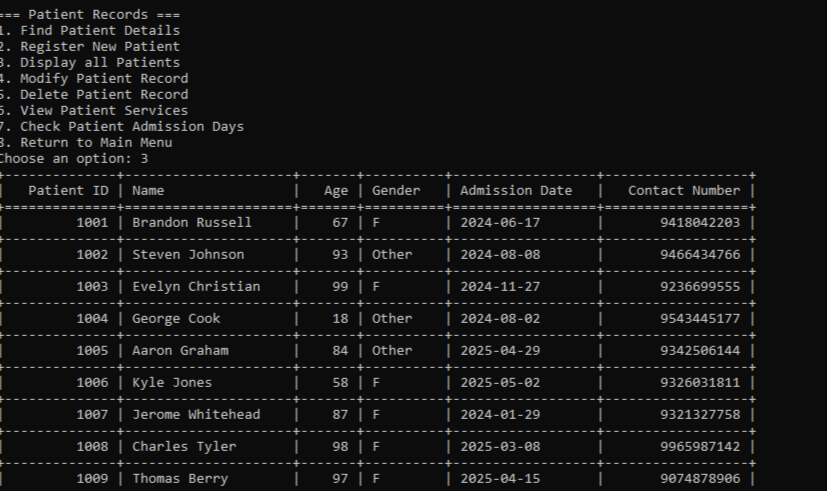
AI-generated content may be incorrect.

Patient Record

A computer screen shot of a black screen

AI-generated content may be incorrect.





A computer screen shot of a patient record

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect. A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

Doctor Records

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Service Records

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Appointment Records

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

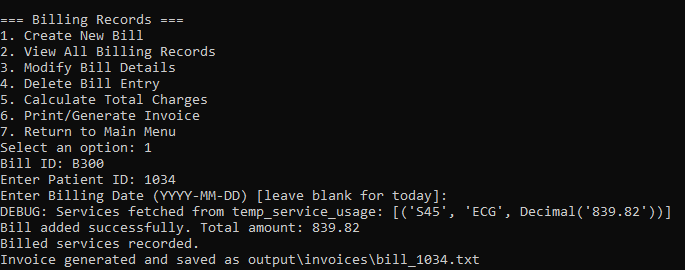
A screenshot of a computer

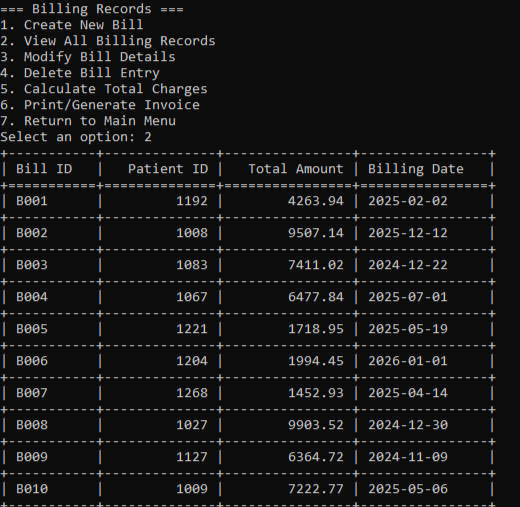
AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Billing Records





A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Export Records

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

1. **CONCLUSION**

This system provides a robust backend for hospital appointments and billing management, ensuring data integrity, operational efficiency, and ease of reporting. It is suitable for integration into larger hospital management solutions or use as a standalone administrative tool.