

**A PROJECT REPORT ON**

**HOSPITAL AUTOMATION- DOCUMENT TRACKING**

**SUBMITTED BY**

|  |  |  |
| --- | --- | --- |
| **NAME** | : | Alper Can  Mustafa Efe Özkan  Rafet Bartuğ Bartınlı  Ömer Talha Yurtturan |
| **ID** | : | 2221251040  2221251030  2221251034  2221251026 |
| **COURSE** | : | Database System Design |

**1-Aim of The Project and Scope**

* 1. **Aim of the Project:**

The aim of this Hospital Management and Document Tracking System is to streamline hospital operations and facilitate the management and accessibility of critical information, such as appointments, prescriptions, invoices, and medical records. Here's a detailed breakdown of its objectives:

**Primary Goals**

**Automation of Hospital Operations:**

Digitally manage appointments, prescriptions, lab tests, admissions, and patient discharges.

Eliminate manual errors by automating billing, invoicing, and report generation.

**Improved Patient Care:**

Ensure timely and accurate information sharing between doctors, patients, and administrative staff.

Provide a centralized platform for patients to view their medical data, such as prescriptions, test results, and invoices.

**Efficient Document Management:**

Track and manage important documents such as medical reports, invoices, and discharge summaries.

Facilitate document sharing between authorized users (e.g., doctors, patients, nurses).

**Enhanced Billing and Invoicing:**

Automate billing processes for appointments, prescriptions and room stays.

Provide patients with clear, itemized invoices for transparency.

Seamless Communication:

Enable better communication between doctors and patients through shared records and updates.

Allow patients to view their medical information without the need for physical visits.

**Key Features of the System**

For Patients:

Appointments: View and manage upcoming and past appointments.

Prescriptions: Access prescriptions with details of medications and dosages.

Lab Results: Review lab test results.

Invoices and Bills: View and pay invoices for appointments, prescriptions, and hospital stays.

Reports: Access discharge summaries and medical reports.

**For Doctors:**

Patient Management: View patient profiles, medical histories, and manage appointments.

Prescriptions: Write, update, and issue prescriptions digitally.

Billing: Assign consultation fees and manage billing status.

Reports: Generate medical and discharge reports.

**Document Management:**

Store and manage documents (invoices, reports, prescriptions) linked to patients and treatments.

Control access permissions (e.g., only patients and authorized staff can view specific documents).

**Hospital Operations:**

Admissions and Discharges: Automate the patient admission and discharge processes.

Room Management: Assign room types (General, ICU, Private) with associated costs.

Inventory Management: Track medications in the pharmacy and update quantities dynamically.

**Benefits of the System**

Time-Saving:

Automates routine tasks like billing, report generation, and prescription creation.

Reduces waiting times for patients by enabling online appointments and access to records.

Improved Accuracy:

Minimizes errors in billing, prescriptions, and patient data.

Ensures accurate tracking of room charges and medication usage.

Transparency:

Provides patients with clear, itemized bills and real-time access to medical data.

Builds trust by allowing patients to review their records anytime.

Compliance and Record-Keeping:

Ensures that all patient data and medical records are securely stored and accessible for audits or legal compliance.

Facilitates easy retrieval of records for patients and doctors.

Centralized System:

All data is stored in a single system, making it easier for the hospital staff to manage and for patients to access.

Vision

The ultimate aim of the system is to build a smart, paperless hospital management system that prioritizes:

Patient satisfaction,

Operational efficiency,

Cost-effectiveness,

Data security and privacy.

By combining hospital workflows with document tracking and management capabilities, the system aspires to be a comprehensive solution for healthcare facilities of all sizes.

* 1. **Project Scope:**

This project is designed to digitize and centralize hospital operations while integrating a robust document tracking system to manage critical records like invoices, prescriptions, and medical reports. The system targets doctors, patients, and hospital staff to enhance efficiency, reduce errors, and improve patient care through seamless communication and document management.

2. Functional Scope

2.1 Patient Management

Patient registration and profile management.

View and manage personal medical history, prescriptions, lab results, invoices, and reports.

Book, modify, or cancel appointments with doctors.

2.2 Appointment Management

Doctors can create, update, and manage patient appointments.

Patients can view their scheduled appointments and associated notes.

Integrated billing for appointments, including consultation fees.

2.3 Document Management

Types of Documents:

Prescriptions

Lab Test Results

Invoices and Bills

Discharge Summaries

Medical Reports

Automate the generation, storage, and access of documents.

Provide patients with read-only access to their documents.

Ensure only authorized users (e.g., doctors, patients) can access specific documents.

2.4 Prescription Management

Doctors can write and issue prescriptions with dynamic medication details.

Patients can view their prescriptions with an itemized breakdown of medication costs.

Pharmacy inventory integration to update stock levels dynamically.

2.5 Billing and Invoicing

Automate billing for:

Doctor appointments.

Room stays during patient admissions.

Prescriptions and lab tests.

Generate itemized invoices with detailed descriptions of charges.

Maintain paid and unpaid status for invoices.

Integration with the document system for creating and managing invoices.

2.6 Admission and Discharge Management

Manage patient admissions with room assignments (General, ICU, Private) and calculate daily room charges.

Automate patient discharge processes, including:

Calculating total charges for the stay based on room type and duration.

Generating a discharge summary and billing documents.

Creating a medical report for the patient upon discharge.

2.7 Lab Test Management

Request lab tests and assign lab technicians.

Store and retrieve lab test results linked to the patient and appointment.

Generate lab test invoices for billing.

2.8 Room Management

Track room availability and manage patient assignments.

Calculate room costs based on room type (General, ICU, Private).

Automate the release of rooms upon patient discharge.

2.9 Reporting and Analytics

Generate detailed reports for hospital management:

Revenue reports (from billing and invoices).

Patient treatment history.

Room occupancy statistics.

Lab test usage and results summary.

3. Technical Scope

3.1 Database Design

Utilize SQL databases to manage entities:

Users: Doctors, Patients, Nurses, Lab Technicians.

Appointments: Scheduling, status, and consultation details.

Documents: Prescriptions, invoices, reports, lab results.

Pharmacy Inventory: Medicines and stock management.

Admissions: Room assignments, charges, and discharge details.

Maintain relationships between tables for seamless integration.

3.2 Frontend

Java Swing-based UI:

Separate interfaces for doctors, patients, and administrators.

Interactive tables for viewing appointments, lab results, invoices, etc.

Simple and user-friendly navigation.

3.3 Backend

Java-based backend logic to handle:

Database operations (insert, update, retrieve).

Business rules (e.g., automating billing or document generation).

Transaction management for critical operations like billing and discharges.

3.4 Security

Role-based access control to restrict access to sensitive data.

Encrypted storage of sensitive patient data (e.g., passwords, medical details).

Audit trails for document access and updates.

3.5 Document Management

Generate documents dynamically upon triggering events (e.g., discharge, appointment completion).

Store documents securely in the Documents table and link them to the appropriate patient or appointment.

4. Out of Scope

Real-time payment processing or integration with external payment gateways.

Integration with third-party systems like health insurance providers.

Advanced AI/ML-based diagnostics or decision support.

Hardware integrations like RFID for room assignments or patient tracking.

Mobile application development (limited to desktop-based Java Swing).

5. Project Boundaries

Target Users:

Patients.

Doctors.

Administrative staff (e.g., nurses, lab technicians).

Supported Operations:

Hospitals or clinics with general departments and specialty divisions.

Deployment Platform:

Local desktop systems running Java and SQL databases.

System Limitations:

Single-hospital deployment (multi-location integration not included).

6. Key Deliverables

Fully functional Hospital Management System.

Dynamic Document Management System.

Comprehensive and accurate Billing and Invoicing System.

Intuitive Patient and Doctor Interfaces.

End-to-end Admission and Discharge Workflow.

7. Expected Outcomes

Streamlined hospital operations with minimal manual intervention.

Enhanced patient experience with digital access to their medical records.

Reduced errors in billing, prescriptions, and documentation.

Improved communication and coordination between hospital staff.

2. DATABASE DESIGN

2.1 Table Explanations:

**1. Users Table**

* Purpose: Stores basic information for all system users, including patients and doctors.
* Key Columns:
  + UserID: Primary key to uniquely identify each user.
  + FirstName and LastName: User's name.
  + Email: Unique identifier for the user, ensuring no duplicate entries.
  + Password: Used for authentication.
  + UserType: Specifies whether the user is a "Doctor" or "Patient".
  + RegistrationDate: Records the date of user registration.

metin, ekran görüntüsü, doküman, belge, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

**2. Patients Table**

* **Purpose**: Contains details specific to patients, linked to their UserID.
* **Key Columns**:
  + PatientID: Primary key for identifying patients.
  + UserID: Foreign key referencing the Users table.
  + DateOfBirth: Records the patient's birth date.
  + Gender: Specifies the patient's gender (Male, Female, Other).
  + ContactNumber and Address: Contact details of the patient.
  + RegistrationDate: Date when the patient was registered in the system.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3. Doctors Table**

* **Purpose**: Stores doctor-specific information.
* **Key Columns**:
  + DoctorID: Primary key for identifying doctors.
  + UserID: Foreign key referencing the Users table.
  + Specialization: Doctor's field of expertise (e.g., Cardiology, Neurology).
  + ContactNumber and AvailabilityStatus: Indicates the doctor’s contact details and availability.
  + JoinDate: Date the doctor joined the system.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**4. Appointments Table**

* **Purpose**: Tracks appointments between doctors and patients.
* **Key Columns**:
  + AppointmentID: Primary key for each appointment.
  + PatientID: Foreign key referencing the Patients table.
  + DoctorID: Foreign key referencing the Doctors table.
  + AppointmentDate: Scheduled date and time of the appointment.
  + Status: Indicates the appointment status (Scheduled, Completed, Cancelled).
  + Notes: Additional information about the appointment.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**5. Billing Table**

* **Purpose**: Manages financial details for appointments.
* **Key Columns**:
  + BillID: Primary key for each bill.
  + AppointmentID: Foreign key referencing the Appointments table.
  + TotalAmount: Total cost of the appointment.
  + PaymentStatus: Indicates whether the bill is "Paid" or "Unpaid".
  + IssueDate: Date the bill was generated.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**6. PharmacyInventory Table**

* **Purpose**: Tracks the inventory of medications in the pharmacy.
* **Key Columns**:
  + ItemID: Primary key for each inventory item.
  + ItemName: Name of the medication.
  + Quantity: Available stock of the item.
  + PricePerUnit: Price per unit of the medication.
  + ExpiryDate: Expiration date of the medication.
  + SupplierName: Name of the supplier providing the medication.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**7. Prescriptions Table**

* **Purpose**: Records prescriptions issued by doctors to patients.
* **Key Columns**:
  + PrescriptionID: Primary key for each prescription.
  + PatientID: Foreign key referencing the Patients table.
  + DoctorID: Foreign key referencing the Doctors table.
  + DateIssued: Date when the prescription was issued.
  + Notes: Additional instructions or notes for the prescription.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**8. PrescriptionItems Table**

* **Purpose**: Stores details of medications included in each prescription.
* **Key Columns**:
  + PrescriptionItemID: Primary key for each item entry.
  + PrescriptionID: Foreign key referencing the Prescriptions table.
  + ItemID: Foreign key referencing the PharmacyInventory table.
  + Quantity: Quantity of the medication prescribed.

metin, sayı, numara, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**9. Staff Table**

* **Purpose**: Manages details of hospital staff, including nurses and lab technicians, and their roles in patient care.
* **Key Columns**:
  + StaffID: Primary key for each staff member.
  + FirstName/LastName: Full name of the staff member.
  + Role: Designation, such as "Nurse" or "Lab Technician.".
  + ContactNumber: Phone number for communication.
  + Email: Staff email address.
  + JoinDate: Date the staff member joined the hospital

metin, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

**10. Documents Table**

* **Purpose**: Manages various documents, such as invoices and reports.
* **Key Columns**:
  + DocumentID: Primary key for each document.
  + DocumentType: Type of document (Invoice, Prescription, Report).
  + RelatedID: Links the document to a specific record.
  + CreatedBy and CreatedFor: References the creator (doctor) and recipient (patient).
  + CreationDate: Date the document was created.
  + Status: Indicates the document's status (Draft, Completed, Approved).

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**11. DocumentAccess Table**

* **Purpose**: Tracks access permissions for documents.
* **Key Columns**:
  + AccessID: Primary key for each access record.
  + DocumentID: Foreign key referencing the Documents table.
  + UserID: Foreign key referencing the Users table.
  + AccessLevel: Specifies the level of access (Read, Write).
  + AccessDate: Date when access was granted.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**12. InvoiceDetails Table**

* **Purpose**: Stores details of individual line items in invoices.
* **Key Columns**:
  + InvoiceDetailID: Primary key for each invoice detail.
  + DocumentID: Foreign key referencing the Documents table.
  + ItemDescription: Description of the invoiced item (e.g., room type, medication).
  + Quantity: Quantity of the item.
  + UnitPrice: Price per unit of the item.
  + TotalPrice: Calculated column (Quantity \* UnitPrice) representing the total cost.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**13. ReportDetails Table**

* **Purpose**: Stores details of reports associated with documents.
* **Key Columns**:
  + ReportID: Primary key for each report detail.
  + DocumentID: Foreign key referencing the Documents table.
  + ReportTitle: Title of the report.
  + ReportContent: Content of the report providing detailed information.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**14. PrescriptionDetails Table**

* **Purpose**: Tracks medications included in prescriptions, linked to documents.
* **Key Columns**:
  + PrescriptionDetailID: Primary key for each prescription detail.
  + DocumentID: Foreign key referencing the Documents table.
  + ItemID: Foreign key referencing the PharmacyInventory table.
  + Quantity: Quantity of the medication prescribed.

metin, ekran görüntüsü, sayı, numara, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu

**15. Rooms Table**

* **Purpose**: Tracks hospital room availability and usage.
* **Key Columns**:
  + RoomID: Primary key for each room.
  + RoomType: Type of room (General, ICU, Private).
  + RoomNumber: Unique identifier for the room.
  + AvailabilityStatus: Indicates whether the room is "Available" or "Occupied".

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**16. Admissions Table**

* **Purpose**: Manages patient admissions to rooms, assigning responsibilities to doctors and nurses.
* **Key Columns**:
  + AdmissionID: Primary key for each admission.
  + PatientID: Foreign key referencing the Patients table.
  + RoomID: Foreign key referencing the Rooms table.
  + AdmissionDate and DischargeDate: Tracks the duration of admission.
  + ResponsibleDoctorID and ResponsibleNurseID: References the doctor and nurse responsible for the admission.
  + Notes: Additional information about the admission.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**17. MedicalRecords Table**

* **Purpose**: Tracks medical records for patients, created by doctors.
* **Key Columns**:
  + RecordID: Primary key for each medical record.
  + PatientID: Foreign key referencing the Patients table.
  + DoctorID: Foreign key referencing the Doctors table.
  + RecordDate: Date the record was created.
  + Description: Details of the patient's medical condition or treatment.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**18. LabTechnicians Table**

* **Purpose**: Tracks information about lab technicians who perform lab tests.
* **Key Columns**:
  + TechnicianID: Primary key for each technician.
  + FirstName and LastName: Name of the technician.
  + ContactNumber and Email: Contact details.
  + HireDate: Date when the technician joined the hospital.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**19. LabTests Table**

* **Purpose**: Records lab tests performed for patients by technicians.
* **Key Columns**:
  + TestID: Primary key for each lab test.
  + PatientID: Foreign key referencing the Patients table.
  + TechnicianID: Foreign key referencing the LabTechnicians table.
  + TestName: Name of the lab test (e.g., MRI, X-Ray).
  + TestDate: Date the test was conducted.
  + TestResult: Result of the lab test.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**20. LabTestAssignments Table**

* **Purpose**: Tracks assignments of lab tests to technicians.
* **Key Columns**:
  + AssignmentID: Primary key for each assignment.
  + TestID: Foreign key referencing the LabTests table.
  + TechnicianID: Foreign key referencing the LabTechnicians table.
  + AssignmentDate: Date the test was assigned.

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

* 1. Table Relationships:

1. Users Table

* Relationships:
  + Referenced by:
    - Patients table through the UserID column.
    - Doctors table through the UserID column.
    - DocumentAccess table through the UserID column.

2. Patients Table

* Relationships:
  + References:
    - Users table through the UserID column.
  + Referenced by:
    - Appointments table through the PatientID column.
    - Prescriptions table through the PatientID column.
    - Admissions table through the PatientID column.
    - MedicalRecords table through the PatientID column.
    - LabTests table through the PatientID column.
    - Documents table through the CreatedFor column.

3. Doctors Table

* Relationships:
  + References:
    - Users table through the UserID column.
  + Referenced by:
    - Appointments table through the DoctorID column.
    - Prescriptions table through the DoctorID column.
    - Admissions table through the ResponsibleDoctorID column.
    - MedicalRecords table through the DoctorID column.
    - Documents table through the CreatedBy column.

4. Appointments Table

* Relationships:
  + References:
    - Patients table through the PatientID column.
    - Doctors table through the DoctorID column.
  + Referenced by:
    - Billing table through the AppointmentID column.

5. Billing Table

* Relationships:
  + References:
    - Appointments table through the AppointmentID column.

6. PharmacyInventory Table

* Relationships:
  + Referenced by:
    - PrescriptionItems table through the ItemID column.
    - PrescriptionDetails table through the ItemID column.

7. Prescriptions Table

* Relationships:
  + References:
    - Patients table through the PatientID column.
    - Doctors table through the DoctorID column.
  + Referenced by:
    - PrescriptionItems table through the PrescriptionID column.

8. PrescriptionItems Table

* Relationships:
  + References:
    - Prescriptions table through the PrescriptionID column.
    - PharmacyInventory table through the ItemID column.

9. Staff Table

* Relationships:
  + Referenced by:
    - Admissions table through the ResponsibleNurseID column.

10. Rooms Table

* Relationships:
  + Referenced by:
    - Admissions table through the RoomID column.

11. Admissions Table

* Relationships:
  + References:
    - Patients table through the PatientID column.
    - Rooms table through the RoomID column.
    - Doctors table through the ResponsibleDoctorID column.
    - Staff table through the ResponsibleNurseID column.

12. MedicalRecords Table

* Relationships:
  + References:
    - Patients table through the PatientID column.
    - Doctors table through the DoctorID column.

13. LabTechnicians Table

* Relationships:
  + Referenced by:
    - LabTests table through the TechnicianID column.
    - LabTestAssignments table through the TechnicianID column.

14. LabTests Table

* Relationships:
  + References:
    - Patients table through the PatientID column.
    - LabTechnicians table through the TechnicianID column.

15. LabTestAssignments Table

* Relationships:
  + References:
    - LabTests table through the TestID column.
    - LabTechnicians table through the TechnicianID column.

16. Documents Table

* Relationships:
  + References:
    - Doctors table through the CreatedBy column.
    - Patients table through the CreatedFor column.
  + Referenced by:
    - DocumentAccess table through the DocumentID column.
    - InvoiceDetails table through the DocumentID column.
    - ReportDetails table through the DocumentID column.
    - PrescriptionDetails table through the DocumentID column.

17. DocumentAccess Table

* Relationships:
  + References:
    - Documents table through the DocumentID column.
    - Users table through the UserID column.

18. InvoiceDetails Table

* Relationships:
  + References:
    - Documents table through the DocumentID column.

19. ReportDetails Table

* Relationships:
  + References:
    - Documents table through the DocumentID column.

20. PrescriptionDetails Table

* Relationships:
  + References:
    - Documents table through the DocumentID column.
    - PharmacyInventory table through the ItemID column.
  1. Relationships Types:

**Binary Relationships**

A binary relationship involves two entities (or tables).

1. **Users → Patients**
   * **Relationship**: One User corresponds to one Patient.
   * **Cardinality**: One-to-One.
2. **Users → Doctors**
   * **Relationship**: One User corresponds to one Doctor.
   * **Cardinality**: One-to-One.
3. **Patients → Appointments**
   * **Relationship**: One Patient can have multiple Appointments.
   * **Cardinality**: One-to-Many.
4. **Doctors → Appointments**
   * **Relationship**: One Doctor can have multiple Appointments.
   * **Cardinality**: One-to-Many.
5. **Appointments → Billing**
   * **Relationship**: One Appointment can have one Billing record.
   * **Cardinality**: One-to-One.
6. **Doctors → Prescriptions**
   * **Relationship**: One Doctor can issue multiple Prescriptions.
   * **Cardinality**: One-to-Many.
7. **Patients → Prescriptions**
   * **Relationship**: One Patient can have multiple Prescriptions.
   * **Cardinality**: One-to-Many.
8. **Prescriptions → PrescriptionItems**
   * **Relationship**: One Prescription contains multiple PrescriptionItems.
   * **Cardinality**: One-to-Many.
9. **PrescriptionItems → PharmacyInventory**
   * **Relationship**: Each PrescriptionItem corresponds to one item in the PharmacyInventory.
   * **Cardinality**: One-to-One.
10. **Rooms → Admissions**
    * **Relationship**: One Room can host multiple Admissions.
    * **Cardinality**: One-to-Many.
11. **Staff → Admissions**
    * **Relationship**: One Staff member (e.g., a nurse) can be responsible for multiple Admissions.
    * **Cardinality**: One-to-Many.
12. **Patients → Admissions**
    * **Relationship**: One Patient can have multiple Admissions.
    * **Cardinality**: One-to-Many.
13. **Doctors → Admissions**
    * **Relationship**: One Doctor can supervise multiple Admissions.
    * **Cardinality**: One-to-Many.
14. **Patients → MedicalRecords**
    * **Relationship**: One Patient can have multiple MedicalRecords.
    * **Cardinality**: One-to-Many.
15. **Doctors → MedicalRecords**
    * **Relationship**: One Doctor can create multiple MedicalRecords.
    * **Cardinality**: One-to-Many.
16. **Patients → LabTests**
    * **Relationship**: One Patient can undergo multiple LabTests.
    * **Cardinality**: One-to-Many.
17. **LabTechnicians → LabTests**
    * **Relationship**: One LabTechnician can perform multiple LabTests.
    * **Cardinality**: One-to-Many.
18. **LabTests → LabTestAssignments**
    * **Relationship**: One LabTest can have multiple LabTestAssignments.
    * **Cardinality**: One-to-Many.
19. **LabTechnicians → LabTestAssignments**
    * **Relationship**: One LabTechnician can be assigned to multiple LabTests.
    * **Cardinality**: One-to-Many.
20. **Documents → DocumentAccess**
    * **Relationship**: One Document can have multiple access records.
    * **Cardinality**: One-to-Many.
21. **Users → DocumentAccess**
    * **Relationship**: One User can have access to multiple Documents.
    * **Cardinality**: One-to-Many.
22. **Documents → InvoiceDetails**
    * **Relationship**: One Document can include multiple InvoiceDetails.
    * **Cardinality**: One-to-Many.
23. **Documents → ReportDetails**
    * **Relationship**: One Document can include multiple ReportDetails.
    * **Cardinality**: One-to-Many.
24. **Documents → PrescriptionDetails**
    * **Relationship**: One Document can include multiple PrescriptionDetails.
    * **Cardinality**: One-to-Many.
25. **PharmacyInventory → PrescriptionDetails**
    * **Relationship**: One PharmacyInventory item can be referenced in multiple PrescriptionDetails.
    * **Cardinality**: One-to-Many.

**Ternary Relationships**

A ternary relationship involves three entities (or tables) simultaneously.

1. **Patients → Doctors → Admissions**
   * **Relationship**: A Patient is admitted to a Room, under the supervision of a Doctor and a Staff member (e.g., nurse).
   * **Tables Involved**: Patients, Doctors, Staff.
   * **Cardinality**: Many-to-Many.
2. **Prescriptions → PrescriptionItems → PharmacyInventory**
   * **Relationship**: A Prescription consists of multiple items (medications), each of which is stored in the PharmacyInventory.
   * **Tables Involved**: Prescriptions, PrescriptionItems, PharmacyInventory.
   * **Cardinality**: Many-to-Many.
3. **LabTests → LabTestAssignments → LabTechnicians**
   * **Relationship**: A LabTest is assigned to a LabTechnician for completion.
   * **Tables Involved**: LabTests, LabTestAssignments, LabTechnicians.
   * **Cardinality**: Many-to-Many.
4. **Documents → DocumentAccess → Users**
   * **Relationship**: A Document can be accessed by multiple Users with varying access levels.
   * **Tables Involved**: Documents, DocumentAccess, Users.
   * **Cardinality**: Many-to-Many.

**Binary Relationships:**

|  |  |  |
| --- | --- | --- |
| **Relationship** | **Type** | **Cardinality** |
| Users → Patients | Binary | One-to-One |
| Users → Doctors | Binary | One-to-One |
| Patients → Appointments | Binary | One-to-Many |
| Doctors → Appointments | Binary | One-to-Many |
| Appointments → Billing | Binary | One-to-One |
| Doctors → Prescriptions | Binary | One-to-Many |
| Patients → Prescriptions | Binary | One-to-Many |
| Prescriptions → PrescriptionItems | Binary | One-to-Many |
| PrescriptionItems → PharmacyInventory | Binary | One-to-One |
| Rooms → Admissions | Binary | One-to-Many |
| Staff → Admissions | Binary | One-to-Many |
| Patients → Admissions | Binary | One-to-Many |
| Doctors → Admissions | Binary | One-to-Many |
| Patients → MedicalRecords | Binary | One-to-Many |
| Doctors → MedicalRecords | Binary | One-to-Many |
| Patients → LabTests | Binary | One-to-Many |
| LabTechnicians → LabTests | Binary | One-to-Many |
| LabTests → LabTestAssignments | Binary | One-to-Many |
| LabTechnicians → LabTestAssignments | Binary | One-to-Many |
| Documents → DocumentAccess | Binary | One-to-Many |
| Users → DocumentAccess | Binary | One-to-Many |
| Documents → InvoiceDetails | Binary | One-to-Many |
| Documents → ReportDetails | Binary | One-to-Many |
| Documents → PrescriptionDetails | Binary | One-to-Many |
| PharmacyInventory → PrescriptionDetails | Binary | One-to-Many |

**Ternary Relationships:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Relationship** | **Type** | **Cardinality** | **Tables Involved** |
| Patients → Doctors → Admissions | Ternary | Many-to-Many | Patients, Doctors, Admissions (Staff Involved) |
| Prescriptions → PrescriptionItems → PharmacyInventory | Ternary | Many-to-Many | Prescriptions, PrescriptionItems, PharmacyInventory |
| LabTests → LabTestAssignments → LabTechnicians | Ternary | Many-to-Many | LabTests, LabTestAssignments, LabTechnicians |
| Documents → DocumentAccess → Users | Ternary | Many-to-Many | Documents, DocumentAccess, Users |

* 1. **Constraints**:

**1. Primary Key Constraints**

* Each table has a unique primary key to uniquely identify records:
  + Users: UserID
  + Patients: PatientID
  + Doctors: DoctorID
  + Appointments: AppointmentID
  + Billing: BillID
  + PharmacyInventory: ItemID
  + Prescriptions: PrescriptionID
  + PrescriptionItems: PrescriptionItemID
  + Staff: StaffID
  + Rooms: RoomID
  + Admissions: AdmissionID
  + MedicalRecords: RecordID
  + LabTechnicians: TechnicianID
  + LabTests: TestID
  + LabTestAssignments: AssignmentID
  + Documents: DocumentID
  + DocumentAccess: AccessID
  + InvoiceDetails: InvoiceDetailID
  + ReportDetails: ReportID
  + PrescriptionDetails: PrescriptionDetailID

**2. Foreign Key Constraints**

* **Users → Patients**: UserID in Patients references UserID in Users.
* **Users → Doctors**: UserID in Doctors references UserID in Users.
* **Appointments → Patients**: PatientID references PatientID in Patients.
* **Appointments → Doctors**: DoctorID references DoctorID in Doctors.
* **Billing → Appointments**: AppointmentID references AppointmentID in Appointments.
* **Prescriptions → Patients**: PatientID references PatientID in Patients.
* **Prescriptions → Doctors**: DoctorID references DoctorID in Doctors.
* **PrescriptionItems → Prescriptions**: PrescriptionID references PrescriptionID in Prescriptions.
* **PrescriptionItems → PharmacyInventory**: ItemID references ItemID in PharmacyInventory.
* **Admissions → Patients**: PatientID references PatientID in Patients.
* **Admissions → Rooms**: RoomID references RoomID in Rooms.
* **Admissions → Doctors**: ResponsibleDoctorID references DoctorID in Doctors.
* **Admissions → Staff**: ResponsibleNurseID references StaffID in Staff.
* **LabTests → Patients**: PatientID references PatientID in Patients.
* **LabTests → LabTechnicians**: TechnicianID references TechnicianID in LabTechnicians.
* **LabTestAssignments → LabTests**: TestID references TestID in LabTests.
* **LabTestAssignments → LabTechnicians**: TechnicianID references TechnicianID in LabTechnicians.
* **Documents → Doctors**: CreatedBy references DoctorID in Doctors.
* **Documents → Patients**: CreatedFor references PatientID in Patients.
* **DocumentAccess → Documents**: DocumentID references DocumentID in Documents.
* **DocumentAccess → Users**: UserID references UserID in Users.
* **InvoiceDetails → Documents**: DocumentID references DocumentID in Documents.
* **ReportDetails → Documents**: DocumentID references DocumentID in Documents.
* **PrescriptionDetails → Documents**: DocumentID references DocumentID in Documents.
* **PrescriptionDetails → PharmacyInventory**: ItemID references ItemID in PharmacyInventory.

**3. Unique Constraints**

* **Users**: Email is unique to ensure no duplicate user accounts.
* **Rooms**: RoomNumber is unique to prevent duplicate room entries.

**4. Default Constraints**

* **Users**: RegistrationDate defaults to GETDATE() to record the account creation date.
* **Doctors**: JoinDate defaults to GETDATE() for the date they joined the system.
* **Patients**: RegistrationDate defaults to GETDATE() for the date of registration.
* **Appointments**: AppointmentDate and Status have default constraints where applicable.
* **Billing**: IssueDate defaults to GETDATE().
* **Documents**: CreationDate defaults to GETDATE().
* **Admissions**: AdmissionDate defaults to GETDATE().
* **LabTests**: TestDate defaults to GETDATE().

**5. Check Constraints**

* **Users**: UserType must be either 'Doctor' or 'Patient'.
* **Patients**: Gender must be 'Male', 'Female', or 'Other'.
* **Doctors**: AvailabilityStatus must be 'Available' or 'Unavailable'.
* **Appointments**: Status must be 'Scheduled', 'Completed', or 'Cancelled'.
* **Rooms**: RoomType must be 'General', 'ICU', or 'Private'. AvailabilityStatus must be 'Available' or 'Occupied'.
* **Billing**: PaymentStatus must be 'Paid' or 'Unpaid'.
* **Documents**: DocumentType must be 'Invoice', 'Prescription', or 'Report'.
* **DocumentAccess**: AccessLevel must be 'Read' or 'Write'.

**6. Cascading Actions**

* ON DELETE CASCADE for relationships involving patient records, such as Appointments, Prescriptions, and Documents, ensures that related records are deleted when a patient is removed.
* ON DELETE SET NULL for relationships like Admissions ensures that fields like ResponsibleDoctorID are set to NULL instead of breaking the foreign key constraints.

These constraints collectively ensure robust data consistency and security across the database.

**2.5 Diagrams**:

metin, ekran görüntüsü, sayı, numara, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu

Diagram with cardinalities

diyagram, metin, plan, şematik içeren bir resim

Açıklama otomatik olarak oluşturuldu

SSMS Diagram

**2.6 PROCEDURE and FUNCTIONS**:

**Stored Procedures**

1. **GetAllAppointments**
   * **Purpose**: Retrieves all records from the Appointments table.
   * **Syntax**:

CREATE PROCEDURE GetAllAppointments

AS

BEGIN

SELECT \* FROM Appointments;

END;

* + **Use Case**: Used in GUI components to display all appointments in a table format for admins, doctors, or patients.

1. **AddAppointment**
   * **Purpose**: Inserts a new appointment record into the database.
   * **Parameters**:
     + @PatientID: The ID of the patient.
     + @DoctorID: The ID of the doctor.
     + @AppointmentDate: The date and time of the appointment.
     + @Status: Current status of the appointment (e.g., scheduled, completed).
     + @Notes: Additional notes for the appointment.
   * **Syntax**:

CREATE PROCEDURE AddAppointment

@PatientID INT,

@DoctorID INT,

@AppointmentDate DATETIME,

@Status NVARCHAR(50),

@Notes NVARCHAR(255)

AS

BEGIN

INSERT INTO Appointments (PatientID, DoctorID, AppointmentDate, Status, Notes)

VALUES (@PatientID, @DoctorID, @AppointmentDate, @Status, @Notes);

END;

* + **Use Case**: Called when adding a new appointment in the admin or doctor interface.

1. **UpdateAppointment**
   * **Purpose**: Updates an existing appointment record in the database.
   * **Parameters**:
     + @AppointmentID: The ID of the appointment to be updated.
     + Other parameters are the same as in AddAppointment.
   * **Syntax**:

CREATE PROCEDURE UpdateAppointment

@AppointmentID INT,

@PatientID INT,

@DoctorID INT,

@AppointmentDate DATETIME,

@Status NVARCHAR(50),

@Notes NVARCHAR(255)

AS

BEGIN

UPDATE Appointments

SET PatientID = @PatientID,

DoctorID = @DoctorID,

AppointmentDate = @AppointmentDate,

Status = @Status,

Notes = @Notes

WHERE AppointmentID = @AppointmentID;

END;

* + **Use Case**: Called when editing an appointment in the admin or doctor interface.

1. **DeleteAppointment**
   * **Purpose**: Deletes a specific appointment from the database.
   * **Parameters**:
     + @AppointmentID: The ID of the appointment to be deleted.
   * **Syntax**:

CREATE PROCEDURE DeleteAppointment

@AppointmentID INT

AS

BEGIN

DELETE FROM Appointments

WHERE AppointmentID = @AppointmentID;

END;

* + **Use Case**: Used to delete appointments directly via the admin interface.

1. **SearchAppointments**
   * **Purpose**: Searches for appointments based on a keyword in their Status or Notes fields.
   * **Parameters**:
     + @Keyword: A keyword to search for.
   * **Syntax**:

CREATE PROCEDURE SearchAppointments

@Keyword NVARCHAR(255)

AS

BEGIN

SELECT \* FROM Appointments

WHERE Status LIKE @Keyword OR Notes LIKE @Keyword;

END;

* + **Use Case**: Implements search functionality for the admin and doctor panels.

**User-Defined Functions**

1. **IsRoomAvailable**
   * **Purpose**: Checks if a specific room is available based on its AvailabilityStatus.
   * **Parameters**:
     + @RoomID: The ID of the room to check.
   * **Return Type**: BIT (1 if available, 0 if not).
   * **Syntax**:

CREATE FUNCTION IsRoomAvailable (@RoomID INT)

RETURNS BIT

AS

BEGIN

DECLARE @IsAvailable BIT;

SELECT @IsAvailable =

CASE

WHEN AvailabilityStatus = 'Available' THEN 1

ELSE 0

END

FROM Rooms

WHERE RoomID = @RoomID;

RETURN @IsAvailable;

END;

* + **Use Case**: Used in the admin or doctor interface when assigning a room to a patient.

**2.7 BUISNESS RULES**:

**1. User Management Rules**

1. **Unique Email Constraint**:
   * Each user must have a unique email address for authentication purposes.
   * Ensured by a UNIQUE constraint on the Email column in the Users table.
2. **User Type Validation**:
   * A user must be either a "Doctor" or "Patient" as specified in the UserType column.
   * Enforced using a CHECK constraint.
3. **User Deletion Impact**:
   * If a user is deleted, related data (e.g., appointments, prescriptions, documents) should also be handled gracefully.
   * Cascading rules:
     + **ON DELETE CASCADE** for Patients and Doctors.
     + **ON DELETE SET NULL** for non-critical references like ResponsibleDoctorID in the Admissions table.

**2. Patient Management Rules**

1. **Patient-Specific Details**:
   * All patients must have a DateOfBirth, Gender, and ContactNumber specified during registration.
   * The Gender must be one of "Male," "Female," or "Other," enforced by a CHECK constraint.
2. **Patient-User Relationship**:
   * Each patient must be associated with exactly one user account via the UserID column.
3. **Patient Data Consistency**:
   * If a patient record is deleted, all associated appointments, prescriptions, admissions, and medical records must also be deleted (ON DELETE CASCADE).

**3. Doctor Management Rules**

1. **Doctor-Specific Details**:
   * Every doctor must have a Specialization and AvailabilityStatus defined.
   * The AvailabilityStatus can only be "Available" or "Unavailable," enforced by a CHECK constraint.
2. **Doctor-User Relationship**:
   * Each doctor must be associated with exactly one user account via the UserID column.
3. **Doctor Data Consistency**:
   * Deleting a doctor should cascade to all related appointments, prescriptions, admissions, and medical records (ON DELETE CASCADE).

**4. Appointment Management Rules**

1. **Appointment Validation**:
   * An appointment must be associated with both a valid patient (PatientID) and a doctor (DoctorID).
   * FOREIGN KEY constraints ensure this relationship.
2. **Appointment Status**:
   * The status of an appointment must be one of "Scheduled," "Completed," or "Cancelled," enforced by a CHECK constraint.
3. **Billing Dependency**:
   * Each appointment can have a corresponding billing record.
   * Deleting an appointment will cascade to its associated billing record (ON DELETE CASCADE).
4. **Searchable Notes and Status**:
   * Appointments can be searched by keywords in their Notes or Status.

**5. Billing Rules**

1. **Billing Status**:
   * Bills must have a PaymentStatus of either "Paid" or "Unpaid."
   * Enforced by a CHECK constraint.
2. **Dependency on Appointments**:
   * Every bill must be associated with an appointment (AppointmentID).
3. **Invoice Management**:
   * Bills can be linked to invoice details stored in the InvoiceDetails table.

**6. Room Management Rules**

1. **Unique Room Numbers**:
   * Each room must have a unique RoomNumber in the Rooms table.
2. **Room Type and Status**:
   * The RoomType must be one of "General," "ICU," or "Private."
   * The AvailabilityStatus must be either "Available" or "Occupied."
   * Enforced by CHECK constraints.
3. **Room Availability Check**:
   * The function IsRoomAvailable checks if a room is available for admission.
4. **Room Admissions**:
   * A room can host multiple admissions over time but cannot have overlapping admissions for the same time period.

**7. Prescription and Pharmacy Management Rules**

1. **Prescription-Patient Relationship**:
   * Each prescription must be linked to a valid patient (PatientID) and doctor (DoctorID).
2. **Medication Availability**:
   * Medications included in a prescription must exist in the PharmacyInventory table.
3. **Prescription Item Validation**:
   * Quantities prescribed in the PrescriptionItems table must not exceed the available stock in the PharmacyInventory.
4. **Medication Expiry Check**:
   * Medications past their ExpiryDate cannot be prescribed.

**8. Admissions and Staff Rules**

1. **Admission Validation**:
   * Each admission must specify a patient, a room, a responsible doctor, and a responsible nurse.
   * FOREIGN KEY constraints ensure relationships with the Patients, Rooms, Doctors, and Staff tables.
2. **Discharge Management**:
   * Discharge details, including DischargeDate and billing information, must be recorded during the discharge process.
3. **Room Assignment**:
   * A room must be marked as "Occupied" upon admission and "Available" upon discharge.
4. **Staff Roles**:
   * Staff can have multiple roles (e.g., Nurse, Lab Technician) but must have a defined Role.

**9. Document Management Rules**

1. **Document Validation**:
   * Each document must have a DocumentType, Status, and valid references (CreatedBy and CreatedFor).
2. **Access Control**:
   * Access to documents is managed through the DocumentAccess table.
   * Users can have "Read" or "Write" access, specified in the AccessLevel column.
3. **Document Types**:
   * Valid document types include "Invoice," "Prescription," and "Report."
4. **Cascading Deletion**:
   * Deleting a document cascades to related records in InvoiceDetails, ReportDetails, and PrescriptionDetails.

**10. Lab Test Management Rules**

1. **Lab Test Validation**:
   * Each lab test must be associated with a patient and optionally a lab technician.
2. **Technician Assignment**:
   * Technicians can be assigned to lab tests either manually or randomly.
   * The LabTestAssignments table tracks these assignments.
3. **Test Results**:
   * Test results must be recorded in the TestResult column of the LabTests table.

**11. Medical Records Rules**

1. **Record Validation**:
   * Each medical record must specify a patient and doctor.
   * FOREIGN KEY constraints ensure the relationships.
2. **Deletion Impact**:
   * Deleting a patient or doctor cascades to all their associated medical records.

**12. General Rules**

1. **Data Integrity**:
   * NOT NULL constraints ensure that critical columns like FirstName, LastName, Email, and Password are always populated.
2. **Audit Trails**:
   * Timestamps like RegistrationDate, JoinDate, AdmissionDate, and CreationDate track when records were created.
3. **Search and Filtering**:
   * Tables like Appointments, Patients, and Prescriptions include searchable fields (Notes, Status) for easy retrieval.
4. **Default Values**:
   * Default constraints ensure fields like RegistrationDate and JoinDate are automatically populated with the current date.
5. JAVA CLASSES

metin, ekran görüntüsü, sayı, numara, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu metin, ekran görüntüsü, doküman, belge, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**3.1 Overview**

1. auth – Handles user authentication and security.
2. database – Manages database interactions.
3. backend – Implements the core business logic and operational workflows.
4. forms – Contains Graphical User Interface (GUI) components for user interaction.
5. models – Defines the application's data structures and entities.
6. utils – Provides utility functions and common methods.
7. AdminPanelDao – Provides functionality for Admin Panel
8. AdminModels

**3.2 Detailed Package and Class Descriptions**

**1. hbys.auth Package**

The hbys.auth package handles user authentication and password management. It includes two key classes:

**1. LoginValidator**

* Purpose: Validates user credentials during login.
* Key Functionality:
  + Hashes the input password using PasswordHasher.
  + Verifies the email and hashed password against the database.
  + Returns true if the credentials are valid.

**2. PasswordUpdater**

* Purpose: Manages secure password updates.
* Key Functionality:
  + Checks the current password by hashing and matching it with the stored value.
  + Updates the password with the new hashed value if valid.

Both classes emphasize secure database operations through hashing and prepared statements.

**2. hbys.database Package**

The hbys.database package is responsible for establishing and testing connections to the database. It contains the following classes:

**1. DatabaseConnection**

* **Purpose:** Provides a reusable method to establish a connection to the database.
* **Key Functionality:**
  + Stores database connection details (URL, username, password).
  + Provides a static method, getConnection(), that creates and returns a connection.
  + Handles connection errors by printing error details and returning null.

**2. TestDBConnection**

* **Purpose:** Tests the functionality of the database connection and retrieves sample data.
* **Key Functionality:**
  + Connects to the database using provided credentials.
  + Executes a query to retrieve the top 10 records from the Patients table.
  + Prints the retrieved data to verify the connection and query functionality.

**3. TestDatabaseConnection**

* **Purpose:** Tests the database connection setup and validates the JDBC driver.
* **Key Functionality:**
  + Explicitly loads the SQL Server JDBC driver using Class.forName.
  + Attempts to establish a connection to the database.
  + Provides detailed feedback on the connection status (success or error).
  + Ensures proper cleanup by closing the connection in a finally block.

**3.hbys.gui.backend Package**

The hbys.gui.backend package is responsible for managing GUI interactions related to backend operations, such as adding admissions, deleting appointments, and deleting prescriptions. The package includes the following classes:

**1. AddAdmissionPage**

* **Purpose:** Handles the GUI logic for adding a new admission to the hospital management system.
* **Key Functionality:**
  + **Doctor-Specific Initialization:**
    - Accepts the responsible doctor's ID during initialization.
  + **Room and Nurse Loading:**
    - Retrieves and populates available rooms and nurses from the database.
    - Ensures that only available rooms and active nurses are listed.
  + **Admission Management:**
    - Gathers admission details, including patient ID, room, nurse, and notes.
    - Inserts the new admission into the database with the current date as the admission date.
    - Provides feedback on the success or failure of the operation.
  + **Error Handling:**
    - Handles database connection issues and query errors gracefully with user-friendly error messages.

**2. DeleteAppointmentPage**

* **Purpose:** Manages the deletion of an appointment from the hospital management system.
* **Key Functionality:**
  + **Appointment Initialization:**
    - Accepts the appointment ID during initialization and confirms the action with the user.
  + **Appointment Deletion:**
    - Deletes the specified appointment from the database using its ID.
    - Provides feedback on the success or failure of the deletion.
  + **Error Handling:**
    - Handles database connection issues and query errors gracefully.
  + **User Confirmation:**
    - Displays a confirmation dialog to ensure that users intend to delete the appointment.

**3. DeletePrescriptionPage**

* **Purpose:** Manages the deletion of a prescription from the hospital management system.
* **Key Functionality:**
  + **Prescription Initialization:**
    - Accepts the prescription ID during initialization and confirms the action with the user.
  + **Prescription Deletion:**
    - Deletes the specified prescription from the database using its ID.
    - Provides feedback on the success or failure of the deletion.
  + **Error Handling:**
    - Handles database connection issues and query errors gracefully.
  + **User Confirmation:**
    - Displays a confirmation dialog to ensure that users intend to delete the prescription.

**4. DischargePatientPage**

* Purpose: Handles the logic for discharging a patient and managing the associated administrative tasks, including billing and invoice creation.

Key Features:

1. Discharge Management:
   * Allows setting the discharge date and notes for the patient.
   * Updates the Admissions table with the discharge date and additional notes.
2. Billing Calculation:
   * Calculates the total amount for the patient's room stay based on the admission and discharge dates.
   * Inserts billing information into the Billing table with Unpaid status.
3. Invoice Creation:
   * Generates an invoice document in the Documents table.
   * Creates detailed invoice records in the InvoiceDetails table.
4. Transaction Management:
   * Ensures atomic operations using transaction handling (conn.setAutoCommit(false)).
   * Rolls back changes in case of errors.

GUI Components:

* Discharge Date Picker: To select the discharge date.
* Notes Area: To enter additional notes for the discharge.
* Save Button: Triggers the discharge logic.
* Cancel Button: Closes the form without saving changes.

**5. ManageAppointmentsPage Class**

* Purpose: Provides a GUI interface for patients to manage their appointments.

Key Features:

1. Appointment Management:
   * Displays all appointments for the logged-in patient in a table format.
   * Allows cancellation of appointments.
2. Data Loading:
   * Fetches appointment details such as ID, date, status, and notes from the Appointments table.
   * Dynamically populates the table with the retrieved data.
3. Appointment Cancellation:
   * Cancels the selected appointment and updates its status to Cancelled in the Appointments table.
   * Prompts for confirmation before cancellation.

GUI Components:

* Appointments Table: Displays appointment details in columns (Appointment ID, Appointment Date, Status, Notes).
* Cancel Appointment Button: Cancels the selected appointment.
* Back Button: Returns to the previous screen.

**6. RequestLabTestPage**

• **Purpose**: Allows doctors to request lab tests for a specific patient and assigns the test to a random lab technician.  
• **Key Functionality**:  
o **Patient Details**:  
 Displays patient name and email.  
o **Random Technician Assignment**:  
 Selects a random lab technician from the database for the lab test.  
o **Test Request**:  
 Inserts lab test details into the LabTests table.  
o **Error Handling**:  
 Manages database connection issues and input errors.

**7. RequestNewLabTestPage**

• **Purpose**: Allows doctors to manually select a patient and lab technician to request a new lab test.  
• **Key Functionality**:  
o **Patient and Technician Loading**:  
 Populates dropdowns with available patients and lab technicians.  
o **Lab Test Request**:  
 Saves lab test details, including patient ID, technician ID, test name, and notes, into the LabTests table.  
o **Error Handling**:  
 Ensures proper data validation and database connection checks.

**8. SetAppointmentPage**

• **Purpose**: Enables doctors to set up new appointments for patients.  
• **Key Functionality**:  
o **Patient Details**:  
 Displays patient information, including contact details and address.  
o **Appointment Creation**:  
 Inserts appointment details (date, status, and notes) into the Appointments table.  
 Optionally associates an appointment fee, saved in the Billing table.  
o **Error Handling**:  
 Handles invalid inputs and database connection issues.

**9. UpdateAdmissionPage**   
• **Purpose**: Allows updating of existing patient admission details, including room, nurse, and notes.  
• **Key Functionality**:  
o **Load Admission Details**:  
 Fetches current admission details such as room ID, nurse ID, and notes.  
 Pre-selects the current room and nurse in the dropdowns.  
o **Update Admission**:  
 Updates the admission record in the database with new room, nurse, and notes.  
o **Error Handling**:  
 Handles database connection issues and query errors gracefully.  
• **GUI Components**:  
o Patient ID Field: Displays the patient ID associated with the admission.  
o Room Dropdown: Allows selecting a room.  
o Nurse Dropdown: Allows selecting a nurse.  
o Notes Area: Enables updating notes related to the admission.  
o Update Button: Saves changes to the database.  
o Cancel Button: Closes the form without saving changes.

**10.UpdateAppointmentPage**   
• **Purpose**: Enables updating the status, notes, and consultation fee of an appointment.  
• **Key Functionality**:  
o **Appointment Update**:  
 Updates the appointment record in the database, including status and notes.  
 Adds or updates billing records if the appointment is marked as "Completed".  
o **Error Handling**:  
 Ensures proper transaction management and input validation for the consultation fee.  
• **GUI Components**:  
o Status Dropdown: Allows changing the appointment status.  
o Notes Area: Enables updating notes related to the appointment.  
o Consultation Fee Field: Allows entering or updating the consultation fee.  
o Update Button: Saves changes to the database.  
o Cancel Button: Closes the form without saving changes.

**11. UpdateLabTestPage**   
• **Purpose**: Enables updating of lab test details, including test name, result, and assigned technician.  
• **Key Functionality**:  
o **Load Lab Test Details**:  
 Fetches current lab test details such as name, result, and assigned technician.  
o **Technician Assignment**:  
 Allows updating the technician assigned to the lab test.  
o **Lab Test Update**:  
 Saves updated details, including test name, result, and technician, to the database.  
o **Error Handling**:  
 Handles database connection issues and invalid data inputs gracefully.  
• **GUI Components**:  
 Patient Name Label: Displays the name of the patient associated with the test.  
o Test Name Field: Allows updating the test name.  
o Test Result Area: Enables updating the test result.  
o Technician Dropdown: Allows assigning a different technician.  
o Save Button: Saves changes to the database.  
o Cancel Button: Closes the form without saving changes.

**12.UpdatePrescriptionPage**

• **Purpose**: Allows updating the notes associated with an existing prescription.  
• **Key Functionality**:

1. **Load Prescription Details**:
   * Fetches the current notes of the prescription from the database.
   * Displays the notes in a text area for editing.
2. **Update Prescription**:
   * Saves the updated notes back to the database.
3. **Error Handling**:
   * Handles database connection errors and provides user-friendly messages.  
     • **GUI Components**:

* **Notes Area**: Displays and allows editing of the prescription notes.
* **Update Button**: Saves the updated notes to the database.
* **Cancel Button**: Closes the form without saving changes.

**13. WritePrescriptionPage**

• **Purpose**: Enables doctors to create new prescriptions for patients and link them to pharmacy inventory items.  
• **Key Functionality**:

1. **Load Pharmacy Inventory**:
   * Retrieves the list of available medicines and their details (e.g., quantity, price) from the database.
   * Displays the medicines in a table for selection.
2. **Create Prescription**:
   * Allows doctors to select a medicine, specify the quantity, and add notes.
   * Saves the prescription and its details in the Prescriptions, PrescriptionItems, and Documents tables.
3. **Transaction Management**:
   * Ensures atomic operations for prescription and document creation.
   * Rolls back changes in case of errors.
4. **Error Handling**:
   * Handles invalid quantity inputs, database connection issues, and query errors.  
     • **GUI Components**:

* **Pharmacy Inventory Table**: Displays available medicines with details.
* **Quantity Field**: Allows specifying the quantity of the selected medicine.
* **Notes Area**: Enables adding notes for the prescription.
* **Save Prescription Button**: Creates and saves the prescription.
* **Cancel Button**: Closes the form without saving changes.

**4.hbys.models Package**

**Document Class**

• Purpose: Represents a document in the hospital management system, such as prescriptions or invoices.  
• Key Attributes:

* documentID: Unique identifier for the document.
* documentName: Name of the document.
* documentType: Type of the document (e.g., prescription, invoice).
* status: Current status of the document (e.g., completed, pending).  
  • Key Methods:
* Getters and Setters: For accessing and modifying document properties.
* toString: Returns a formatted string representation of the document, combining its name and status.

**Patient Class**

• Purpose: Represents a patient in the hospital management system.  
• Key Attributes:

* patientID: Unique identifier for the patient.
* firstName: Patient's first name.
* lastName: Patient's last name.
* gender: Gender of the patient.
* contactNumber: Contact number of the patient.
* address: Address of the patient.  
  • Key Methods:
* Getters and Setters: For accessing and modifying patient details.
* toString: Returns a formatted string representation of the patient, combining their full name and gender.

**5.hbys.utils Package**

**PasswordHasher Class**

**Purpose**

The PasswordHasher class ensures the secure handling of user passwords by providing a method to hash them. It is critical for maintaining security within the application and safeguarding sensitive user credentials.

Key Functionality

1. Password Hashing:
   * Utilizes the SHA-256 algorithm to hash user passwords.
   * Ensures that plaintext passwords are not stored in the database, adhering to security best practices.
2. String Encoding:
   * Converts hashed bytes into a hexadecimal string format for easy storage and comparison.

Methods

1. hashPassword(String password):
   * Input: A plaintext password.
   * Output: A hashed representation of the password as a hexadecimal string.
   * Steps:
     + Uses the MessageDigest class to compute the hash using the SHA-256 algorithm.
     + Converts the resulting byte array into a hexadecimal string using a StringBuilder and String.format.

**6.hbys.AdminPanelDao Package**

**1. AdmissionDAO**

Purpose:

Manages database interactions related to hospital admission processes.

Accessed Table:

Admissions: Stores patient admission details.

Capabilities:

List all admissions.

Add a new admission.

Update an existing admission.

Delete an admission.

Search admissions by keyword.

**2. AppointmentDAO**

Purpose:

Handles database interactions for appointment processes.

Accessed Table:

Appointments: Stores patient appointment details.

Capabilities:

List all appointments.

Add a new appointment.

Update an existing appointment.

Delete an appointment.

Search appointments by keyword.

**3. BillingDAO**

Purpose:

Manages billing processes.

Accessed Table:

Billing: Stores billing details.

Capabilities:

List all billing information.

Add a new bill.

Update an existing bill.

Delete a bill.

Search bills by keyword.

**4. DoctorDAO**

Purpose:

Manages the database operations for doctors.

Accessed Table:

Doctors: Stores doctor details.

Capabilities:

List all doctors.

Add a new doctor.

Update an existing doctor.

Delete a doctor.

Search doctors by keyword.

**5. DocumentAccessDAO**

Purpose:

Manages access rights for documents.

Accessed Table:

DocumentAccess: Stores document access information.

Capabilities:

List all access records.

Add a new access record.

Update an access record.

Delete an access record.

Search access records by keyword.

**6. DocumentDAO**

Purpose:

Handles operations related to documents.

Accessed Table:

Documents: Stores documents in the system.

Capabilities:

List all documents.

Add a new document.

Update an existing document.

Delete a document.

Search documents by keyword.

**7. InvoiceDAO**

Purpose:

Manages invoice details.

Accessed Table:

InvoiceDetails: Stores invoice details.

Capabilities:

List all invoice details.

Add a new invoice detail.

Update an existing invoice detail.

Delete an invoice detail.

Search invoice details by keyword.

**8. LabTechnicianDAO**

Purpose:

Manages operations related to lab technicians.

Accessed Table:

LabTechnicians: Stores lab technician details.

Capabilities:

List all lab technicians.

Add a new lab technician.

Update an existing lab technician.

Delete a lab technician.

Search lab technicians by keyword.

**9. LabTestDAO**

Purpose:

Manages laboratory tests.

Accessed Table:

LabTests: Stores laboratory test details.

Capabilities:

List all lab tests.

Add a new lab test.

Update an existing lab test.

Delete a lab test.

Search lab tests by keyword.

**10. PatientDAO**

Purpose:

Manages patient information.

Accessed Table:

Patients: Stores patient details.

Capabilities:

List all patients.

Add a new patient.

Update an existing patient.

Delete a patient.

Search patients by keyword.

**11. PharmacyInventoryDAO**

Purpose:

Manages pharmacy inventory.

Accessed Table:

PharmacyInventory: Stores medication and inventory details.

Capabilities:

List all inventory items.

Add a new item.

Update an existing item.

Delete an item.

Search items by keyword.

**12. PrescriptionDAO**

Purpose:

Manages prescription information.

Accessed Table:

Prescriptions: Stores prescriptions.

Capabilities:

List all prescriptions.

Add a new prescription.

Update an existing prescription.

Delete a prescription.

Search prescriptions by keyword.

**13. RoomDAO**

Purpose:

Manages room information.

Accessed Table:

Rooms: Stores room details.

Capabilities:

List all rooms.

Add a new room.

Update an existing room.

Delete a room.

Search rooms by keyword.

Check room availability.

**14. StaffDAO**

Purpose:

Manages staff information.

Accessed Table:

Staff: Stores staff details.

Capabilities:

List all staff members.

Add a new staff member.

Update an existing staff member.

Delete a staff member.

Search staff members by keyword.

**15. UserDAO**

Purpose:

Manages user accounts.

Accessed Table:

Users: Stores user account information.

Capabilities:

List all users.

Add a new user.

Update an existing user.

Delete a user.

Search users by keyword.

**16. LabTestAssignmentDAO**

Purpose:

Manages the assignment of lab tests to technicians.

Accessed Table:

LabTestAssignments: Stores lab test assignment details.

Capabilities:

List all assignments.

Add a new assignment.

Update an existing assignment.

Delete an assignment.

Search assignments by Test ID or Technician ID.

**17. MedicalRecordDAO**

Purpose:

Manages medical records of patients.

Accessed Table:

MedicalRecords: Stores medical record details.

Capabilities:

List all medical records.

Add a new medical record.

Update an existing medical record.

Delete a medical record.

Search records by Patient ID, Doctor ID, or description.

**18. PrescriptionDetailsDAO**

Purpose:

Manages detailed information about prescriptions.

Accessed Table:

PrescriptionDetails: Stores prescription details.

Capabilities:

List all prescription details.

Add a new prescription detail.

Update an existing prescription detail.

Delete a prescription detail.

Search prescription details by Document ID.

**19. PrescriptionItemsDAO**

Purpose:

Manages the quantity and details of medication in prescriptions.

Accessed Table:

PrescriptionItems: Stores prescription items.

Capabilities:

List all prescription items.

Add a new prescription item.

Update an existing prescription item.

Delete a prescription item.

Search prescription items by Prescription ID.

**20. ReportDetailDAO**

Purpose:

Manages report details.

Accessed Table:

ReportDetails: Stores report information.

Capabilities:

List all reports.

Add a new report.

Update an existing report.

Delete a report.

Search reports by title or content.

**7.hbys.AdminModels Package**

**1. Admission**

Purpose: Manages hospital admission details.

Fields:

admissionID: Admission ID

patientID: Patient ID

roomID: Room ID

admissionDate: Admission date

dischargeDate: Discharge date

notes: Notes

responsibleDoctorID: Responsible doctor ID

responsibleNurseID: Responsible nurse ID

**2. Appointment**

Purpose: Manages appointment details.

Fields:

appointmentID: Appointment ID

patientID: Patient ID

doctorID: Doctor ID

appointmentDate: Appointment date

status: Appointment status

notes: Notes

**3. Billing**

Purpose: Manages billing operations.

Fields:

billID: Bill ID

appointmentID: Appointment ID

totalAmount: Total amount

paymentStatus: Payment status

issueDate: Issue date

**4. Doctor**

Purpose: Manages doctor details.

Fields:

doctorID: Doctor ID

userID: User ID

specialization: Specialization

contactNumber: Contact number

availabilityStatus: Availability status

joinDate: Join date

**5. Document**

Purpose: Manages document details.

Fields:

documentID: Document ID

documentType: Document type

relatedID: Related ID

createdBy: Created by user ID

createdFor: Created for target ID

creationDate: Creation date

status: Status

description: Description

**6. DocumentAccess**

Purpose: Manages document access details.

Fields:

accessID: Access ID

documentID: Document ID

userID: User ID

accessLevel: Access level

accessDate: Access date

**7. InvoiceDetail**

Purpose: Manages invoice details.

Fields:

invoiceDetailID: Invoice detail ID

documentID: Document ID

itemDescription: Item description

quantity: Quantity

unitPrice: Unit price

totalPrice: Total price

* 1. **LabTechnician**

Purpose: Manages lab technician details.

Fields:

technicianID: Technician ID

firstName: First name

lastName: Last name

contactNumber: Contact number

email: Email

hireDate: Hire date

**9. LabTest**

Purpose: Manages laboratory test details.

Fields:

testID: Test ID

patientID: Patient ID

technicianID: Technician ID

testName: Test name

testDate: Test date

testResult: Test result

**10. LabTestAssignment**

Purpose: Manages lab test assignment details.

Fields:

assignmentID: Assignment ID

testID: Test ID

technicianID: Technician ID

assignmentDate: Assignment date

**11. Patient1**

Purpose: Manages patient details.

Fields:

patientID: Patient ID

userID: User ID

dateOfBirth: Date of birth

gender: Gender

contactNumber: Contact number

address: Address

registrationDate: Registration date

**12. PharmacyItem**

Purpose: Manages pharmacy item details.

Fields:

itemID: Item ID

itemName: Item name

quantity: Quantity

pricePerUnit: Price per unit

expiryDate: Expiry date

supplierName: Supplier name

**13. Prescription**

Purpose: Manages prescription details.

Fields:

prescriptionID: Prescription ID

patientID: Patient ID

doctorID: Doctor ID

dateIssued: Issued date

notes: Notes

**14. PrescriptionDetail**

Purpose: Manages prescription details.

Fields:

prescriptionDetailID: Prescription detail ID

documentID: Document ID

itemID: Item ID

quantity: Quantity

**15. PrescriptionItem**

Purpose: Manages prescription items in detail.

Fields:

prescriptionItemID: Prescription item ID

prescriptionID: Prescription ID

itemID: Item ID

quantity: Quantity

**16. MedicalRecord**

Purpose: Manages medical records.

Fields:

recordID: Record ID

patientID: Patient ID

doctorID: Doctor ID

recordDate: Record date

description: Description

**17. ReportDetail**

Purpose: Manages report details.

Fields:

reportID: Report ID

documentID: Document ID

reportTitle: Report title

reportContent: Report content

**18. Room**

Purpose: Manages room details.

Fields:

roomID: Room ID

roomType: Room type

roomNumber: Room number

availabilityStatus: Availability status

**19. Staff**

Purpose: Manages staff details.

Fields:

staffID: Staff ID

firstName: First name

lastName: Last name

role: Role

contactNumber: Contact number

email: Email

joinDate: Join date

**20. User**

Purpose: Manages user account details.

Fields:

userID: User ID

firstName: First name

lastName: Last name

email: Email

password: Password

userType: User type

registrationDate: Registration date

**4.Graphical User Interface (GUI)**

**4.1 Overview**

* The GUI of the Hospital Management and Document Tracking System is designed to be intuitive and responsive.
* The interface is implemented using Java Swing, offering separate views and workflows for doctors, patients, and administrators.
* Each module is represented by interactive tables, buttons, and forms for seamless navigation and operation.

**4.2 GUI Features**

**4.2.1 Login Page:**

**metin, ekran görüntüsü, ekran, görüntüleme, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

The **Login Page** is the entry point for the Hospital Automation System, allowing both doctors and patients to securely access their respective dashboards.

**Features:**

1. **User-Friendly Layout:**
   * Two labeled input fields:
     + **User** for email or username.
     + **Password**, which is masked for security.
   * A password visibility toggle button for convenience.
2. **Secure Authentication:**
   * Validates credentials through the database using prepared SQL statements.
   * Redirects users to either the **Doctor** or **Patient** dashboard based on their role.
3. **Error Handling:**
   * Displays informative messages for invalid credentials.
   * Ensures proper database connectivity before proceeding.
4. **Password Toggle:**
   * The eye icon (👁️) allows users to view or hide their entered password.

**GUI Components:**

* **Input Fields:**
  + jTextField\_Username for entering the username or email.
  + jPasswordField for entering the password.
* **Buttons:**
  + jButton\_Login for submitting credentials.
  + jToggleButton\_ShowPassword for toggling password visibility.

**Key Functionality in Code:**

1. **Authentication Logic:**
   * Retrieves user credentials from the database via the authenticateUser method.
   * Checks user type (Doctor or Patient) and redirects accordingly.
2. **Password Visibility:**
   * Implements a toggle button to show/hide the password using jPasswordField.setEchoChar.
3. **Role-Based Navigation:**
   * If the user is a doctor, they are redirected to the **DoctorPage**.
   * If the user is a patient, they are redirected to the **PatientPage**.

**Challenges and Solutions:**

* **Challenge:** Handling invalid inputs gracefully.
  + **Solution:** Used JOptionPane for user-friendly error messages.

**Code Overview:**

* **authenticateUser(String email, String password):**
  + Connects to the database and verifies the user credentials.
  + Returns the user type (Doctor or Patient) or null for invalid credentials.
* **getPatientID(int userID):**
  + Fetches the patient ID linked to a user's account.
* **getDoctorID(int userID):**
  + Fetches the doctor ID linked to a user's account.

**4.2.2 Patient Page:**

**metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu**

The **Patient Page** serves as the primary interface for patients to manage their hospital-related activities, such as viewing appointments, prescriptions, lab results, and invoices.

**Features:**

1. **Customizable Dashboard:**
   * Displays a personalized welcome message (e.g., "Welcome, Ahmet Yılmaz!") and the patient's registered email address.
2. **Centralized Access to Information:**
   * Provides navigation options for:
     + **Appointments**: View and manage upcoming and past appointments.
     + **Lab Results**: Access and review test results.
     + **Prescriptions**: View issued prescriptions with details.
     + **Invoices/Bills**: Access payment and billing details.
     + **Profile**: View and update profile information.
3. **Interactive Table:**
   * A dynamic table (jTable) displays relevant information based on the selected navigation button.
   * The table updates in real-time to show data such as appointments, prescriptions, or lab results.
4. **Action Buttons:**
   * **Manage Appointments**: Redirects to a separate page for modifying or canceling appointments.
   * **Bills/Invoices**: Displays the patient's billing details and payment status.
   * **Log Out**: Safely logs out the patient and redirects to the login page.

**GUI Components:**

* **Navigation Panel:**
  + Buttons such as jButton\_PatientAppointments, jButton\_PatientLabResults, and jButton\_PatientPrescriptions for accessing specific data categories.
* **Dynamic Table:**
  + Displays data like appointment status, doctor notes, prescription details, lab results, and billing information.
* **User Info Section:**
  + Labels (jLabel\_Welcome and jLabel\_Email) to show the patient's full name and email.
* **Log Out Button:**
  + A jButton\_LogOut to terminate the session and return to the login screen.

**Key Functionalities in Code:**

1. **Dynamic Data Loading:**
   * **Appointments:** Retrieves and displays details such as appointment ID, doctor name, date, and notes.
   * **Lab Results:** Shows lab test results, technician details, and test dates.
   * **Prescriptions:** Lists prescriptions with itemized medication costs and notes.
   * **Invoices/Bills:** Displays billing information, including payment status and linked documents.
2. **User Customization:**
   * The customizePage method fetches patient-specific data (e.g., full name, email) and updates the GUI dynamically.
3. **Event Handling:**
   * Navigation buttons trigger corresponding data loading methods (e.g., loadAppointments, loadPrescriptions) to update the table with relevant information.
4. **Error Handling:**
   * Ensures smooth user experience by:
     + Displaying friendly messages for database connection issues or data unavailability.
     + Validating patient ID before loading data.
5. **Seamless Integration with Backend:**
   * Interacts with the database using SQL queries to fetch data based on the logged-in patient's ID.

**Challenges and Solutions:**

1. **Challenge:** Handling large amounts of data in the table.
   * **Solution:** Used DefaultTableModel to dynamically update and manage table content.
2. **Challenge:** Ensuring secure logout.
   * **Solution:** The jButton\_LogOutActionPerformed method clears the session and redirects to the login page.
3. **Challenge:** Displaying personalized data.
   * **Solution:** The patient ID (patientID) and user ID (userID) are passed to the constructor to fetch and display patient-specific information.

**Code Overview:**

1. **Dynamic Table Loading:**
   * loadAppointments: Fetches and populates appointment details.
   * loadPrescriptions: Displays prescription details with cost breakdown.
   * loadLabResults: Shows lab test results with technician information.
   * jButton\_ShowInvoiceAndBillsActionPerformed: Fetches and displays billing and invoice details.
2. **Patient-Specific Customization:**
   * customizePage: Fetches and displays the patient’s full name and email.
3. **Navigation Buttons:**
   * Each button (e.g., jButton\_PatientAppointments, jButton\_PatientLabResults) is linked to a method that fetches and updates the table content dynamically.

**4.2.3 Patient Profile Page:**

**metin, ekran görüntüsü, ekran, görüntüleme, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

The **Patient Profile Page** is a personalized interface for patients to view their profile details and update their password securely.

**Features:**

1. **Profile Information Display:**
   * Displays essential information about the patient, including:
     + Full Name (e.g., "Name: Ahmet Yılmaz").
     + User ID (e.g., "User ID: 1").
     + Email Address (e.g., "E-Mail: ahmet.yilmaz@gmail.com").
2. **Secure Password Management:**
   * Allows patients to update their password with proper validation and error handling.
3. **Interactive GUI Components:**
   * A password input field with a toggle to show/hide the entered password for convenience.
   * Buttons for navigating back to the main menu or saving the updated password.
4. **Database Integration:**
   * Fetches user details from the database dynamically based on the logged-in user's ID.
   * Updates the password securely in the database after validation.

**GUI Components:**

1. **Profile Details Section:**
   * **Labels:**
     + jLabel\_Name: Displays the patient's full name.
     + jLabel\_ID: Shows the user's unique ID.
     + jLabel\_EMail: Displays the user's registered email.
2. **Password Section:**
   * **Input Field:**
     + jPasswordField\_Password: Allows the patient to enter a new password.
   * **Toggle Button:**
     + jToggleButton\_ShowPassword: Toggles the visibility of the entered password.
3. **Action Buttons:**
   * **Back Button (jButton\_BackToMenu):**
     + Returns the user to the main menu.
   * **Change Password Button (jButton\_ChangePassword):**
     + Updates the user's password in the database.

**Key Functionalities in Code:**

1. **Customizing the Profile:**
   * The customizeProfile method:
     + Retrieves the patient's name, user ID, and email from the Users table in the database.
     + Dynamically updates the GUI with this information.
2. **Password Update:**
   * The jButton\_ChangePasswordActionPerformed method:
     + Validates the new password to ensure it is not empty.
     + Updates the password in the Users table using a prepared SQL statement.
     + Displays success or error messages depending on the outcome.
3. **Password Visibility Toggle:**
   * The jToggleButton\_ShowPasswordActionPerformed method:
     + Toggles the password visibility by setting or clearing the echo character in the password field (jPasswordField\_Password).
4. **Navigation:**
   * The jButton\_BackToMenuActionPerformed method:
     + Closes the profile page and returns to the previous menu.

**Challenges and Solutions:**

1. **Challenge:** Ensuring secure password updates.
   * **Solution:**
     + Used prepared statements to prevent SQL injection.
     + Implemented basic validation to ensure the new password is not empty.
2. **Challenge:** Displaying dynamic user data.
   * **Solution:**
     + Fetched user-specific details from the database using the customizeProfile method.
3. **Challenge:** Providing a user-friendly password input experience.
   * **Solution:**
     + Added a toggle button to show or hide the entered password.

**Code Overview:**

1. **Profile Customization:**
   * The customizeProfile method:
     + Queries the Users table to fetch the patient's full name, email, and user ID.
     + Updates the respective GUI labels (jLabel\_Name, jLabel\_ID, jLabel\_EMail).
2. **Password Management:**
   * The jButton\_ChangePasswordActionPerformed method:
     + Retrieves the entered password.
     + Updates the password securely in the database using a prepared SQL statement.
3. **Navigation and Interaction:**
   * The jButton\_BackToMenuActionPerformed method:
     + Closes the profile page and navigates back to the main menu.
   * The jToggleButton\_ShowPasswordActionPerformed method:
     + Toggles the visibility of the entered password.

**4.2.4 Doctor Page**

The **Doctor Page** serves as the central dashboard for doctors, providing a comprehensive interface to manage patients, appointments, prescriptions, admissions, and lab tests. It facilitates seamless interaction with the hospital automation system through various features and actions.

ekran görüntüsü, metin, yazılım, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, ekran, görüntüleme, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, ekran, görüntüleme, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturulduekran görüntüsü, metin, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturulduekran görüntüsü, metin, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Features:**

1. **Tabbed Interface for Modular Design:**
   * Organized into tabs: Patients, Appointments, Prescriptions, Admissions, and Lab Tests.
   * Each tab focuses on specific functionalities for streamlined navigation.
2. **Dynamic Data Loading:**
   * Real-time fetching and display of data (patients, appointments, prescriptions, etc.) from the database.
   * Data is presented in well-structured tables for clarity.
3. **Action-Oriented Buttons:**
   * Provides dedicated buttons for CRUD operations (Load, Update, Delete, etc.) and specific actions (Set Appointment, Write Prescription, etc.).
4. **Role-Based Customization:**
   * Tailored for the logged-in doctor, showing data specific to their assigned patients and tasks.
5. **Error and Exception Handling:**
   * Ensures robust error handling with detailed feedback for failed database connections or invalid user actions.

**GUI Components:**

1. **Tabs:**
   * **Patients Tab:**
     + Displays patient information (ID, Name, Email, Contact Number, Registration Date).
     + Buttons for loading patients, setting appointments, writing prescriptions, and requesting lab tests.
   * **Appointments Tab:**
     + Shows appointment details (ID, Patient Name, Date, Status, Notes).
     + Buttons for loading, updating, and deleting appointments.
   * **Prescriptions Tab:**
     + Displays prescription details (ID, Patient Name, Date Issued, Item Name, Quantity, Notes).
     + Buttons for loading, updating, and deleting prescriptions.
   * **Admissions Tab:**
     + Displays admission details (ID, Patient Name, Room Number, Room Type, Admission Date, etc.).
     + Buttons for loading, adding, updating admissions, and discharging patients.
   * **Lab Tests Tab:**
     + Displays lab test details (ID, Patient Name, Test Name, Date, Technician Name, Result).
     + Buttons for loading, requesting, updating, and deleting lab tests.
2. **Tables:**
   * jTable\_Patient, jTable\_Appointments, jTable\_Prescriptions, jTable\_Admissions, jTable\_LabTests for displaying data dynamically.
3. **Action Buttons:**
   * Example: jButton\_LoadPatientsq, jButton\_SetAppointment, jButton\_WritePrescription, jButton\_RequestLabTest, etc.

**Key Functionalities in Code:**

1. **Customizing the Page:**
   * The customizePage() method fetches and displays the doctor's name and email from the database using their ID.
2. **Data Loading:**
   * Each tab has a corresponding method to load data dynamically:
     + loadPatients()
     + loadAppointments()
     + loadPrescriptions()
     + loadAdmissions()
     + loadLabTests()
   * Uses SQL queries and PreparedStatement to fetch data securely and populate the respective table models.
3. **CRUD Operations:**
   * Buttons for updating, deleting, and adding new entries (appointments, prescriptions, admissions, etc.) invoke specific methods or navigate to other pages for further actions.
4. **Error Management:**
   * Uses JOptionPane to alert users about errors like database connection issues or invalid actions.
   * Handles SQLException for database-related errors with proper logging.

**Tab-Specific Functionalities:**

1. **Patients Tab:**
   * **Load Patients:** Fetches patients related to the doctor from the database using loadPatients().
   * **Set Appointment:** Opens SetAppointmentPage with the selected patient's ID.
   * **Write Prescription:** Opens WritePrescriptionPage with the selected patient's ID.
   * **Request Lab Test:** Opens RequestLabTestPage with the selected patient's ID.
2. **Appointments Tab:**
   * **Load Appointments:** Fetches all appointments for the doctor using loadAppointments().
   * **Update Appointment:** Opens UpdateAppointmentPage for the selected appointment.
   * **Delete Appointment:** Opens DeleteAppointmentPage for the selected appointment.
3. **Prescriptions Tab:**
   * **Load Prescriptions:** Fetches all prescriptions issued by the doctor using loadPrescriptions().
   * **Update Prescription:** Opens UpdatePrescriptionPage for the selected prescription.
   * **Delete Prescription:** Opens DeletePrescriptionPage for the selected prescription.
4. **Admissions Tab:**
   * **Load Admissions:** Fetches all admissions managed by the doctor using loadAdmissions().
   * **Add Admission:** Opens AddAdmissionPage for adding a new admission.
   * **Update Admission:** Opens UpdateAdmissionPage for the selected admission.
   * **Discharge Patient:** Opens DischargePatientPage for the selected admission.
5. **Lab Tests Tab:**
   * **Load Lab Tests:** Fetches all lab tests using loadLabTests().
   * **Request New Lab Test:** Opens RequestNewLabTestPage for a new test.
   * **Update Lab Test:** Opens UpdateLabTestPage for the selected lab test.
   * **Delete Lab Test:** Deletes the selected lab test from the database.

**Challenges and Solutions:**

1. **Challenge:** Dynamic data fetching and display for different doctors.
   * **Solution:** SQL queries are parameterized with the doctor's ID for role-based data filtering.
2. **Challenge:** Handling null or missing data.
   * **Solution:** Added checks for database connectivity and table row selection before performing actions.
3. **Challenge:** Managing multiple tabs and actions.
   * **Solution:** Organized functionalities into separate methods for modularity and maintainability.

**4.2.5 AdminPanel Page**

metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**1. Doctors Management**

* **Search Doctor:** Search for doctors using a keyword. The method populateDoctorTableWithSearch() dynamically updates the table.
* **Add Doctor:** Admins can add doctors by entering UserID, specialization, contact number, and availability status. The join date is set automatically.
* **Update Doctor:** Updates the selected doctor’s details, such as specialization and contact information.
* **Delete Doctor:** Removes the selected doctor from the database and refreshes the doctor table.

**2. Patients Management**

* **Search Patient:** Search patients using a keyword with populatePatientTableWithSearch().
* **Add Patient:** Admins input UserID, date of birth, gender, contact number, address, and registration date to add a new patient.
* **Update Patient:** Updates patient details like gender or contact number for the selected row in jTablePatients.
* **Delete Patient:** Deletes the selected patient record.

**3. Room Management**

* **Search Room:** Search rooms by keyword using populateRoomTableWithSearch().
* **Add Room:** Adds a room by entering the type, number, and availability status.
* **Update Room:** Updates details like room type and availability for the selected room.
* **Delete Room:** Removes the selected room from the database.

**4. Prescription Management**

* **Search Prescription:** Searches for prescriptions using a keyword.
* **Add Prescription:** Adds a prescription with PatientID, DoctorID, and notes.
* **Update Prescription:** Updates prescription details, such as the notes or issuing date.
* **Delete Prescription:** Deletes the selected prescription record.

**5. Lab Test Management**

* **Search Lab Test:** Searches lab tests using populateLabTestTableWithSearch().
* **Add Lab Test:** Captures patient ID, technician ID, test name, and test date to add a new lab test.
* **Update Lab Test:** Updates the selected test's details.
* **Delete Lab Test:** Deletes the selected lab test.

**6. Appointment Management**

* **Search Appointments:** Searches appointments using a keyword.
* **Add Appointment:** Inputs PatientID, DoctorID, appointment date, status, and notes to schedule a new appointment.
* **Update Appointment:** Updates details of the selected appointment, such as status or notes.
* **Delete Appointment:** Deletes the selected appointment record.

**7. Document Management**

* **Search Document:** Search for documents using keywords.
* **Add Document:** Adds a document specifying type, related ID, creator, and status.
* **Update Document:** Updates the selected document's details, such as the type or description.
* **Delete Document:** Deletes the selected document from the system.

**8. Billing Management**

* **Search Billing Records:** Searches billing records by keyword.
* **Add Billing Record:** Admins enter AppointmentID, total amount, payment status, and issue date to create a new record.
* **Update Billing Record:** Updates the selected billing record’s total amount or status.
* **Delete Billing Record:** Deletes the selected billing record.

**9. Lab Technicians Management**

* **Search Lab Technicians:** Searches technicians using populateLabTechniciansTable().
* **Add Lab Technician:** Adds a technician by entering their name, contact number, and email. The hire date is auto-assigned.
* **Update Lab Technician:** Updates details for the selected technician.
* **Delete Lab Technician:** Deletes the selected technician record.

**10. Staff Management**

* **Search Staff:** Searches staff members using keywords.
* **Add Staff:** Adds staff with fields like role, contact number, and join date.
* **Update Staff:** Updates details for the selected staff member, such as their role or contact number.
* **Delete Staff:** Deletes the selected staff member.

**11. Admissions Management**

* **Search Admissions:** Searches admissions by keyword.
* **Add Admission:** Captures patient ID, room ID, admission date, discharge date, and notes to create an admission.
* **Update Admission:** Updates details like room ID or discharge date for the selected admission.
* **Delete Admission:** Deletes the selected admission.

**12. Medical Records Management**

* **Search Medical Records:** Searches records using a keyword.
* **Add Medical Record:** Adds a record by specifying PatientID, DoctorID, date, and description.
* **Update Medical Record:** Updates the selected record's details.
* **Delete Medical Record:** Deletes the selected medical record.

**13. Pharmacy Inventory Management**

* **Search Inventory:** Searches for pharmacy inventory items using populatePharmacyItemsWithSearch().
* **Add Inventory Item:** Captures item name, quantity, price, expiry date, and supplier to add a new item.
* **Update Inventory Item:** Updates details like quantity or price for the selected item.
* **Delete Inventory Item:** Deletes the selected inventory item.

**14. Reports Management**

* **Search Reports:** Searches reports using keywords.
* **Add Report:** Adds a report by specifying the document ID, title, and content.
* **Update Report:** Updates details for the selected report.
* **Delete Report:** Deletes the selected report.

**15. User Management**

* **Search Users:** Searches users by keyword.
* **Add User:** Inputs user details like name, email, and password to create a new user.
* **Update User:** Updates details for the selected user.
* **Delete User:** Deletes the selected user.

**16. Invoice Details Management**

* **Search Invoice Details:** Searches invoice details by keyword.
* **Add Invoice Detail:** Adds details like DocumentID, item, quantity, and price.
* **Update Invoice Detail:** Updates selected details for invoice items.
* **Delete Invoice Detail:** Deletes the selected invoice detail.

**17. Prescription Details Management**

* **Search Prescription Details:** Searches details using populatePrescriptionDetailsWithSearch().
* **Add Prescription Detail:** Captures DocumentID, ItemID, and quantity to add a prescription detail.
* **Update Prescription Detail:** Updates the selected prescription detail.
* **Delete Prescription Detail:** Deletes the selected detail.

**18. Test Assignment Management**

* **Search Assignments:** Searches assignments using keywords.
* **Add Assignment:** Adds a new lab test assignment by capturing TestID and TechnicianID.
* **Update Assignment:** Updates details of the selected assignment.
* **Delete Assignment:** Deletes the selected lab test assignment.

**19. Document Access Management**

* **Search Document Access:** Searches access records using a keyword.
* **Add Access:** Captures DocumentID, UserID, and access level (read/write).
* **Update Access:** Updates the selected access record.
* **Delete Access:** Deletes the selected access record.

**20. Invoice Management**

* **Search Invoices:** Searches invoices by keyword using populateInvoiceTableWithSearch().
* **Add Invoice:** Adds a new invoice by capturing appointment ID, amount, payment status, and issue date.
* **Update Invoice:** Updates details of the selected invoice.
* **Delete Invoice:** Deletes the selected invoice record.

**Core Classes**

1. **AdminPanel.java**: The main class responsible for the GUI layout and user interactions.
2. **DAO Classes**:
   * DoctorDAO: Handles CRUD operations for doctors.
   * PatientDAO: Manages patients' records.
   * RoomDAO: Responsible for room management.
   * PrescriptionDAO, LabTestDAO, AppointmentDAO: Handle prescriptions, lab tests, and appointments, respectively.
   * BillingDAO: Manages billing and invoices.
   * StaffDAO: Handles staff-related operations.
   * DocumentDAO: Manages document creation and access.
3. **Model Classes**:
   * Each entity has a corresponding model class, such as Doctor, Patient1, Room, Prescription, Billing, etc., encapsulating the data and fields for that entity.

**2. GUI Components**

The GUI is designed using Java Swing, consisting of JTables, JButtons, JTextFields, and JOptionPanes for data manipulation and user interaction.

**Key GUI Components**

* **Tables (JTable)**: Used to display data for each entity.
* **Buttons (JButton)**: Provide actions for CRUD operations like "Add," "Update," "Delete," and "Search."
* **Text Fields (JTextField)**: Allow input for search keywords or specific field updates.
* **Dialogs (JOptionPane)**: Used for user prompts, confirmations, and error handling.

**3. CRUD Operations**

The Admin Panel supports Create, Read, Update, and Delete operations for all major entities.

**Core CRUD Methods**

1. **Create Operations**:
   * jButtonAddDoctorActionPerformed(): Adds a new doctor using input from dialogs.
   * Similar methods for other entities, such as jButtonAddPatientActionPerformed() and jButtonAddRoomActionPerformed().
2. **Read Operations**:
   * populate...Table(): Retrieves all records from the database and updates the corresponding JTable.
   * populate...TableWithSearch(String keyword): Retrieves filtered records based on the keyword.
3. **Update Operations**:
   * jButtonUpdateDoctorActionPerformed(): Updates the selected doctor's details.
   * Similar methods for other entities, such as jButtonUpdatePatientActionPerformed() and jButtonUpdatePrescriptionActionPerformed().
4. **Delete Operations**:
   * jButtonDeleteDoctorActionPerformed(): Deletes the selected doctor.
   * Similar methods for other entities, such as jButtonDeletePatientActionPerformed() and jButtonDeleteRoomActionPerformed().

**4. Search Functionality**

The panel supports dynamic search functionality for each entity, allowing administrators to quickly find specific records.

**Core Search Methods**

* btnSearchDoctorActionPerformed(): Searches doctors using the entered keyword.
* Similar methods for other entities, such as JButtonSearchPatientActionPerformed() and btnSearchLabTestActionPerformed().

Search results are dynamically displayed in the associated JTable using methods like populateDoctorTableWithSearch(String keyword).

**5. Table Population**

Data from the database is displayed in tables using the DefaultTableModel. Each entity has a dedicated method to populate its table.

**Core Table Population Methods**

1. populateDoctorTable(): Populates the doctor table with all records.
2. populateDoctorTableWithSearch(String keyword): Populates the doctor table with filtered results.
3. Similar methods exist for other entities, such as:
   * populatePatientTable()
   * populateRoomTable()
   * populatePrescriptionTableWithSearch(String keyword)
   * populateLabTestTable()

These methods fetch data from the DAO layer and update the JTable dynamically.

**6. Data Validation**

Input validation is performed before executing database operations. For example:

* Ensuring numerical fields like IDs are valid integers (NumberFormatException handling).
* Checking mandatory fields like DoctorID, PatientID, and others.

Error messages are displayed using JOptionPane.

**7. Database Interactions**

Database interactions are abstracted into DAO classes, ensuring separation of concerns. Each DAO class uses prepared statements to execute SQL queries securely.

**Core DAO Methods**

1. addEntity(Entity entity): Adds a new record to the database.
2. updateEntity(Entity entity): Updates an existing record.
3. deleteEntity(int id): Deletes a record by ID.
4. searchEntity(String keyword): Performs keyword-based search.
5. getAllEntities(): Retrieves all records for an entity.

Example: DoctorDAO has methods like addDoctor(), updateDoctor(), deleteDoctor(), searchDoctors(), and getAllDoctors().

**8. Error Handling**

Robust error handling is implemented throughout the code:

* **Try-Catch Blocks**: Handle exceptions like SQLException and NumberFormatException.
* **User-Friendly Messages**: Display error messages using JOptionPane.
* **Database Connection Handling**: Checks for connection issues and notifies the user if the connection fails.

5.Conclusion

This project, "Hospital Management and Document Tracking System," has been an enriching journey that provided significant insights into designing, developing, and deploying a real-world application. As students, our primary goal was to create a functional, efficient, and user-friendly system to address the challenges faced by healthcare facilities. By the end of this project, we were able to accomplish this objective through diligent planning, research, and implementation.

**5.1 Project Overview**

The system was developed as a comprehensive solution for hospitals to manage their day-to-day operations, ranging from user authentication to appointment scheduling, billing, prescription management, and document tracking. It also incorporated administrative controls for managing staff, inventory, rooms, and lab tests. The integration of Java Swing for the GUI and MSSQL for the backend provided a robust and secure platform that ensured smooth functionality.

**5.2 Challenges Encountered**

1. **Database Design Complexity:** Designing a relational database to handle interdependent data (e.g., appointments, billing, prescriptions) while maintaining data integrity required careful consideration of constraints, relationships, and normalization techniques.
2. **GUI Development:** Creating an intuitive user interface using Java Swing was challenging due to its extensive components and event-driven programming model. Ensuring that each module (admin, patient, and doctor) was tailored to the respective user was a learning experience.
3. **Business Logic Implementation:** Translating real-world healthcare operations into procedural and functional code posed a significant challenge. Implementing complex stored procedures, cascading rules, and user-defined functions for operations like checking room availability and generating invoices required meticulous attention to detail.
4. **Error Handling:** Ensuring the system could handle unexpected inputs, database connectivity issues, and user errors gracefully was another critical aspect of development.

**5.3 Learning Outcomes**

1. **Technical Skills:**
   * Mastery of SQL: Designing normalized databases, creating stored procedures, and managing constraints were invaluable skills.
   * Java Swing Development: Gaining hands-on experience in building a GUI from scratch, managing layouts, and handling events.
   * Security Practices: Understanding the importance of hashing passwords and implementing secure data operations.
2. **Problem-Solving:** Each challenge, from debugging to optimizing database queries, enhanced our analytical thinking and problem-solving skills.
3. **Team Collaboration:** Working as a team on different aspects of the project—GUI, backend, and database—taught us effective communication, task delegation, and time management.
4. **Real-World Application:** By simulating a hospital's operational workflow, we gained an understanding of how software systems can transform organizational efficiency in the healthcare industry.

**5.4 Strengths of the System**

1. **Comprehensive Features:** The system covers a wide range of hospital management operations, including appointments, billing, prescriptions, room management, lab tests, and document access.
2. **Role-Based Access Control:** By segregating functionalities for admins, doctors, and patients, the system ensures ease of use and secure access to data.
3. **Database Design:** The use of primary and foreign keys, cascading actions, and check constraints ensures data consistency and integrity.
4. **User-Friendly Interface:** The Java Swing GUI, with its well-organized layout and clear labels, enhances the user experience.
5. **Automation and Efficiency:** Automating repetitive tasks such as billing and room availability checks saves time and minimizes errors.

**5.5 Limitations and Areas for Improvement**

1. **Limited Deployment:** Currently, the system is designed for desktop environments, which limits its accessibility. Future iterations could include web and mobile versions.
2. **Scalability:** While the system is scalable for small to medium-sized hospitals, it would require enhancements to handle operations across multiple locations.
3. **Integration:** Real-time integration with third-party services, such as payment gateways and health insurance providers, is not yet implemented.
4. **User Experience:** Although the GUI is functional, it could be further improved by incorporating modern design principles and themes.

**5.6 Final Thoughts**

This project was a culmination of our theoretical knowledge and practical skills, allowing us to address a real-world problem effectively. The system not only demonstrates the potential of technology in transforming healthcare management but also serves as a strong foundation for future developments. We are proud of our achievements and excited about the possibilities for further enhancements to make this system even more impactful.