PROJECT

PATIENT INFORMATION SYSTEM USING JAVA HIBERNATE AND MYSQL

TEAM MEMBERS:

B. Balaji

N. Mukthi shree

K. Evangline Brito

ABSTRACT

In a given day, number of patients visits a hospital or a clinic. Many hospitals in India still manage the patient data manually. Hospitals will be able to save money and time if they have a good software program for managing patient's data. The idea is to develop web based patient management software that can be used to keep track of the patients registering in a hospital or clinic. Doctors and the rooms available in a hospital can be managed using this system. Also, this system should support accessing the previous visit histories of any patient, search for patients by name etc.

A few points to be noted about the system we are developing here

- A patient can be categorized as " In patient" or "Out Patient". If patient type is "In Patient", a bed will be assigned to the patient.
- A doctor will be assigned to each patient before the patient meets the doctor. Only one doctor can be assigned to a patient at a given time.
- A patient can visit the hospital any number of times

CONTENTS

INTRODUCTION

1.1. INTRODUCTION TO PROJECT

Hospital administrators are often inundated with information about a large number of patients and their visits to the hospital that need to be organized and kept up-to-date. The patient management system is a web based application that is designed and developed for hospital administrators and doctors to organize information on patient visits. The system intends to facilitate several steps in the process from the patient registration and to the patient evaluation. During this process, there will be many tasks that have to be handled by this system including maintaining complete information. The main objective of the system is to provide the administration staff and doctors with an easily maintainable information system for patient registration, visit scheduling and patient tracking with latest information.

ORGANIZATION PROFILE

SOFTWARE SOLUTIONS

Chigulla Software is an IT solution provider for a dynamic environment where business and technology strategies converge. Their approach focuses on new ways of business combining IT innovation and adoption while also leveraging an organization's current IT assets. Their work with large global corporations and new products or services and to implement prudent business and technology strategies in today's environment.

CHIGULLA SOFTWARE RANGE OF EXPERTISE INCLUDES:

- Software Development Services
- Engineering Services
- Systems Integration
- Customer Relationship Management
- Product Development
- Electronic Commerce

- Consulting
- IT Outsourcing

We apply technology with innovation and responsibility to achieve two broad objectives:

- Effectively address the business issues our customers face today.
- Generate new opportunities that will help them stay ahead in the future.

THIS APPROACH RESTS ON:

- A strategy where we architect, integrate and manage technology services and solutions - we call it AIM for success.
- A robust offshore development methodology and reduced demand on customer resources.
- A focus on the use of reusable frameworks to provide cost and times benefits.

They combine the best people, processes and technology to achieve excellent results - consistency. We offer customers the advantages of:

SPEED:

They understand the importance of timing, of getting there before the competition. A rich portfolio of reusable, modular frameworks helps jump-start projects. Tried and tested methodology ensures that we follow a predictable, low - risk path to achieve results. Our track record is testimony to complex projects delivered within and evens before schedule.

EXPERTISE:

Our teams combine cutting edge technology skills with rich domain expertise. What's equally important - they share a strong customer orientation that means they actually start by listening to the customer. They're focused on coming up with solutions that serve customer requirements today and anticipate future needs.

A FULL SERVICE PORTFOLIO:

They offer customers the advantage of being able to Architect, integrate and manage technology services. This means that they can rely on one, fully accountable source instead of trying to integrate disparate multi vendor solutions.

SERVICES:

Chigulla Software is providing it's services to companies which are in the field of production, quality control etc With their rich expertise and experience and information technology they are in best position to provide software solutions to distinct business requirements.

PURPOSE OF THE PROJECT

In a given day, number of patients visits a hospital or a clinic. Many hospitals in India still manage the patient data manually. Hospitals will be able to save money and time if they have a good software program for managing patient's data. The idea is to develop web based patient management software that can be used to keep track of the patients registering in a hospital or clinic. Doctors and the rooms available in a hospital can be managed using this system. Also, this system should support accessing the previous visit histories of any patient, search for patients by name etc.

A few points to be noted about the system we are developing here

- A patient can be categorized as " In patient" or "Out Patient". If patient type is "In Patient", a bed will be assigned to the patient.
- A doctor will be assigned to each patient before the patient meets the doctor. Only one doctor can be assigned to a patient at a given time.
- A patient can visit the hospital any number of times

PROBLEM IN EXISTING SYSTEM

- Cannot Upload and Download the latest updates.
- No use of Web Services and Remoting.
- Risk of mismanagement and of data when the project is under development.
- Less Security.
- No proper coordination between different Applications and Users.
- Fewer Users - Friendly.

SOLUTION OF THESE PROBLEMS

The development of the new system contains the following activities, which try to automate the entire process keeping in view of the database integration approach.

1. User friendliness is provided in the application with various controls.
2. The system makes the overall project management much easier and flexible.
3. Readily upload the latest updates, allows user to download the alerts by clicking the URL.
4. There is no risk of data mismanagement at any level while the project development is under process.
5. It provides high level of security with different level of authentication.

SYSTEM ANALYSIS

INTRODUCTION

After analyzing the requirements of the task to be performed, the next step is to analyze the problem and understand its context. The first activity in the phase is studying the existing system and other is to understand the requirements and domain of the new system. Both the activities are equally important, but the first activity serves as a basis of giving the functional specifications and then successful design of the proposed system. Understanding the properties and requirements of a new system is more difficult and requires creative thinking and understanding of existing running system is also difficult, improper understanding of present system can lead diversion from solution.

ANALYSIS MODEL

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, "A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.
The steps for Spiral Model can be generalized as follows:

- The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
- A preliminary design is created for the new system.
- A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
- A second prototype is evolved by a fourfold procedure:

1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
2. Defining the requirements of the second prototype.
3. Planning an designing the second prototype.
4. Constructing and testing the second prototype.

- At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involved development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer's judgment, result in a less-than-satisfactory final product.
- The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
- The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
- The final system is constructed, based on the refined prototype.
- The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

MODULES OF THE SYSTEM

MODULES:

Patients Table:

The "Patients" table stores information about individual patients, including their unique patient IDs, names, contact details, and other relevant demographic information. This table serves as the central repository for patient data and provides the foundation for various interactions and relationships with other tables.

Appointments Table:

The "Appointments" table records details about scheduled appointments between patients and healthcare providers. It includes the appointment ID, patient ID (referencing the Patients table), doctor ID (referencing the Doctors table), appointment date, time, status, and any additional notes or comments related to the appointment.

Doctors Table:

The "Doctors" table stores information about healthcare providers, including doctors, physicians, and specialists. It contains details such as the doctor's unique ID, name, specialization, contact information, and other relevant data. The doctor ID from this table is referenced in the Appointments table to establish the connection between doctors and their scheduled appointments.

Diagnosis Table:

The "Diagnosis" table captures medical diagnoses made by healthcare providers for individual patients. It includes a diagnosis ID, patient ID (referencing the Patients table), doctor ID (referencing the Doctors table), diagnosis date, the medical condition or disease diagnosed, and any additional notes or comments related to the diagnosis.

Billing Table:

The "Billing" table manages patient billing and financial transactions. It records details such as a unique billing ID, patient ID (referencing the Patients table), billing date, description of the billing transaction (e.g., consultation fee, lab test charge), the amount charged or paid, payment status, and payment date (if applicable).

DESCRIPTION:

Project Description:

The Patient Information System (PIS) is a comprehensive healthcare management application designed to streamline the handling of patient data, appointments, medical diagnoses, and billing processes. The system aims to enhance the efficiency and accuracy of healthcare service delivery while improving patient care and administrative workflows. Developed using JDK 20, Hibernate, and SQL, the project offers a robust and scalable solution for healthcare facilities.

Key Concepts:

Patient Management:

The system allows healthcare providers to register and manage patient information, including demographics, medical history, and contact details.

Comprehensive patient records ensure accurate treatment and personalized care.

Appointment Scheduling:

Healthcare providers can schedule and manage patient appointments efficiently.

Appointment reminders and notifications enhance patient engagement.

Medical Diagnoses:

The system records and manages medical diagnoses made by healthcare providers for individual patients.

Accurate diagnosis records aid in treatment planning and continuity of care.

Billing and Payments:

The system manages patient billing, invoicing, and payment tracking. Transparent financial management ensures accurate billing and payments.

Data Security and Privacy:

The project implements data security measures to protect patient privacy and comply with regulations.

Role-based access control ensures data is accessible only to authorized personnel.

User-Friendly Interface:

The application features an intuitive user interface for easy navigation and efficiency.

Healthcare providers and administrators can quickly access and update patient information.

By focusing on these core concepts, the Patient Information System project aims to improve patient care, enhance healthcare workflows, and provide a reliable and secure platform for healthcare providers and administrators.

SYSTEM REQUIREMENTS SPECIFICATIONS

HARDWARE REQUIREMENTS:

- Pentium 5 Processor and Above
- RAM 512MB and Above
- HDD 20 GB Hard Disk Space and Above

SOFTWARE REQUIREMENTS:

- WINDOWS OS (XP / 2000 / 200 Server / 2003 Server)
- Eclipse IDE
- Hibernate
- SQL Server 2000 Enterprise Edition

PROPOSED SYSTEM

To debug the existing system, remove procedures those cause data redundancy, make navigational sequence proper. To provide information about audits on different level and also to reflect the current work status depending on organization/auditor or date. To build strong password mechanism.

NEED FOR COMPUTERIZATION

We all know the importance of computerization. The world is moving ahead at lightening speed and every one is running short of time. One always wants to get the information and perform a task he/she/they desire(s) within a short period of time and too with amount of efficiency and accuracy. The application areas for the computerization have been selected on the basis of following factors:

- Minimizing the manual records kept at different locations.
- There will be more data integrity.
- Facilitating desired information display, very quickly, by retrieving information from users.
- Facilitating various statistical information which helps in decision-making?
- To reduce manual efforts in activities that involved repetitive work.
- Updating and deletion of such a huge amount of data will become easier.

PROCESS MODELS USED WITH JUSTIFICATION

ACCESS CONTROL FOR DATA WHICH REQUIRE USER AUTHENTICATION

The following commands specify access control identifiers and they are typically used to authorize and authenticate the user (command codes are shown in parentheses)

USER NAME (USER)

The user identification is that which is required by the server for access to its file system. This command will normally be the first command transmitted by the user after the control connections are made (some servers may require this).

PASSWORD (PASS)

This command must be immediately preceded by the user name command, and, for some sites, completes the user's identification for access control. Since password information is quite sensitive, it is desirable in general to "mask" it or suppress type out.

Feasibility Report

Preliminary investigation examine project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

- Technical Feasibility
- Operation Feasibility
- Economical Feasibility

3.2. Operational Feasibility

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization's operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. Some of the important issues raised are to test the operational feasibility of a project includes the following: -

- Is there sufficient support for the management from the users?
- Will the system be used and work properly if it is being developed and implemented?
- Will there be any resistance from the user that will undermine the possible application benefits?

This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits.

The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

Economic Feasibility

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economical feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs.

SELECTED SOFTWARE

INTRODUCTION TO HIBERNATE

Hibernate is used to overcome the limitations of JDBC like:

JDBC code is dependent upon the Database software being used i.e. our persistence logic is dependent, because of using JDBC. Here we are inserting a record into Employee table but our query is Database software-dependent i.e. Here we are using MySQL. But if we change our Database then this query won't work.

If working with JDBC, changing of Database in middle of the project is very costly.

JDBC code is not portable code across the multiple database software.

In JDBC, Exception handling is mandatory. Here We can see that we are handling lots of Exception for connection. -line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.

While working with JDBC, There is no support Object-level relationship.

In JDBC, there occurs a Boilerplate problem i.e. For each and every project we have to write the below code. That increases the code length and reduce the readability.

MySQL SERVER

A database management, or DBMS, gives the user access to their data and helps them transform the data into information. Such database management systems include dBase, paradox, IMS, SQL Server and SQL Server. These systems allow users to create, update and extract information from their database.

A database is a structured collection of data. Data refers to the characteristics of people, things and events. SQL Server stores each data item in its own fields. In SQL Server, the fields relating to a particular person, thing or event are bundled together to form a single complete unit of data, called a record (it can also be referred to as raw or an occurrence). Each record is made up of a number of fields. No two fields in a record can have the same field name.

During an SQL Server Database design project, the analysis of your business needs identifies all the fields or attributes of interest. If your business needs change over time, you define any additional fields or change the definition of existing fields.

SQL SERVER TABLES

SQL Server stores records relating to each other in a table. Different tables are created for the various groups of information. Related tables are grouped together to form a database.

PRIMARY KEY

Every table in SQL Server has a field or a combination of fields that uniquely identifies each record in the table. The Unique identifier is called the Primary Key, or simply the Key. The primary key provides the means to distinguish one record from all other in a table. It allows the user and the database system to identify, locate and refer to one particular record in the database.

RELATIONAL DATABASE

Sometimes all the information of interest to a business operation can be stored in one table. SQL Server makes it very easy to link the data in multiple tables. Matching an employee to the department in which they work is one example. This is what makes SQL Server a relational database management system, or RDBMS. It stores data in two or more

tables and enables you to define relationships between the table and enables you to define relationships between the tables.

FOREIGN KEY

When a field is one table matches the primary key of another field is referred to as a foreign key. A foreign key is a field or a group of fields in one table whose values match those of the primary key of another table.

REFERENTIAL INTEGRITY

Not only does SQL Server allow you to link multiple tables, it also maintains consistency between them. Ensuring that the data among related tables is correctly matched is referred to as maintaining referential integrity.

DATA ABSTRACTION

A major purpose of a database system is to provide users with an abstract view of the data. This system hides certain details of how the data is stored and maintained. Data abstraction is divided into three levels.

Physical level: This is the lowest level of abstraction at which one describes how the data are actually stored.

Conceptual Level: At this level of database abstraction all the attributed and what data are actually stored is described and entries and relationship among them.

View level: This is the highest level of abstraction at which one describes only part of the database.

ADVANTAGES OF RDBMS

- Redundancy can be avoided
- Inconsistency can be eliminated
- Data can be Shared
- Standards can be enforced
- Security restrictions ca be applied
- Integrity can be maintained
- Conflicting requirements can be balanced

- Data independence can be achieved.

DISADVANTAGES OF DBMS

A significant disadvantage of the DBMS system is cost. In addition to the cost of purchasing of developing the software, the hardware has to be upgraded to allow for the extensive programs and the workspace required for their execution and storage. While centralization reduces duplication, the lack of duplication requires that the database be adequately backed up so that in case of failure the data can be recovered.

FEATURES OF SQL SERVER (RDBMS)

SQL SERVER is one of the leading database management systems (DBMS) because it is the only Database that meets the uncompromising requirements of today's most demanding information systems. From complex decision support systems (DSS) to the most rigorous online transaction processing (OLTP) application, even application that require simultaneous DSS and OLTP access to the same critical data, SQL Server leads the industry in both performance and capability

SQL SERVER is a truly portable, distributed, and open DBMS that delivers unmatched performance, continuous operation and support for every database.
SQL SERVER RDBMS is high performance fault tolerant DBMS which is specially designed for online transactions processing and for handling large database application.

SQL SERVER with transactions processing option offers two features which contribute to very high level of transaction processing throughput, which are

- The row level lock manager

ENTERPRISE WIDE DATA SHARING

The unrivaled portability and connectivity of the SQL SERVER DBMS enables all the systems in the organization to be linked into a singular, integrated computing resource.

PORTABILITY

SQL SERVER is fully portable to more than 80 distinct hardware and operating systems platforms, including UNIX, MSDOS, OS/2, Macintosh and dozens of proprietary

platforms. This portability gives complete freedom to choose the database sever platform that meets the system requirements.

OPEN SYSTEMS

SQL SERVER offers a leading implementation of industry –standard SQL. SQL Server's open architecture integrates SQL SERVER and non –SQL SERVER DBMS with industries most comprehensive collection of tools, application, and third party software products SQL Server's Open architecture provides transparent access to data from other relational database and even non-relational database.

DISTRIBUTED DATA SHARING

SQL Server's networking and distributed database capabilities to access data stored on remote server with the same ease as if the information was stored on a single local computer. A single SQL statement can access data at multiple sites. You can store data where system requirements such as performance, security or availability dictate.

UNMATCHED PERFORMANCE

The most advanced architecture in the industry allows the SQL SERVER DBMS to deliver unmatched performance.

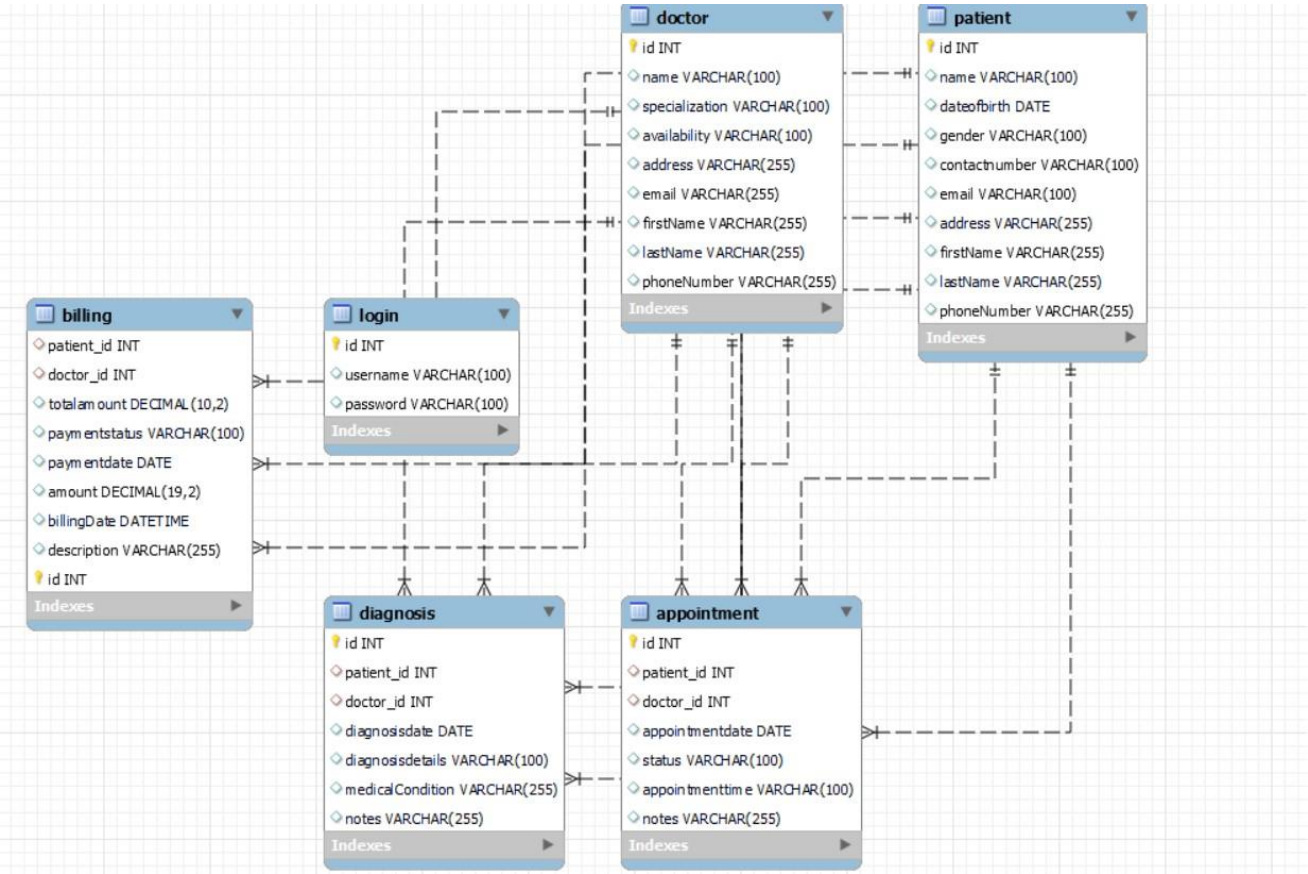
SOPHISTICATED CONCURRENCY CONTROL

Real World applications demand access to critical data. With most database Systems application becomes "contention bound" – which performance is limited not by the CPU power or by disk I/O, but user waiting on one another for data access . SQL Server employs full, unrestricted row-level locking and contention free queries to minimize and in many cases entirely eliminates contention wait times.

E – R DIAGRAMS

- The relation upon the system is structure through a conceptual ER-Diagram, which not only specifics the existential entities but also the standard relations through which the system exists and the cardinalities that are necessary for the system state to continue.
- The entity Relationship Diagram (ERD) depicts the relationship between the data objects. The ERD is the notation that is used to conduct the date modeling activity the attributes of each data object noted is the ERD can be described resign a data object descriptions.
- The set of primary components that are identified by the ERD are
  - ◆ Data object
  - ◆ Relationships
  - ◆ Attributes
  - ◆ Various types of indicators.

The primary purpose of the ERD is to represent data objects and their relationships.


doctor
id INT
name VARCHAR(100)
specialization VARCHAR(100)
availability VARCHAR(100)
address VARCHAR(255)
email VARCHAR(255)
firstName VARCHAR(255)
lastName VARCHAR(255)
phoneNumber VARCHAR(255)
Indexes
patient
id INT
name VARCHAR(100)
dateofbirth DATE
gender VARCHAR(100)
contactnumber VARCHAR(100)
email VARCHAR(100)
address VARCHAR(255)
firstName VARCHAR(255)
lastName VARCHAR(255)
phoneNumber VARCHAR(255)
Indexes
billing
patient_id INT
doctor_id INT
totalamount DECIMAL(10,2)
paymentstatus VARCHAR(100)
paymentdate DATE
amount DECIMAL(19,2)
billingDate DATETIME
description VARCHAR(255)
id INT
Indexes
login
id INT
username VARCHAR(100)
password VARCHAR(100)
Indexes
diagnosis
id INT
patient_id INT
doctor_id INT
diagnosisdate DATE
diagnosisdetails VARCHAR(100)
medicalCondition VARCHAR(255)
notes VARCHAR(255)
Indexes
appointment
id INT
patient_id INT
doctor_id INT
appointmentdate DATE
status VARCHAR(100)
appointmenttime VARCHAR(100)
notes VARCHAR(255)
Indexes

SYSTEM CODING

SOURCE CODE:

Main :

```
package com.patient.information.system;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.query.Query;

import java.util.Scanner;
import java.util.List;
import java.math.BigDecimal;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.regex.Pattern;
import java.util.regex.Matcher;


public class MainApp {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      SessionFactory sessionFactory = buildSessionFactory();

      boolean loggedIn = false;

      while (!loggedIn) {
         System.out.print("Enter username: ");
         String username = scanner.nextLine();

         System.out.print("Enter password: ");
         String password = scanner.nextLine();

         if (checkLogin(username, password, sessionFactory)) {
            System.out.println("Login successful!");
```

```
            loggedIn = true;
        } else {
            System.out.println("Invalid credentials. Do you want to retry?
(y/n)");
            char choice = scanner.next().charAt(0);
            if (choice == 'y' || choice == 'Y') {
                // Continue to the next iteration of the loop
                scanner.nextLine(); // Clear the newline character
                continue;
            } else {
                System.out.println("Exiting program.");
                System.exit(0);
            }
        }
    }

    // After successful login, display the menu options
    displayMenu(scanner, sessionFactory);

    sessionFactory.close();
    scanner.close();
  }

  private static void displayMenu(Scanner scanner, SessionFactory
sessionFactory) {
    boolean exit = false;
    while (!exit) {
        System.out.println("\nMenu:");
        System.out.println("1. Manage Patients");
        System.out.println("2. Manage Doctors");
        System.out.println("3. Manage Appointments");
        System.out.println("4. Manage Diagnoses");
        System.out.println("5. Manage Billings");
        System.out.println("6. Exit");

        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        scanner.nextLine(); // Clear the newline character

        switch (choice) {
            case 1:
                managePatients(sessionFactory, scanner);
                break;
            case 2:
```

```
                manageDoctors(sessionFactory, scanner);
                break;
            case 3:
                manageAppointments(sessionFactory, scanner);
                break;
            case 4:
                manageDiagnoses(sessionFactory, scanner);
                break;
            case 5:
                manageBillings(sessionFactory, scanner);
                break;
            case 6:
                exit = true;
                System.out.println("Exiting program.");
                break;
            default:
                System.out.println("Invalid choice. Please select a valid
option.");
            }
        }
    }

    private static void managePatients(SessionFactory sessionFactory,
Scanner scanner) {
        // Implement patient management here...

            boolean exit = false;

            while (!exit) {
                System.out.println("\nPatient Management:");
                System.out.println("1. Add Patient");
                System.out.println("2. View Patients");
                System.out.println("3. Update Patient Details");
                System.out.println("4. Delete Patient");
                System.out.println("5. Go back to main menu");

                System.out.print("Enter your choice: ");
                int choice = scanner.nextInt();
                scanner.nextLine(); // Clear the newline character

                switch (choice) {
                    case 1:
                        addPatient(sessionFactory, scanner);
                        break;
```

```
                case 2:
                   viewPatients(sessionFactory);
                   break;
                case 3:
                   updatePatientDetails(sessionFactory, scanner);
                   break;
                case 4:
                   deletePatient(sessionFactory, scanner);
                   break;
                case 5:
                   exit = true;
                   System.out.println("Returning to main menu.");
                   break;
                default:
                   System.out.println("Invalid choice. Please select a valid
option.");
             }
          }
   }

      private static void addPatient(SessionFactory sessionFactory,
Scanner scanner) {
          // Implement adding a new patient here...
              try (Session session = sessionFactory.openSession()) {
                   Transaction transaction = session.beginTransaction();

                   System.out.println("Enter patient details:");
                   System.out.print("First Name: ");
                   String firstName = scanner.nextLine();
                   System.out.print("Last Name: ");
                   String lastName = scanner.nextLine();
                   System.out.print("Gender: ");
                   String gender = scanner.nextLine();
                   System.out.print("Date of Birth (yyyy-MM-dd): ");
                   String dobString = scanner.nextLine();
                   Date dateOfBirth = parseDate(dobString);
                   System.out.print("Phone Number: ");
                   String phoneNumber = scanner.nextLine();
                  String phoneNumberRegex = "^[0-9]{10}$"; // 10-digit
number
                   Pattern pattern = Pattern.compile(phoneNumberRegex);
                   Matcher matcher = pattern.matcher(phoneNumber);

                   if (!matcher.matches()) {
```

```
                    System.out.println("Invalid phone number format.
Please enter a 10-digit number.");
                    return;
                }
                System.out.print("Address: ");
                String address = scanner.nextLine();

                Patient patient = new Patient();
                patient.setFirstName(firstName);
                patient.setLastName(lastName);
                patient.setGender(gender);
                patient.setDateOfBirth(dateOfBirth);
                patient.setPhoneNumber(phoneNumber);
                patient.setAddress(address);

                session.save(patient);
                transaction.commit();

                System.out.println("Patient added successfully.");
            } catch (Exception e) {
                e.printStackTrace();
            }
    }

    private static void viewPatients(SessionFactory sessionFactory) {
        try (Session session = sessionFactory.openSession()) {
            String hql = "FROM Patient";
            Query<Patient> query = session.createQuery(hql,
Patient.class);
            List<Patient> patients = query.list();

            if (!patients.isEmpty()) {
                System.out.println("List of Patients:");
                for (Patient patient : patients) {
                    System.out.println("Patient ID: " + patient.getId());
                    System.out.println("Patient Name: " +
patient.getFirstName() + " " + patient.getLastName());
                    System.out.println("Gender: " + patient.getGender());
                    System.out.println("Date of Birth: " +
patient.getDateOfBirth());
                    System.out.println("Phone Number: " +
patient.getPhoneNumber());
                    System.out.println("Address: " + patient.getAddress());
                    System.out.println("--------------------");
```

```
            }
         } else {
            System.out.println("No patients found.");
         }
      } catch (Exception e) {
         e.printStackTrace();
         System.out.println("An error occurred while viewing
patients.");
      }
   }


   private static void updatePatientDetails(SessionFactory
sessionFactory, Scanner scanner) {
      // Implement updating patient details here...
         try (Session session = sessionFactory.openSession()) {
            Transaction transaction = session.beginTransaction();

            System.out.print("Enter patient ID to update: ");
            long patientId = scanner.nextLong();
            scanner.nextLine(); // Clear the newline character

            Patient patient = session.get(Patient.class, patientId);

            if (patient == null) {
               System.out.println("Patient not found.");
            } else {
               System.out.println("Current patient details:");
               System.out.println(patient);

               System.out.println("Enter updated patient details:");
               System.out.print("First Name: ");
               patient.setFirstName(scanner.nextLine());
               System.out.print("Last Name: ");
               patient.setLastName(scanner.nextLine());
               System.out.print("Gender: ");
               patient.setGender(scanner.nextLine());
               System.out.print("Date of Birth (yyyy-MM-dd): ");
               String dobString = scanner.nextLine();
               patient.setDateOfBirth(parseDate(dobString));
               System.out.print("Phone Number: ");
               patient.setPhoneNumber(scanner.nextLine());
               System.out.print("Address: ");
               patient.setAddress(scanner.nextLine());
```

```
                    session.update(patient);
                    transaction.commit();

                    System.out.println("Patient details updated
successfully.");
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        private static void deletePatient(SessionFactory sessionFactory,
Scanner scanner) {
            // Implement deleting a patient here...
            try (Session session = sessionFactory.openSession()) {
                Transaction transaction = session.beginTransaction();

                System.out.print("Enter patient ID to delete: ");
                long patientId = scanner.nextLong();
                scanner.nextLine(); // Clear the newline character

                Patient patient = session.get(Patient.class, patientId);

                if (patient == null) {
                    System.out.println("Patient not found.");
                } else {
                    session.delete(patient);
                    transaction.commit();

                    System.out.println("Patient deleted successfully.");
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }



    private static void manageDoctors(SessionFactory sessionFactory,
Scanner scanner) {
        // Implement doctor management here...
        while (true) {
            System.out.println("\nDoctor Management Menu:");
```

```
            System.out.println("1. Add New Doctor");
            System.out.println("2. View Doctors");
            System.out.println("3. Update Doctor Details");
            System.out.println("4. Delete Doctor");
            System.out.println("5. Back to Main Menu");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Clear the newline character

            switch (choice) {
               case 1:
                  addDoctor(sessionFactory, scanner);
                  break;
               case 2:
                  viewDoctors(sessionFactory);
                  break;
               case 3:
                  updateDoctorDetails(sessionFactory, scanner);
                  break;
               case 4:
                  deleteDoctor(sessionFactory, scanner);
                  break;
               case 5:
                  return; // Exit the method to go back to the main menu
               default:
                  System.out.println("Invalid choice. Please choose again.");
            }
         }
   }
       private static void addDoctor(SessionFactory sessionFactory,
Scanner scanner) {
              try (Session session = sessionFactory.openSession()) {
                  Transaction transaction = session.beginTransaction();

                  Doctor doctor = new Doctor();

                  System.out.print("Enter First Name: ");
                  doctor.setFirstName(scanner.nextLine());

                  System.out.print("Enter Last Name: ");
                  doctor.setLastName(scanner.nextLine());

                  System.out.print("Enter Specialization: ");
```

```
                    doctor.setSpecialization(scanner.nextLine());

                    System.out.print("Enter Email: ");
                    String email = scanner.nextLine();
                    String emailRegex = "^[A-Za-z0-9+_.-]+@(.+)$"; //
Basic email format
                    Pattern pattern = Pattern.compile(emailRegex);
                    Matcher matcher = pattern.matcher(email);

                    if (!matcher.matches()) {
                       System.out.println("Invalid email address format.
Please enter a valid email address.");
                       return;
                    }


                    System.out.print("Enter Phone Number: ");
                    String phoneNumber = scanner.nextLine();

                    String phoneNumberRegex = "^[0-9]{10}$"; // 10-digit
number
                    Pattern phonePattern =
Pattern.compile(phoneNumberRegex);
                    Matcher phoneMatcher =
phonePattern.matcher(phoneNumber);

                    if (!phoneMatcher.matches()) {
                       System.out.println("Invalid phone number format.
Please enter a 10-digit number.");
                       return;
                    }

                    doctor.setPhoneNumber(phoneNumber);


                    System.out.print("Enter Address: ");
                    doctor.setAddress(scanner.nextLine());

                    session.save(doctor);
                    transaction.commit();

                    System.out.println("Doctor added successfully!");
                 } catch (Exception e) {
                    e.printStackTrace();
```

```
            }
        }

    private static void viewDoctors(SessionFactory sessionFactory) {
            try (Session session = sessionFactory.openSession()) {
                String hql = "FROM Doctor";
                Query<Doctor> query = session.createQuery(hql,
Doctor.class);
                List<Doctor> doctors = query.list();

                if (!doctors.isEmpty()) {
                    System.out.println("List of Doctors:");
                    for (Doctor doctor : doctors) {
                        System.out.println("Doctor ID: " + doctor.getId());
                        System.out.println("Doctor Name: " +
doctor.getFirstName() + " " + doctor.getLastName());
                        System.out.println("Specialization: " +
doctor.getSpecialization());
                        System.out.println("Email: " + doctor.getEmail());
                        System.out.println("Phone Number: " +
doctor.getPhoneNumber());
                        System.out.println("Address: " +
doctor.getAddress());
                        System.out.println("--------------------");
                    }
                } else {
                    System.out.println("No doctors found.");
                }
            } catch (Exception e) {
                e.printStackTrace();
                System.out.println("An error occurred while viewing
doctors.");
            }
        }

    private static void updateDoctorDetails(SessionFactory
sessionFactory, Scanner scanner) {
            try (Session session = sessionFactory.openSession()) {
                Transaction transaction = session.beginTransaction();

                System.out.print("Enter Doctor ID to update: ");
                long doctorId = scanner.nextLong();
                scanner.nextLine(); // Consume newline
```

```
                Doctor doctor = session.get(Doctor.class, doctorId);

                if (doctor == null) {
                   System.out.println("Doctor not found.");
                   return;
                }

                System.out.println("Current Doctor Details:");
                System.out.println(doctor);

                System.out.print("Enter New First Name (or press Enter to
skip): ");
                String newFirstName = scanner.nextLine();
                if (!newFirstName.isEmpty()) {
                   doctor.setFirstName(newFirstName);
                }


                session.update(doctor);
                transaction.commit();

                System.out.println("Doctor details updated
successfully!");
             } catch (Exception e) {
                e.printStackTrace();
             }
          }

      private static void deleteDoctor(SessionFactory sessionFactory,
Scanner scanner) {
             try (Session session = sessionFactory.openSession()) {
                Transaction transaction = session.beginTransaction();

                System.out.print("Enter Doctor ID to delete: ");
                long doctorId = scanner.nextLong();
                scanner.nextLine(); // Consume newline

                Doctor doctor = session.get(Doctor.class, doctorId);

                if (doctor == null) {
                   System.out.println("Doctor not found.");
                   return;
                }
```

```
                session.delete(doctor);
                transaction.commit();

                System.out.println("Doctor deleted successfully!");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }


    private static void manageAppointments(SessionFactory
sessionFactory, Scanner scanner) {
        // Implement appointment management here...
         boolean exit = false;

          while (!exit) {
             System.out.println("Appointments Management");
             System.out.println("1. Add Appointment");
             System.out.println("2. View Appointments");
             System.out.println("3. Update Appointment");
             System.out.println("4. Delete Appointment");
             System.out.println("5. Back to Main Menu");
             System.out.print("Enter your choice: ");

             int choice = scanner.nextInt();
             scanner.nextLine(); // Consume newline

             switch (choice) {
                case 1:
                   addAppointment(sessionFactory, scanner);
                   break;
                case 2:
                   viewAppointments(sessionFactory);
                   break;
                case 3:
                   updateAppointmentDetails(sessionFactory, scanner);
                   break;
                case 4:
                   deleteAppointment(sessionFactory, scanner);
                   break;
                case 5:
                   exit = true;
                   break;
                default:
```

```
                System.out.println("Invalid choice. Please select a valid
option.");
            }
        }
    }
  private static void addAppointment(SessionFactory sessionFactory,
Scanner scanner) {
      try (Session session = sessionFactory.openSession()) {
         Transaction transaction = session.beginTransaction();

         Appointment appointment = new Appointment();

         System.out.print("Enter Patient ID: ");
         long patientId = scanner.nextLong();
         scanner.nextLine(); // Consume newline
         Patient patient = session.get(Patient.class, patientId);
         if (patient == null) {
            System.out.println("Patient not found.");
            return;
         }
         appointment.setPatient(patient);

         System.out.print("Enter Doctor ID: ");
         long doctorId = scanner.nextLong();
         scanner.nextLine(); // Consume newline
         Doctor doctor = session.get(Doctor.class, doctorId);
         if (doctor == null) {
            System.out.println("Doctor not found.");
            return;
         }
         appointment.setDoctor(doctor);

         System.out.print("Enter Appointment Date (yyyy-MM-dd): ");
         String appointmentDateStr = scanner.nextLine();
         try {
            Date appointmentDate = parseDate(appointmentDateStr);
            appointment.setAppointmentDate(appointmentDate);
         } catch (ParseException e) {
            System.out.println("Invalid date format. Appointment not
added.");
            return;
         }

         System.out.print("Enter Appointment Status: ");
```

```
            appointment.setStatus(scanner.nextLine());

            System.out.print("Enter Notes: ");
            appointment.setNotes(scanner.nextLine());

            session.save(appointment);
            transaction.commit();

            System.out.println("Appointment added successfully!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private static void viewAppointments(SessionFactory sessionFactory) {
        try (Session session = sessionFactory.openSession()) {
            String hql = "FROM Appointment";
            Query<Appointment> query = session.createQuery(hql,
Appointment.class);
            List<Appointment> appointments = query.list();

            if (!appointments.isEmpty()) {
                System.out.println("List of Appointments:");
                for (Appointment appointment : appointments) {
                    System.out.println("Appointment ID: " +
appointment.getId());
                    System.out.println("Patient Name: " +
appointment.getPatient().getFirstName() + " " +
appointment.getPatient().getLastName());
                    System.out.println("Doctor Name: " +
appointment.getDoctor().getFirstName() + " " +
appointment.getDoctor().getLastName());
                    System.out.println("Appointment Date: " +
appointment.getAppointmentDate());
                    System.out.println("Status: " + appointment.getStatus());
                    System.out.println("Notes: " + appointment.getNotes());
                    System.out.println("--------------------");
                }
            } else {
                System.out.println("No appointments found.");
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("An error occurred while viewing
appointments.");
```

```
        }
    }

    private static void updateAppointmentDetails(SessionFactory
sessionFactory, Scanner scanner) {
        try (Session session = sessionFactory.openSession()) {
            Transaction transaction = session.beginTransaction();

            System.out.print("Enter Appointment ID to update: ");
            long appointmentId = scanner.nextLong();
            scanner.nextLine(); // Consume newline

            Appointment appointment = session.get(Appointment.class,
appointmentId);

            if (appointment == null) {
                System.out.println("Appointment not found.");
                return;
            }

            System.out.println("Current Appointment Details:");
            System.out.println(appointment);

            System.out.print("Enter New Appointment Date (yyyy-MM-dd) (or
press Enter to skip): ");
            String newAppointmentDateStr = scanner.nextLine();
            if (!newAppointmentDateStr.isEmpty()) {
                try {
                    Date newAppointmentDate =
parseDate(newAppointmentDateStr);
                    appointment.setAppointmentDate(newAppointmentDate);
                } catch (ParseException e) {
                    System.out.println("Invalid date format. Appointment date
not updated.");
                }
            }


            session.update(appointment);
            transaction.commit();

            System.out.println("Appointment details updated successfully!");
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
    }
    private static void deleteAppointment(SessionFactory sessionFactory,
Scanner scanner) {
        try (Session session = sessionFactory.openSession()) {
            Transaction transaction = session.beginTransaction();

            System.out.print("Enter Appointment ID to delete: ");
            long appointmentId = scanner.nextLong();
            scanner.nextLine(); // Consume newline

            Appointment appointment = session.get(Appointment.class,
appointmentId);

            if (appointment == null) {
                System.out.println("Appointment not found.");
                return;
            }

            session.delete(appointment);
            transaction.commit();

            System.out.println("Appointment deleted successfully!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }


    private static void manageDiagnoses(SessionFactory sessionFactory,
Scanner scanner) {
        // Implement diagnosis management here...
         boolean exit = false;

        while (!exit) {
            System.out.println("Diagnosis Management");
            System.out.println("1. Add Diagnosis");
            System.out.println("2. View Diagnoses");
            System.out.println("3. Update Diagnosis");
            System.out.println("4. Delete Diagnosis");
            System.out.println("5. Back to Main Menu");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
```

```
            scanner.nextLine(); // Consume newline

            switch (choice) {
               case 1:
                  addDiagnosis(sessionFactory, scanner);
                  break;
               case 2:
                  viewDiagnoses(sessionFactory);
                  break;
               case 3:
                  updateDiagnosisDetails(sessionFactory, scanner);
                  break;
               case 4:
                  deleteDiagnosis(sessionFactory, scanner);
                  break;
               case 5:
                  exit = true;
                  break;
               default:
                  System.out.println("Invalid choice. Please select a valid
option.");
            }
         }
      }
      private static void addDiagnosis(SessionFactory sessionFactory,
Scanner scanner) {
         try (Session session = sessionFactory.openSession()) {
            Transaction transaction = session.beginTransaction();


            System.out.print("Enter patient ID: ");
            long patientId = scanner.nextLong();
            scanner.nextLine(); // Consume newline

            System.out.print("Enter doctor ID: ");
            long doctorId = scanner.nextLong();
            scanner.nextLine();

            System.out.print("Enter diagnosis date (yyyy-MM-dd): ");
            String diagnosisDateStr = scanner.nextLine();
            Date diagnosisDate = parseDate(diagnosisDateStr);

            System.out.print("Enter medical condition: ");
            String medicalCondition = scanner.nextLine();
```

```
            System.out.print("Enter diagnosis notes: ");
            String notes = scanner.nextLine();

            // Create a new Diagnosis instance and set its properties
            Diagnosis diagnosis = new Diagnosis();
            diagnosis.setPatient(session.get(Patient.class, patientId));
            diagnosis.setDoctor(session.get(Doctor.class, doctorId));
            diagnosis.setDiagnosisDate(diagnosisDate);
            diagnosis.setMedicalCondition(medicalCondition);
            diagnosis.setNotes(notes);

            // Save the new diagnosis record
            session.save(diagnosis);

            transaction.commit();
            System.out.println("Diagnosis added successfully!");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private static void viewDiagnoses(SessionFactory sessionFactory) {
        try (Session session = sessionFactory.openSession()) {
            // Query to retrieve all diagnoses
            String hql = "FROM Diagnosis";
            Query<Diagnosis> query = session.createQuery(hql,
Diagnosis.class);
            List<Diagnosis> diagnoses = query.list();

            if (diagnoses.isEmpty()) {
                System.out.println("No diagnoses found.");
            } else {
                System.out.println("List of Diagnoses:");
                for (Diagnosis diagnosis : diagnoses) {
                    System.out.println("Diagnosis ID: " + diagnosis.getId());
                    System.out.println("Patient: " +
diagnosis.getPatient().getFirstName() + " " +
diagnosis.getPatient().getLastName());
                    System.out.println("Doctor: " +
diagnosis.getDoctor().getFirstName() + " " +
diagnosis.getDoctor().getLastName());
                    System.out.println("Diagnosis Date: " +
diagnosis.getDiagnosisDate());
```

```
                System.out.println("Medical Condition: " +
diagnosis.getMedicalCondition());
                System.out.println("Notes: " + diagnosis.getNotes());
                System.out.println("----------------------");
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
  private static void updateDiagnosisDetails(SessionFactory
sessionFactory, Scanner scanner) {
    try (Session session = sessionFactory.openSession()) {
        System.out.print("Enter Diagnosis ID: ");
        long diagnosisId = Long.parseLong(scanner.nextLine());

        Diagnosis diagnosis = session.get(Diagnosis.class, diagnosisId);

        if (diagnosis == null) {
            System.out.println("Diagnosis with ID " + diagnosisId + " not
found.");
            return;
        }

        System.out.println("Current Medical Condition: " +
diagnosis.getMedicalCondition());
        System.out.print("Enter New Medical Condition: ");
        String newMedicalCondition = scanner.nextLine();
        diagnosis.setMedicalCondition(newMedicalCondition);

        System.out.println("Current Notes: " + diagnosis.getNotes());
        System.out.print("Enter New Notes: ");
        String newNotes = scanner.nextLine();
        diagnosis.setNotes(newNotes);

        Transaction transaction = session.beginTransaction();
        session.update(diagnosis);
        transaction.commit();

        System.out.println("Diagnosis details updated successfully.");
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
```

```
   private static void deleteDiagnosis(SessionFactory sessionFactory,
Scanner scanner) {
      try (Session session = sessionFactory.openSession()) {
         System.out.print("Enter Diagnosis ID: ");
         long diagnosisId = Long.parseLong(scanner.nextLine());

         Diagnosis diagnosis = session.get(Diagnosis.class, diagnosisId);

         if (diagnosis == null) {
            System.out.println("Diagnosis with ID " + diagnosisId + " not
found.");
            return;
         }

         Transaction transaction = session.beginTransaction();
         session.delete(diagnosis);
         transaction.commit();

         System.out.println("Diagnosis with ID " + diagnosisId + " deleted
successfully.");
      } catch (Exception e) {
         e.printStackTrace();
      }
   }


   private static void manageBillings(SessionFactory sessionFactory,
Scanner scanner) {
      // Implement billing management here...
       while (true) {
            System.out.println("\nBilling Management Menu:");
            System.out.println("1. Add Billing");
            System.out.println("2. View Billings");
            System.out.println("3. Update Billing Details");
            System.out.println("4. Delete Billing");
            System.out.println("5. Back to Main Menu");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline character

            switch (choice) {
               case 1:
                  addBilling(sessionFactory, scanner);
```

```
                    break;
                case 2:
                    viewBillings(sessionFactory);
                    break;
                case 3:
                    updateBillingDetails(sessionFactory, scanner);
                    break;
                case 4:
                    deleteBilling(sessionFactory, scanner);
                    break;
                case 5:
                    return; // Return to main menu
                default:
                    System.out.println("Invalid choice. Please enter a valid
option.");
                }
            }
        }
    private static void addBilling(SessionFactory sessionFactory, Scanner
scanner) {
        Transaction transaction = null;

        try (Session session = sessionFactory.openSession()) {
            transaction = session.beginTransaction();

            System.out.println("Enter Patient ID: ");
            Long patientId = scanner.nextLong();

            // Retrieve the corresponding patient
            Patient patient = session.get(Patient.class, patientId);

            // Gather billing details from the user
            System.out.println("Enter Description: ");
            scanner.next();
            String description = scanner.nextLine();

            System.out.println("Enter Amount: ");
            BigDecimal amount = scanner.nextBigDecimal();

            System.out.println("Enter Billing Date (yyyy-MM-dd): ");
            String billingDateStr = scanner.next();
            Date billingDate = parseDate(billingDateStr);

            System.out.println("Enter Payment Status: ");
```

```
            String paymentStatus = scanner.next();

            Billing billing = new Billing();
            billing.setPatient(patient);
            billing.setDescription(description);
            billing.setAmount(amount);
            billing.setBillingDate(billingDate);
            billing.setPaymentStatus(paymentStatus);

            session.save(billing);

            transaction.commit();
            System.out.println("Billing added successfully!");
        } catch (Exception e) {
            if (transaction != null) {
                transaction.rollback();
            }
            e.printStackTrace();
            System.out.println("An error occurred while adding billing.");
        }
    }
    private static void viewBillings(SessionFactory sessionFactory) {
        try (Session session = sessionFactory.openSession()) {
            String hql = "FROM Billing";
            Query<Billing> query = session.createQuery(hql, Billing.class);

            List<Billing> billings = query.list();

            System.out.println("Billing Entries:");
            for (Billing billing : billings) {
                System.out.println("Billing ID: " + billing.getId());
                System.out.println("Patient Name: " +
billing.getPatient().getFirstName() + " " +
billing.getPatient().getLastName());
                System.out.println("Description: " + billing.getDescription());
                System.out.println("Amount: " + billing.getAmount());
                System.out.println("Billing Date: " + billing.getBillingDate());
                System.out.println("Payment Status: " +
billing.getPaymentStatus());
                System.out.println("-------------------------");
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("An error occurred while viewing billings.");
```

```
        }
    }
    private static void updateBillingDetails(SessionFactory sessionFactory,
Scanner scanner) {
        try (Session session = sessionFactory.openSession()) {
            System.out.print("Enter Billing ID to update: ");
            long billingId = scanner.nextLong();
            scanner.nextLine(); // Consume the newline character

            Billing billing = session.get(Billing.class, billingId);

            if (billing != null) {
                System.out.println("Updating billing details for Billing ID: " +
billingId);

                System.out.print("Enter new Description: ");
                String newDescription = scanner.nextLine();
                billing.setDescription(newDescription);

                System.out.print("Enter new Amount: ");
                BigDecimal newAmount = scanner.nextBigDecimal();
                billing.setAmount(newAmount);

                System.out.print("Enter new Payment Status: ");
                String newPaymentStatus = scanner.next();
                billing.setPaymentStatus(newPaymentStatus);

                Transaction transaction = session.beginTransaction();
                session.update(billing);
                transaction.commit();

                System.out.println("Billing details updated successfully.");
            } else {
                System.out.println("Billing with ID " + billingId + " not
found.");
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("An error occurred while updating billing
details.");
        }
    }
    private static void deleteBilling(SessionFactory sessionFactory, Scanner
scanner) {
```

```
        try (Session session = sessionFactory.openSession()) {
            System.out.print("Enter Billing ID to delete: ");
            long billingId = scanner.nextLong();
            scanner.nextLine(); // Consume the newline character

            Billing billing = session.get(Billing.class, billingId);

            if (billing != null) {
                Transaction transaction = session.beginTransaction();
                session.delete(billing);
                transaction.commit();

                System.out.println("Billing with ID " + billingId + " deleted
successfully.");
            } else {
                System.out.println("Billing with ID " + billingId + " not
found.");
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("An error occurred while deleting billing.");
        }
    }
    private static SessionFactory buildSessionFactory() {
        StandardServiceRegistry serviceRegistry = new
StandardServiceRegistryBuilder()
                .configure() // Load hibernate.cfg.xml by default
                .build();

        return new
MetadataSources(serviceRegistry).buildMetadata().buildSessionFactory()
;
    }

    private static boolean checkLogin(String username, String password,
SessionFactory sessionFactory) {
        try (Session session = sessionFactory.openSession()) {
            String hql = "FROM Login WHERE username = :username AND
password = :password";
            Query<Login> query = session.createQuery(hql, Login.class);
            query.setParameter("username", username);
            query.setParameter("password", password);

            List<Login> result = query.list();
```

```
            return !result.isEmpty();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return false;
    }

    private static Date parseDate(String dateString) throws ParseException
{
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-
dd");
        return dateFormat.parse(dateString);
    }
}
```

Patient :

```
package com.patient.information.system;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.JoinColumn;
import java.util.Date;

@Entity
public class Appointment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "patient_id")
    private Patient patientId;

    @ManyToOne
    @JoinColumn(name = "doctor_id")
    private Doctor doctorId;

    private Date appointmentDate;
    private String status;
    private String notes;
```

```
// Constructors
public Appointment() {
}

// Getters and Setters
public Long getId() {
   return id;
}

public void setId(Long id) {
   this.id = id;
}

public Patient getPatient() {
   return patientId;
}

public void setPatient(Patient patientId) {
   this.patientId = patientId;
}

public Doctor getDoctor() {
   return doctorId;
}

public void setDoctor(Doctor doctorId) {
   this.doctorId = doctorId;
}

public Date getAppointmentDate() {
   return appointmentDate;
}

public void setAppointmentDate(Date appointmentDate) {
   this.appointmentDate = appointmentDate;
}

public String getStatus() {
   return status;
}

public void setStatus(String status) {
   this.status = status;
```

```
    }

    public String getNotes() {
        return notes;
    }

    public void setNotes(String notes) {
        this.notes = notes;
    }
}
```

Appointement :

```
package com.patient.information.system;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.JoinColumn;
import java.math.BigDecimal;
import java.util.Date;

@Entity
public class Billing {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "patient_id")
    private Patient patientId;

    private Date billingDate;
    private String description;
    private BigDecimal amount;
    private String paymentStatus;
    private Date paymentDate;

    // Constructors
    public Billing() {
    }
```

```
    // Getters and Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Patient getPatient() {
        return patientId;
    }

    public void setPatient(Patient patientId) {
        this.patientId = patientId;
    }

    public Date getBillingDate() {
        return billingDate;
    }

    public void setBillingDate(Date billingDate) {
        this.billingDate = billingDate;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public BigDecimal getAmount() {
        return amount;
    }

    public void setAmount(BigDecimal amount) {
        this.amount = amount;
    }

    public String getPaymentStatus() {
        return paymentStatus;
```

```
    }

    public void setPaymentStatus(String paymentStatus) {
        this.paymentStatus = paymentStatus;
    }

    public Date getPaymentDate() {
        return paymentDate;
    }

    public void setPaymentDate(Date paymentDate) {
        this.paymentDate = paymentDate;
    }
}
```

Billing :

```
package com.patient.information.system;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.JoinColumn;
import java.util.Date;

@Entity
public class Diagnosis {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "patient_id")
    private Patient patientId;

    @ManyToOne
    @JoinColumn(name = "doctor_id")
    private Doctor doctorId;

    private Date diagnosisDate;
    private String medicalCondition;
    private String notes;
```

```
// Constructors
public Diagnosis() {
}

// Getters and Setters
public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public Patient getPatient() {
    return patientId;
}

public void setPatient(Patient patientId) {
    this.patientId = patientId;
}

public Doctor getDoctor() {
    return doctorId;
}

public void setDoctor(Doctor doctorId) {
    this.doctorId = doctorId;
}

public Date getDiagnosisDate() {
    return diagnosisDate;
}

public void setDiagnosisDate(Date diagnosisDate) {
    this.diagnosisDate = diagnosisDate;
}

public String getMedicalCondition() {
    return medicalCondition;
}

public void setMedicalCondition(String medicalCondition) {
    this.medicalCondition = medicalCondition;
}
```

```
    public String getNotes() {
        return notes;
    }

    public void setNotes(String notes) {
        this.notes = notes;
    }
}
```

Diagonsis :

```
package com.patient.information.system;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity

public class Doctor {
        @Id
          @GeneratedValue(strategy = GenerationType.IDENTITY)
          private Long id;

          private String firstName;
          private String lastName;
          private String specialization;
          private String email;
          private String phoneNumber;
          private String address;

          // Constructors
          public Doctor() {
          }

          // Getters and Setters
          public Long getId() {
              return id;
          }

          public void setId(Long id) {
              this.id = id;
```

```
}

public String getFirstName() {
   return firstName;
}

public void setFirstName(String firstName) {
   this.firstName = firstName;
}

public String getLastName() {
   return lastName;
}

public void setLastName(String lastName) {
   this.lastName = lastName;
}

public String getSpecialization() {
   return specialization;
}

public void setSpecialization(String specialization) {
   this.specialization = specialization;
}

public String getEmail() {
   return email;
}

public void setEmail(String email) {
   this.email = email;
}

public String getPhoneNumber() {
   return phoneNumber;
}

public void setPhoneNumber(String phoneNumber) {
   this.phoneNumber = phoneNumber;
}

public String getAddress() {
   return address;
```

```
        }

        public void setAddress(String address) {
           this.address = address;
        }

}
```

Doctor :

```
package com.patient.information.system;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Login {
       @Id
         @GeneratedValue(strategy = GenerationType.IDENTITY)
         private Long id;

         private String username;
         private String password;

         public Login() {
         }

         public Login(String username, String password) {
            this.username = username;
            this.password = password;
         }

         public Long getId() {
            return id;
         }

         public void setId(Long id) {
            this.id = id;
         }

         public String getUsername() {
            return username;
         }
```

```
        public void setUsername(String username) {
           this.username = username;
        }

        public String getPassword() {
           return password;
        }

        public void setPassword(String password) {
           this.password = password;
        }

}
```

Login :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
   <session-factory>
      <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
      <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
      <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306services/pis</prop
erty>
      <property name="hibernate.connection.username">root</property>
      <property name="hibernate.connection.password">B@l@ji007</property>
      <property name="hbm2ddl.auto">update</property>

     <mapping class="com.patient.information.system.Login"/>
     <mapping class="com.patient.information.system.Patient"/>
     <mapping class="com.patient.information.system.Doctor"/>
     <mapping class="com.patient.information.system.Appointment"/>
     <mapping class="com.patient.information.system.Diagnosis"/>
     <mapping class="com.patient.information.system.Billing"/>
   </session-factory>
</hibernate-configuration>
```

CONCLUSION

Creating a Patient Management System using Hibernate for a medical environment has immense significance in streamlining operations and enhancing patient care. This project encompasses various modules like patient, doctor, appointment, diagnosis, billing, and an admin login, each playing a pivotal role in managing different aspects of healthcare delivery. Here is a comprehensive conclusion for such a system:

Conclusion:
These enhancements can elevate the Patient Management System, making it more comprehensive, user-friendly, and efficient in providing healthcare services. Prioritizing these enhancements could revolutionize the system's capabilities, benefiting both healthcare providers and patients while ensuring compliance and security of sensitive medical data.

FUTURE ENHANCEMENTS

Creating a Patient Management System using Hibernate for a medical environment has immense significance in streamlining operations and enhancing patient care. This project encompasses various modules like patient, doctor, appointment, diagnosis, billing, and an admin login, each playing a pivotal role in managing different aspects of healthcare delivery. Here is a comprehensive conclusion for such a system:

The patient module within the system serves as a central repository for storing patient information. Through this module, patient demographics, medical history, prescriptions, and test results can be efficiently managed, ensuring quick access to critical information for healthcare providers. The utilization of Hibernate in mapping these entities to the database tables allows for seamless data retrieval, manipulation, and storage.

Simultaneously, the doctor module empowers healthcare practitioners to access patient records, prescribe medications, schedule appointments, and record diagnosis details. The efficient integration of doctor-related functionalities with the database ensures real-time updates and improved communication between healthcare professionals and patients.

Appointment scheduling and management play a vital role in ensuring optimal patient care. By leveraging Hibernate in the appointment module, the system enables easy booking, rescheduling, and cancellation of appointments. Moreover, it facilitates the tracking of appointment history, allowing for better resource allocation and reducing patient wait times.

Diagnosis and treatment information are critical components in healthcare management. The system's diagnosis module facilitates the recording and retrieval of diagnosis details, treatment plans, and follow-up procedures. Through Hibernate's efficient data handling capabilities, this information is securely stored and easily accessible, enhancing the continuity of care.

Billing is an essential aspect of healthcare administration. The billing module in the system automates the billing process, generating invoices, recording payments, and managing financial transactions. Hibernate's object-relational mapping ensures data integrity and accuracy, reducing errors in billing and financial reporting.

The centralized admin login provides a secure gateway for system management. Admin privileges encompass user management, database maintenance, and system configuration. The use of a single admin login streamlines system administration while ensuring data security and access control.

In conclusion, the implementation of a Patient Management System using Hibernate technology offers a comprehensive solution for effective healthcare management. By integrating various modules such as patient, doctor, appointment, diagnosis, billing, and admin access, the system optimizes patient care,

enhances operational efficiency, and ensures data security and integrity. Its ability to handle complex data relationships, ease of maintenance, and scalability makes it an invaluable tool in modern healthcare settings. Moving forward, continuous improvement and adaptation to evolving healthcare needs will further enhance the system's functionality and usability, ultimately benefiting both healthcare providers and patients alike.
: Future enhancements for a Patient Management System (PMS) can be crucial in improving its functionality, usability, and efficiency. Here are potential enhancements to consider:

1. *Telemedicine Integration (Remote Consultations)*
   Integrate telemedicine features allowing doctors to conduct remote consultations, video conferences, or chat sessions with patients. This feature would expand access to healthcare services, especially for patients unable to visit in person.

2. *Enhanced Appointment Scheduling*
   Implement an advanced scheduling system that includes recurring appointments, waitlist management, reminders via SMS or email, and real-time updates on doctor availability. This would streamline the scheduling process and reduce no-shows.

3. *Electronic Health Records (EHR)*
   Develop a comprehensive EHR system to store and manage patient medical records securely. Incorporate features like historical data access, lab results, treatment plans, and medication history to facilitate better decision-making for doctors.

4. *Mobile Application*
   Create a user-friendly mobile app for patients to schedule appointments, access their medical records, receive notifications, and communicate with healthcare providers. This mobile platform could enhance patient engagement and convenience.

5. *Analytics and Reporting*
   Introduce analytics tools to analyze data trends, patient outcomes, and resource utilization. Generate insightful reports for administrators to optimize workflows, resource allocation, and identify areas for improvement.

6. *Billing and Insurance Integration*
   Integrate with insurance systems for streamlined billing processes, including claims submission, verification, and payment tracking. This enhancement can reduce administrative errors and improve financial operations.

7. *Patient Portal for Self-Service*
   Develop a patient portal enabling patients to update personal information, view upcoming appointments, access educational materials, and securely communicate with their healthcare providers.

8. *Enhanced Security Measures*
   Strengthen the system's security protocols to comply with healthcare data regulations (such as HIPAA). Implement encryption, role-based access controls, and regular security audits to safeguard patient data.

9. *Machine Learning for Decision Support*
   Implement machine learning algorithms to assist doctors in diagnosing diseases, predicting treatment outcomes, or suggesting personalized treatment plans based on historical data and medical literature.

10. *Integration with Wearable Devices*
   Connect with wearable health devices (e.g., fitness trackers, smartwatches) to collect real-time patient health data. This integration can provide additional insights for doctors and enable proactive healthcare management.

BIBLIOGRAPHY

- **FOR SQL**
  www.msdn.microsoft.com
- **Java**
  https://www.oracle.com/java/technologies/persistence-jsp.html

- **Hibernate**
  https://youtu.be/VtCz0oPtfG0?si=7ys73zgUOPW5GOQG
  https://www.javatpoint.com/hibernate-tutorial