JAVA PROGRAMMING
Course Code - R1UC304C
SEM-III (Session 2025-26)

HEALTHCARE MANAGEMENT SYSTEM

TEAM MEMBERS:

DIVYANSHU AGRAWAL (24SCSE1410118)
VINAY TIWARI (24SCSE1180103)
ASHISH PRATAP (24SCSE1410316)

INTRODUCTION

- Healthcare requires accuracy, safety, and timely service for patient trust.
- Hospitals must manage patient records, appointments, doctor schedules, and reports without errors.
- Increasing patient load makes manual handling slow, confusing, and error-prone.
- A computerized system ensures faster workflow and reduces staff workload.

Project Purpose

- To build a Healthcare Management System that manages:
  - Patients
  - Doctors
  - Appointments
  - Admin tasks
- Provides an organized, fast, and error-free workflow using a simple Java GUI.

PROBLEM STATEMENT

Core Challenges in Healthcare Management

- Hospitals must handle large patient volumes, doctor scheduling, and multiple administrative tasks quickly and accurately.
- Manual record-keeping leads to errors, delays, lost data, and poor patient experience.
- There is a need for a simple, safe, and fast desktop system that can store and manage crucial healthcare information.

System-Specific Problems to Solve

- Administrators require a tool to register new patients, manage doctor availability, and maintain accurate medical and appointment data.
- Adding patients requires capturing personal info, medical history, contact details, and assigned doctor/department.
- The system must support secure login, organized data storage, and easy retrieval of records.

KEY BENEFITS

- Faster & Error-Free Operations
- Smooth Appointment Scheduling
- Smart Data Management
- Reliable Database Connectivity
- Improved Workflow for Admin
- Modern & User-Friendly UI
- Multithreading Support (Where Needed)

OOPS IMPLEMENTATION

INHERITANCE

- User is the base class containing common fields: name, email, password, role.
- Admin, Doctor, Patient extend User, removing duplicate code.
- Shared logic (login, identity, account details) stays in one place → clean structure.
- Improves maintenance and supports future expansion.

POLYMORPHISM

- Method overriding for different user dashboards (Admin vs Doctor vs Patient).
- manageAppointments() behaves differently based on logged-in role.
- Method overloading used in appointment booking (book by date OR date + doctor).
- Ensures flexible and role-specific behavior.

INTERFACES

- DAO Interface: AppointmentDAO defines contract for all DB operations.
- DAOImpl class: AppointmentDAOImpl implements CRUD + filters.
- Possible functional interfaces like Loginable, Schedulable, RecordManageable (conceptually fit the project).
- Supports modular, flexible, scalable architecture.

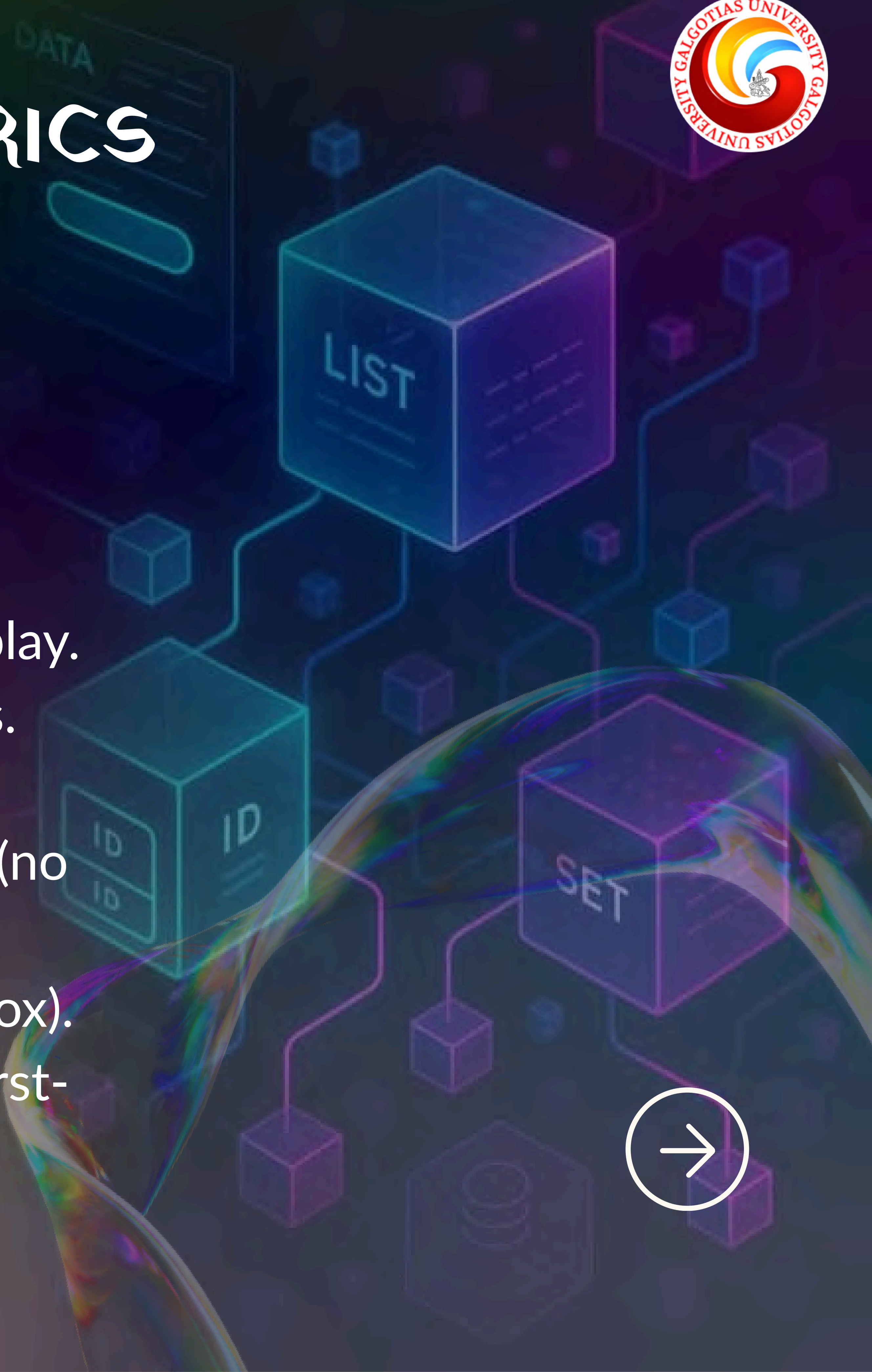
EXCEPTION HANDLING

- Proper try–catch blocks around database operations & invalid inputs.
- Appointment booking errors handled safely (invalid date/time).
- Custom exceptions possible: InvalidAppointmentException, UserNotFoundException.
- Prevents app crashes & ensures smooth user experience.

COLLECTIONS & GENERICS

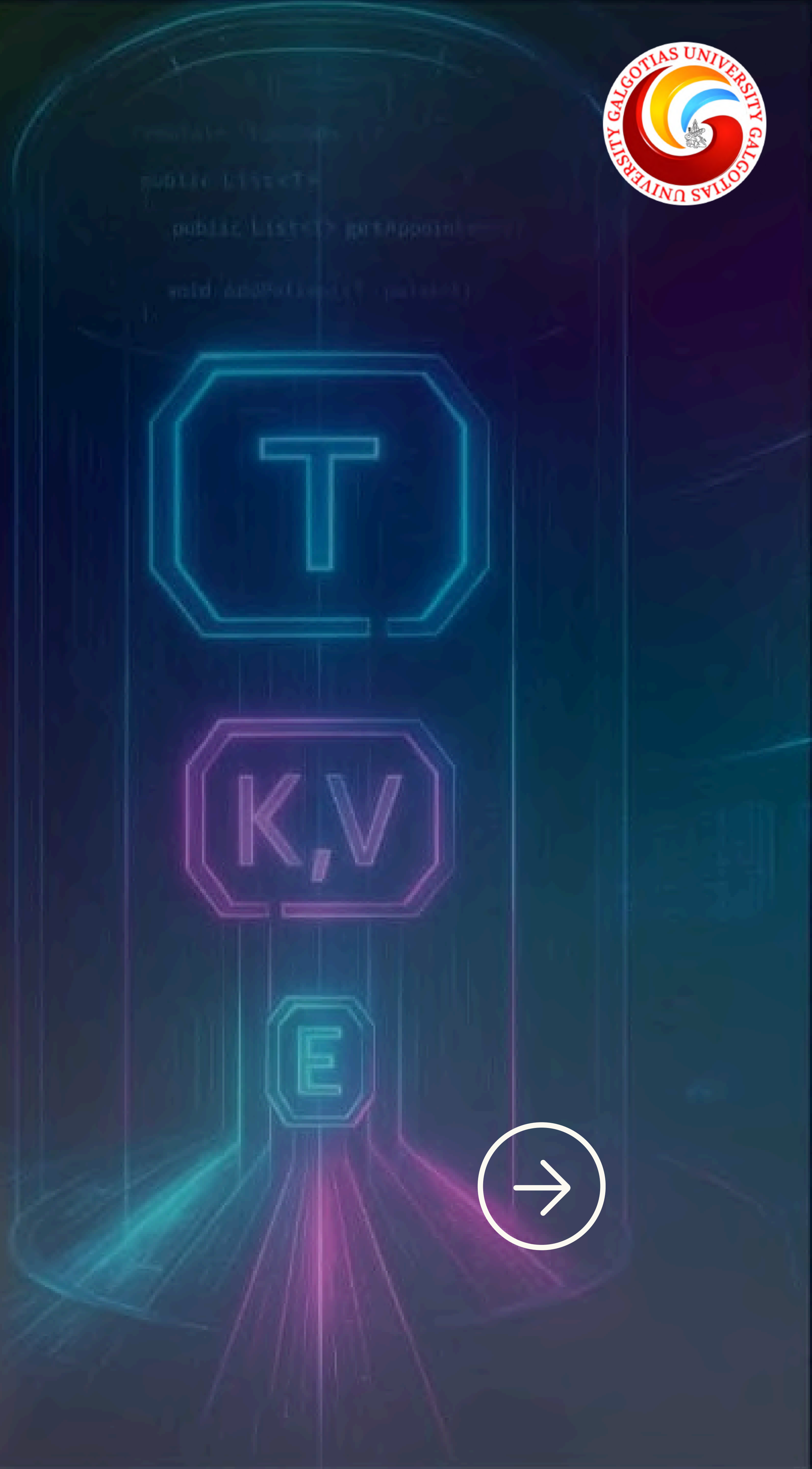
Collections Usage (Lists, Maps, Sets)

- ArrayList stores dynamic lists of Patients, Doctors, Users, and Appointments.
- Supports easy add / update / delete operations in all management frames.
- Used when loading data from database into JTable before display.
- HashMap maps User IDs to User objects for fast login lookups.
- Provides O(1) access time while verifying credentials.
- HashSet can maintain unique departments, specialties, or IDs (no duplicates).
- Ensures clean, non-repeated data inside dropdowns (ComboBox).
- LinkedList useful for appointment queues (FIFO: first-come-first-served).



Generics Usage (Type-Safe Collections)

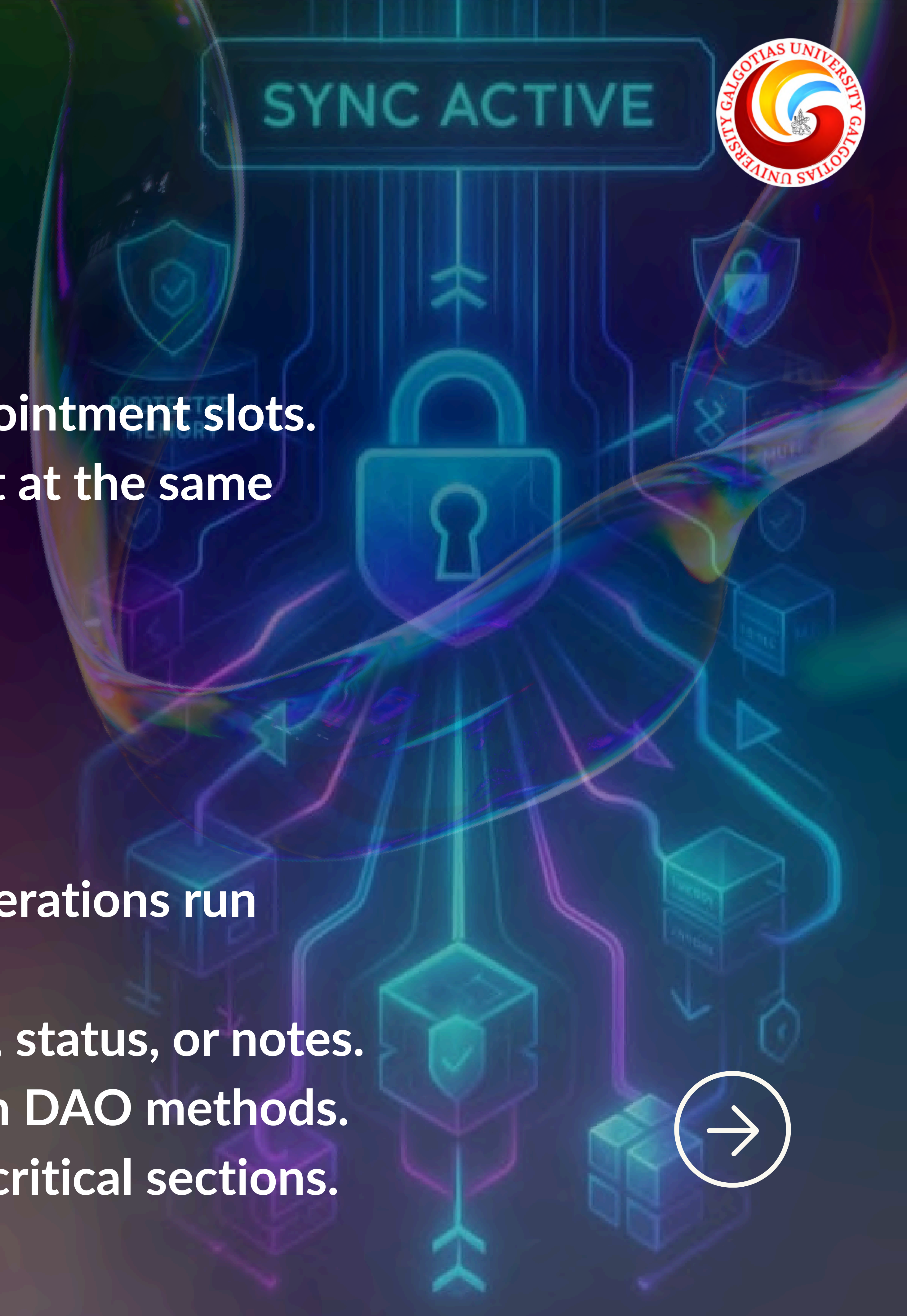
- Project uses parameterized collections such as:
  - ArrayList<Appointment>
  - ArrayList<Patient>
  - HashMap<Integer, Appointment>
  - HashSet<String>
- Generics ensure type safety, preventing wrong data insertion.
- Avoids unnecessary type-casting → improves readability and reduces errors.
- Makes DAO methods consistent, e.g., List<Appointment> getAllAppointments().
- Enhances maintainability and reduces runtime exceptions.



MULTITHREADING & SYNCHRONIZATION

Multithreading Usage

- Background thread used to load doctor/patient lists without freezing the UI.
- Appointment data loads inside a separate worker thread for smoother experience.
- GUI remains responsive even when database operations are running.
- Background thread refreshes appointment table periodically.
- Used in long-running operations inside ManageAppointmentsFrame using Runnable.



Synchronization Usage (Safe Shared Data Access)

- Synchronized block/method ensures safe booking of appointment slots.
- Prevents two threads from writing the same appointment at the same time.
- Protects shared collections like:
  - ArrayList<Appointment>
  - ArrayList<Patient>
  - HashMap<Integer, User>
- Ensures consistent patient/doctor data when multiple operations run together.
- Avoids data corruption when updating appointment time, status, or notes.
- Only one thread at a time can perform write operations in DAO methods.
- Guarantees atomicity of insert/update operations inside critical sections.

CLASSES FOR THE DATABASE OPERATIONS

DBConnection Class (util.DBConnection.java)

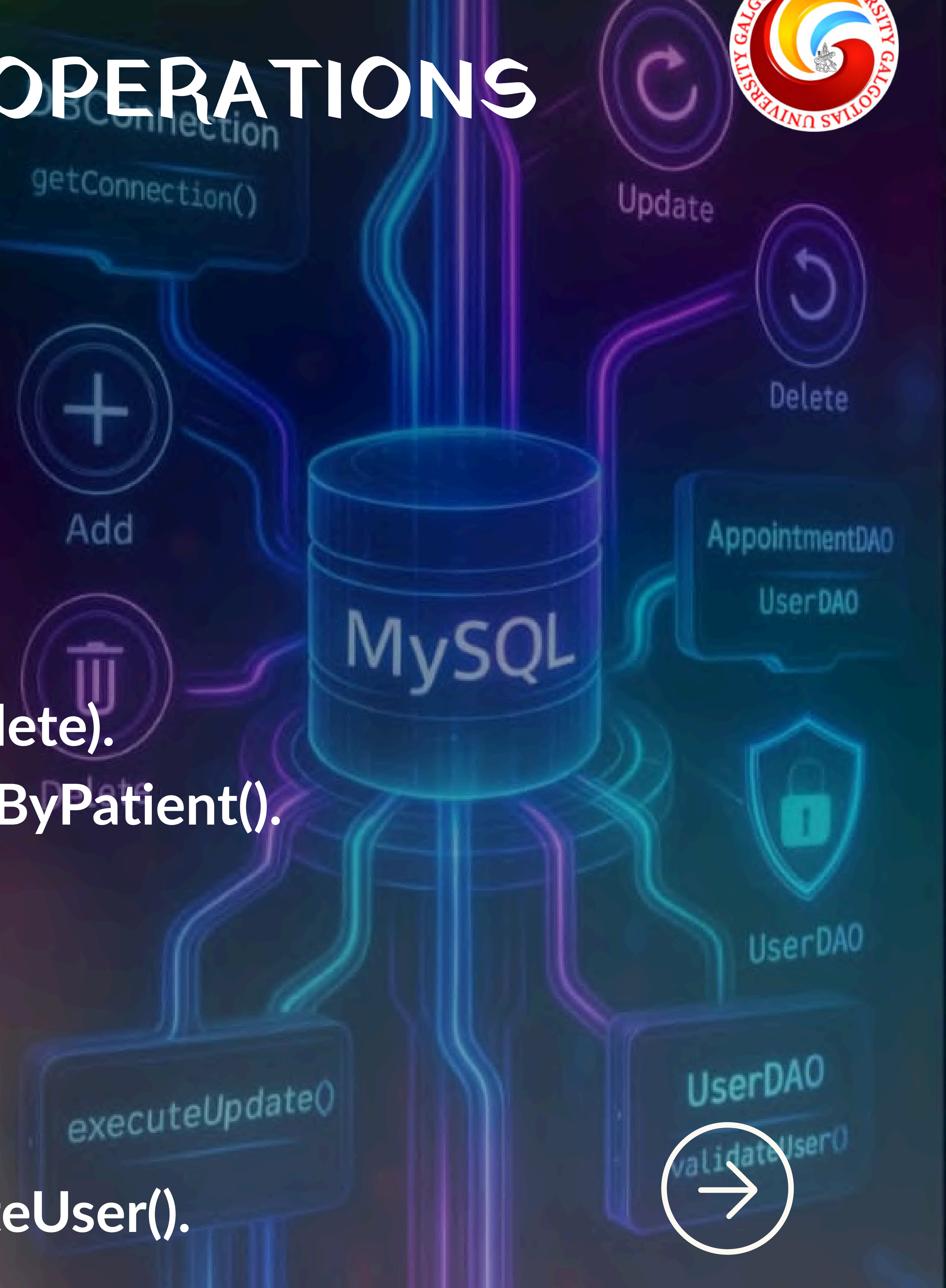
- Central class for opening a single MySQL connection.
- getConnection() returns a live connection for all DAOs.
- Uses PreparedStatement for safe, secure queries.

AppointmentDAO + AppointmentDAOImpl

- Handles all appointment CRUD operations (Add, Update, Delete).
- Supports getAppointmentsByDoctor() and getAppointmentsByPatient().
- Uses Timestamp conversion between Java and MySQL.

UserDAO + UserDAOImpl

- Used for Login & Admin user management.
- Methods: getUserByEmail(), createUser(), updateUser(), deleteUser().
- Handles bcrypt password validation (security).



PatientDAO (Inside Patient management logic)

- Loads patient details along with linked user information (JOIN query).
- Used in ManagePatientsFrame to fetch/update/insert patient data.

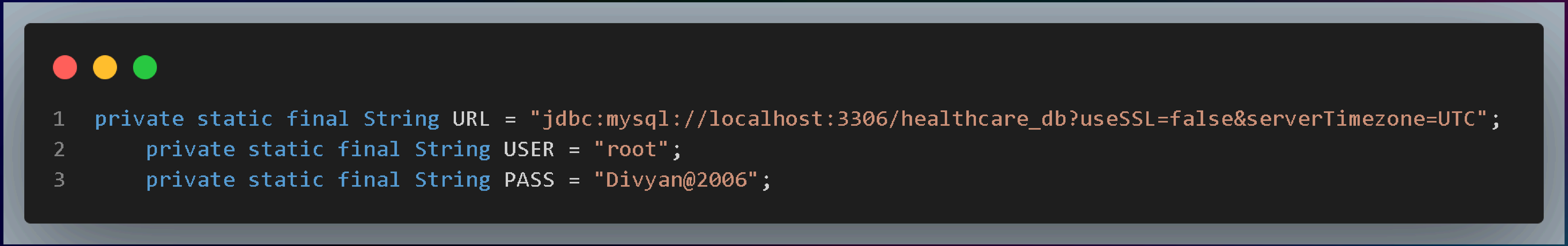
DoctorDAO (Inside Doctor management logic)

- Fetches all doctors with their specialization using JOIN.
- Used to populate dropdowns and load data dynamically.

Model Classes (Appointment, Patient, User)

- Each model represents a database table (1:1 mapping).
- Used to pass structured data between UI → DAO → Database.

DATABASE CONNECTIVITY (JDBC)



```java
private static final String URL = "jdbc:mysql://localhost:3306/healthcare_db?useSSL=false&serverTimezone=UTC";
    private static final String USER = "root";
    private static final String PASS = "Divyan@2006";
```

Use of PreparedStatement

- Prevents SQL Injection, making login and appointments safer.
- Prepared statements bind values using setInt(), setString(), setTimestamp().

CRUD Operations Implemented

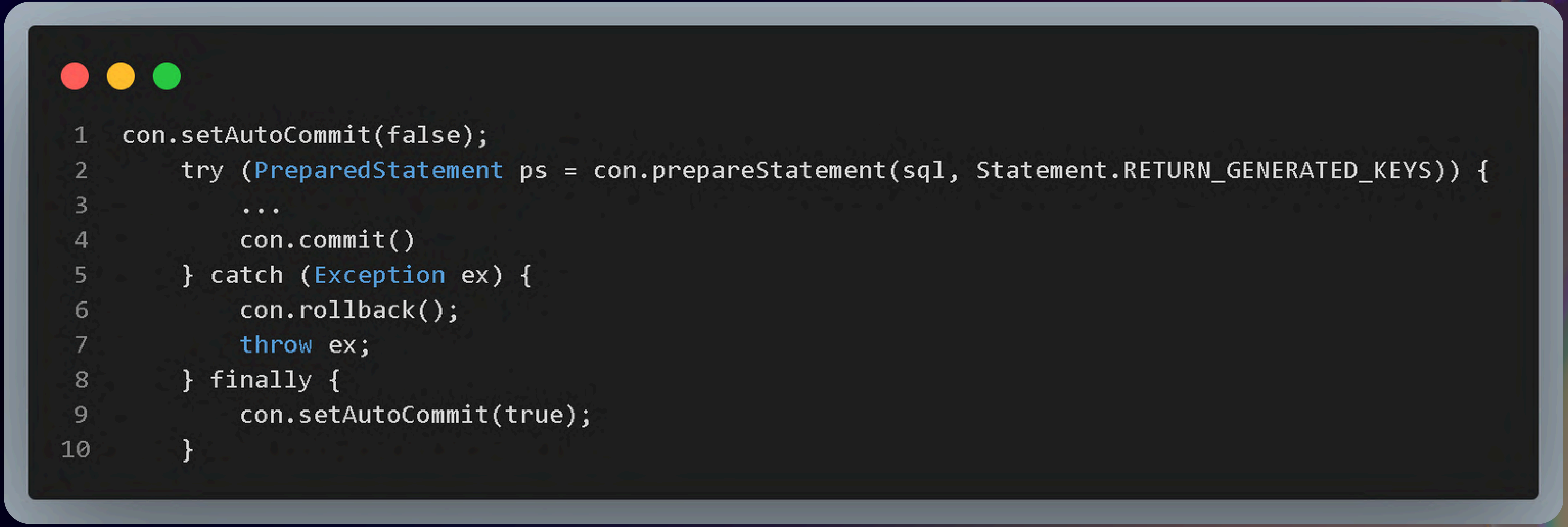
- INSERT → Add doctors, patients, appointments.
- UPDATE → Modify appointment details, patient info, doctor info.
- DELETE → Remove users or appointments.
- SELECT → Fetch joined records (Users + Doctors/Patients).
- JOIN → queries used to retrieve meaningful combined data.

IMPLEMENT JDBC FOR DATABASE CONNECTIVITY

Transaction Usage

- Transactions are used during appointment creation, updating, and deletion.
- If any step fails, rollback() prevents partial changes.
- commit() saves all changes only when operations succeed.

createAppointment(Appointment a)

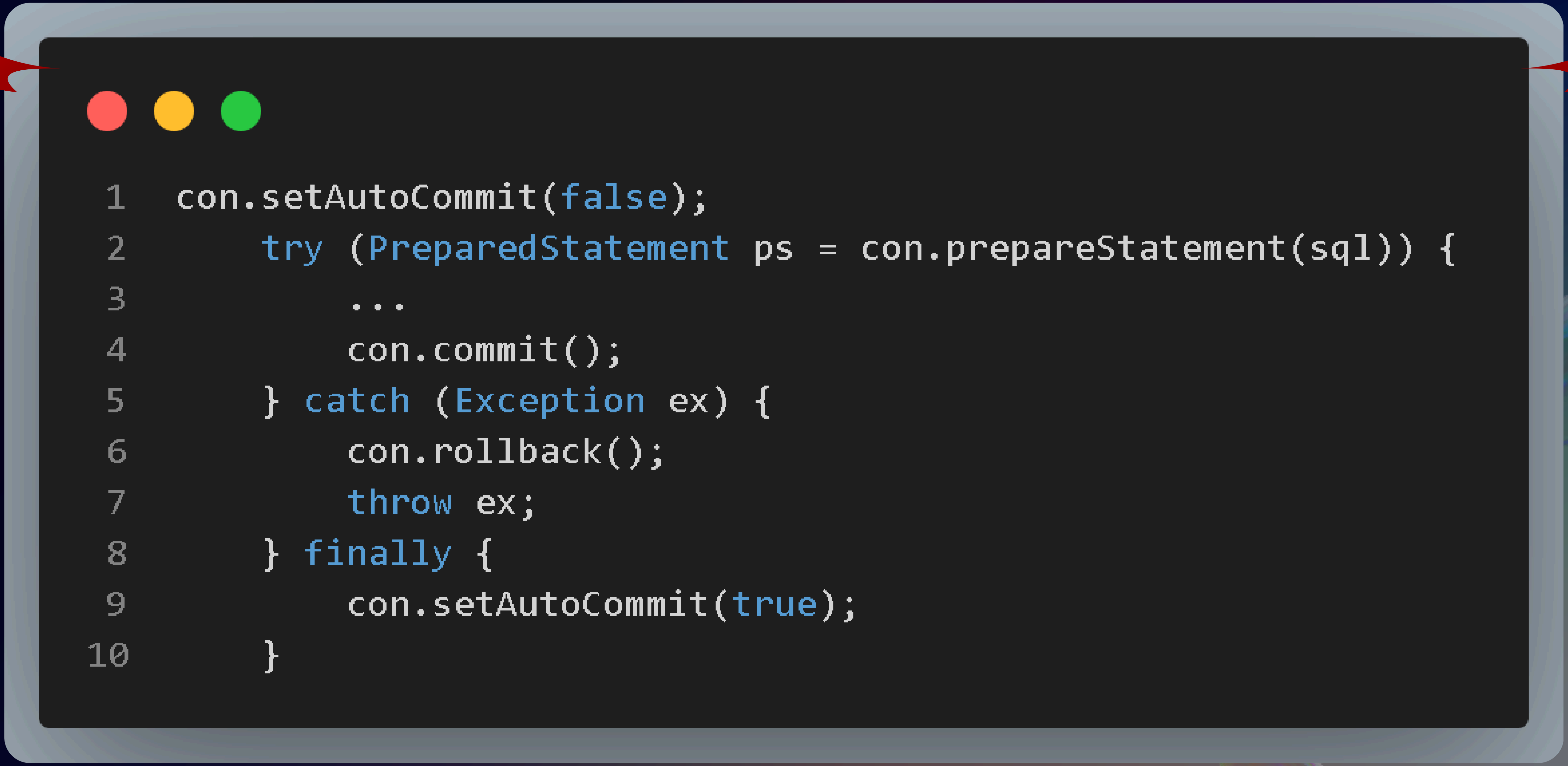


```java
con.setAutoCommit(false);
    try (PreparedStatement ps = con.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
        ...
        con.commit()
    } catch (Exception ex) {
        con.rollback();
        throw ex;
    } finally {
        con.setAutoCommit(true);
    }
```

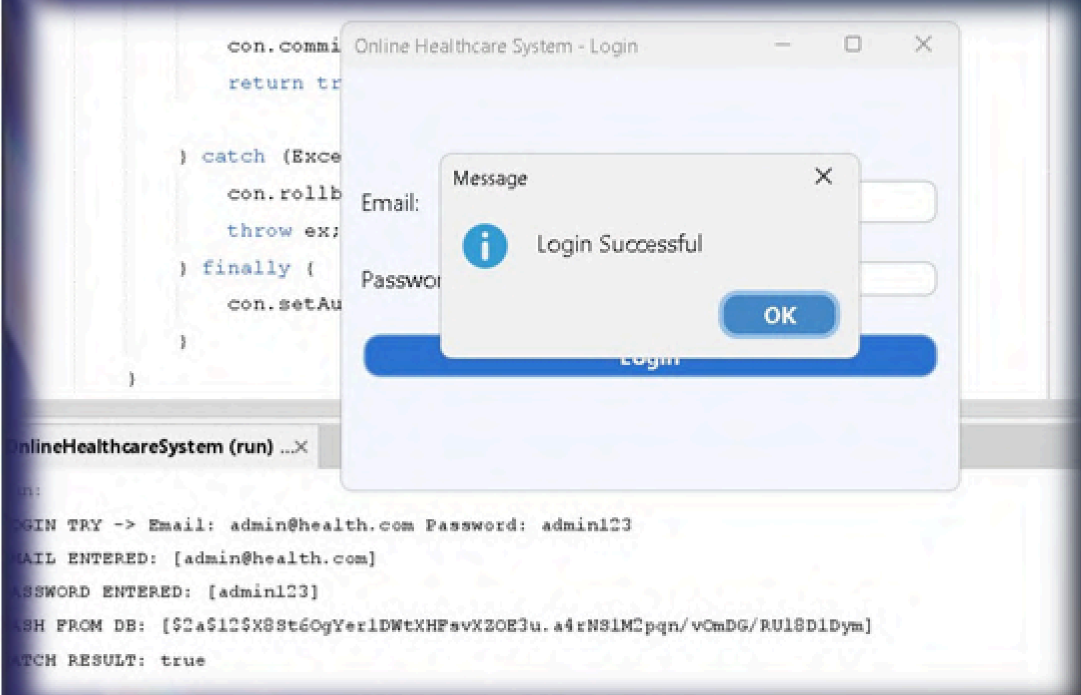
SAME CODE STRUCTURE ≠ SAME PURPOSE

deleteAppointment(int id)

updateAppointment(Appointment a)



```java
con.setAutoCommit(false);
    try (PreparedStatement ps = con.prepareStatement(sql)) {
        ...
        con.commit();
    } catch (Exception ex) {
        con.rollback();
        throw ex;
    } finally {
        con.setAutoCommit(true);
    }
```



Online Healthcare System - Login

Email:

Password

Message

Login Successful

OK

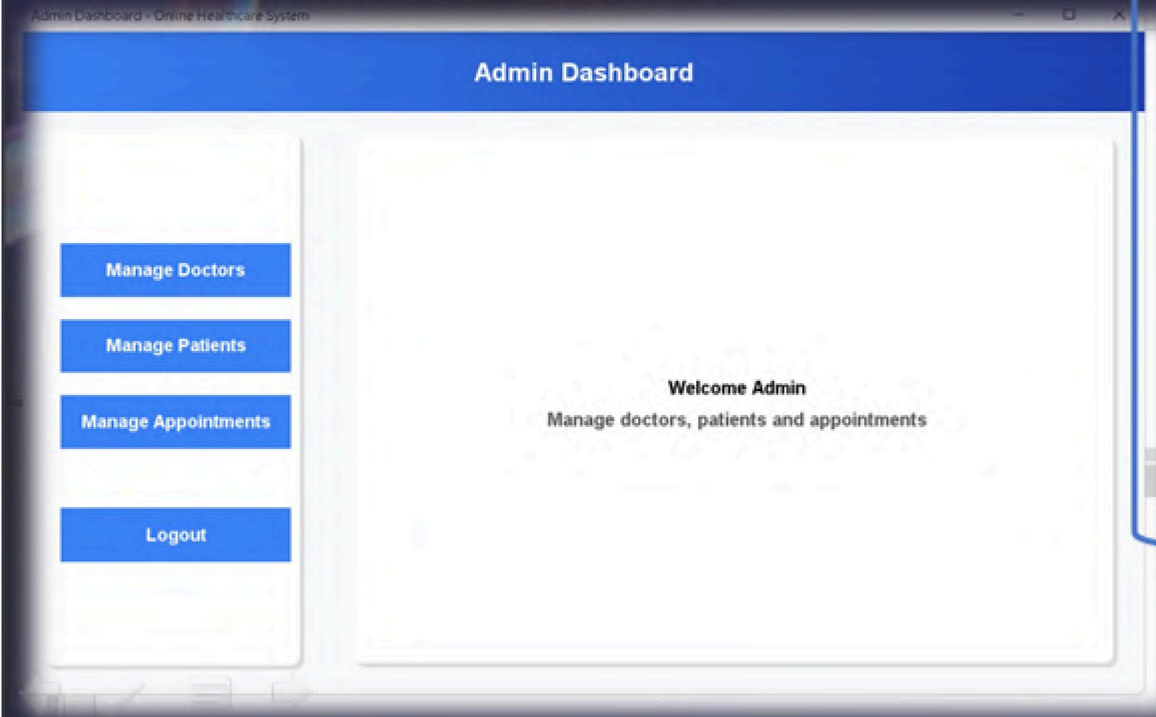
```
con.commi
return tr
} catch (Exce
con.rollb
throw ex;
} finally {
con.setAu
}
}
```

OnlineHealthcareSystem (run)

```
GIN TRY -> Email: admin@health.com Password: admin123
AIL ENTERED: [admin@health.com]
SSWORD ENTERED: [admin123]
SH FROM DB: [$2a$12$X8St6OgYer1DWtXHFsvXZOE3u.a4rNS1M2pqn/vOmDG/RU1SD1Dym]
TCH RESULT: true
```

Login Frame

Admin Dashboard



Admin Dashboard

Manage Doctors

Manage Patients

Manage Appointments

Logout

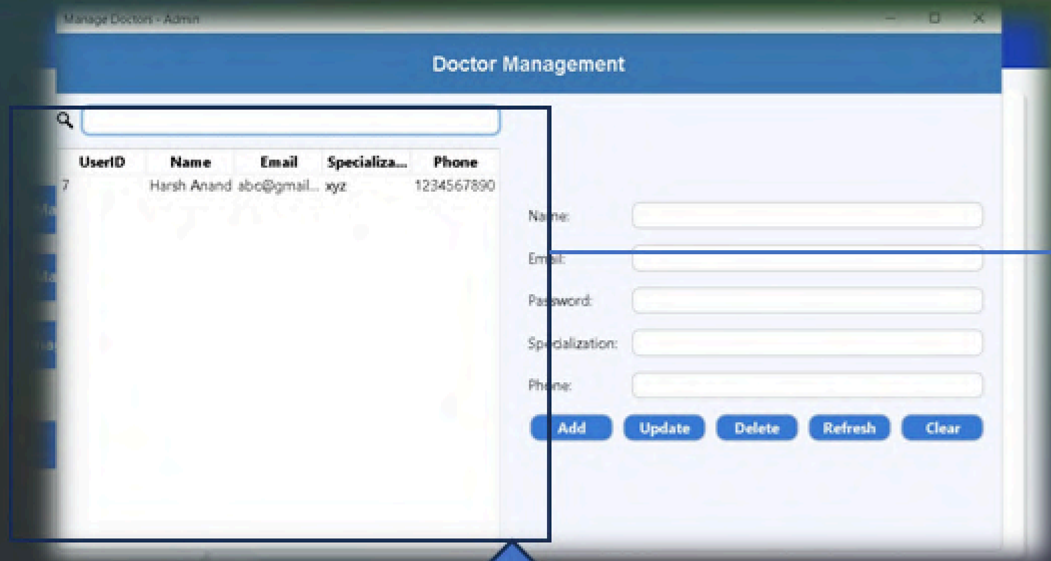
Welcome Admin

Manage doctors, patients and appointments

Doctor's Dashboard

Appointment's Dashboard

Patient's Dashboard



Doctor Management

UserID	Name	Email	Specializa...	Phone
7	Harsh Anand	abc@gmail...	xyz	1234567890

Name:

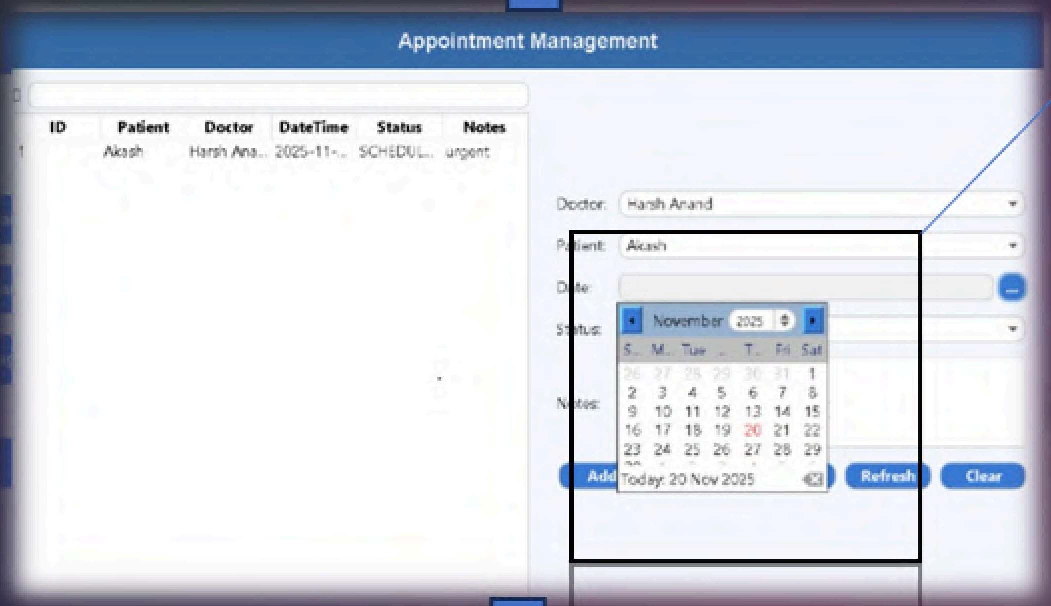
Email:

Password:

Specialization:

Phone:

Record Table



Appointment Management

ID	Patient	Doctor	DateTime	Status	Notes
1	Akash	Harsh Ana...	2025-11-...	SCHEDUL...	urgent

Doctor: Harsh Anand

Patient: Akash

Date:

Status:

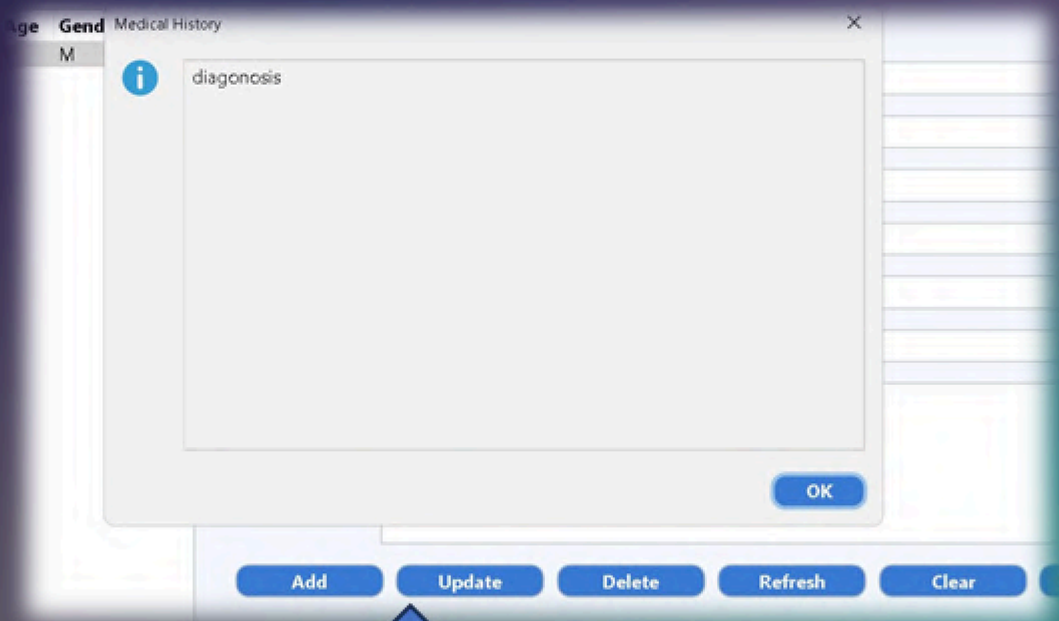
Notes:

November 2025

Today: 20 Nov 2025

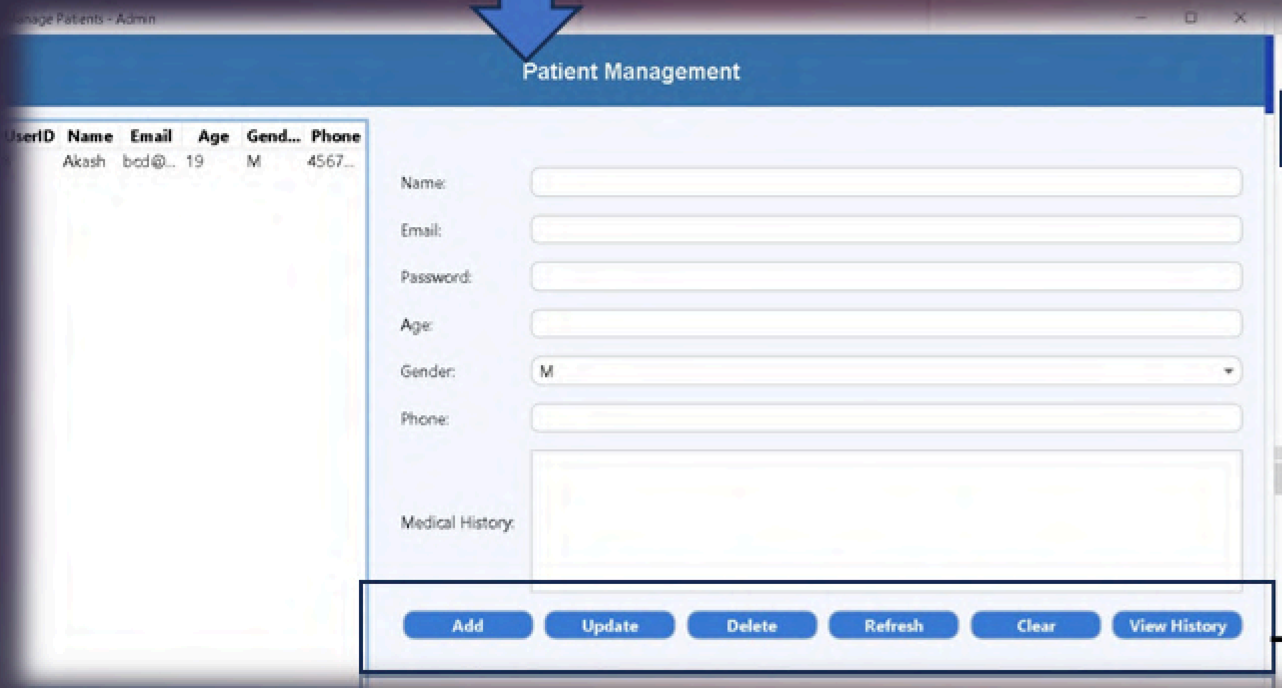
Calender

Manage History Patient



Medical History

diagonosis



Patient Management

UserID	Name	Email	Age	Gend...	Phone
	Akash	bcd@...	19	M	4567...

Name:

Email:

Password:

Age:

Gender: M

Phone:

Medical History:

ADD
UPDATE
DELETE
REFRESH
CLEAR

THANKS