MEDICAL CLINIC MANAGEMENT SYSTEM

INTRODUCTION TO MEDICAL CLINIC MANAGEMENT SYSTEM

The Medical Clinic Management System streamlines clinic operations by improving patient experience, healthcare provider efficiency, and record-keeping. Key features include patient scheduling, doctor availability management, medical records, and secure user logins. It uses design patterns (Singleton, Factory, Builder, Prototype) for maintainability, flexibility, and scalability. The system is powered by a relational database to manage information about patients, doctors, appointments, and medical records. This solution enhances clinic operations while ensuring data security and efficiency.

DESIGN PATTERNS

A DESIGN PATTERN IS A REUSABLE SOLUTION TO A COMMONLY OCCURRING PROBLEM IN SOFTWARE DESIGN.

WHY DESIGN PATTERNS?

IMPROVES CODE REUSABILITY AND MAINTAINABILITY.

HELPS IN SOLVING COMMON ISSUES THAT ARISE IN SOFTWARE DEVELOPMENT.

MAKES THE DESIGN OF SOFTWARE MORE FLEXIBLE AND EASIER TO UNDERSTAND.

1	2	3	4	5
Singleton	Factory	Builder	Prototype	Adapter

1. SINGLETON PATTERN

- Definition:
- Ensures that a class has only one instance and provides a global access point to it.
- Use Case:
- When a single shared resource or object is needed across the entire application, such as a database connection, logging service, or a configuration manager.
- Key Features:
  - Only one instance is created during the lifetime of the application.
  - Controlled access to the instance (often thread-safe).
  - Useful in scenarios like managing global application state.
- Real-World Analogy:
- Think of a government. There can only be one valid passport authority in a country at any time. Everyone uses the same authority to get their passport.

2. FACTORY PATTERN

- Definition:
- A design pattern that provides a way to create objects without specifying their concrete classes.
- Use Case:
- When a system needs to create multiple related objects, but the exact type of object isn't known until runtime.
- Key Features:
- Encapsulates object creation logic.
- Promotes loose coupling by reducing dependencies on specific implementations.
- Real-World Analogy:
- A car manufacturing factory. Depending on the model ordered (sedan, SUV, or truck), the factory produces the specific car without the customer knowing the internal production details.

3. BUILDER PATTERN

- Definition:
- A pattern used to construct complex objects step-by-step, allowing for the creation of different representations of the same object.
- Use Case:
- When creating an object that requires numerous configurations, such as assembling a user profile or constructing an HTML document.
- Key Features:
- Separates object construction from its representation.
- The same construction process can create different types of objects.
- Real-World Analogy:
- Building a custom pizza. You specify the size, crust, toppings, and sauces step-by-step, and the final pizza is made based on your selections.

4. PROTOTYPE PATTERN

- Definition:
- A pattern used to create a new object by copying an existing object (a prototype).
- Use Case:
- When object creation is expensive (e.g., creating complex objects with many initializations or database operations) or when you need a copy of an existing object with slight modifications.
- Key Features:
- Supports cloning of objects.
- Reduces the overhead of creating new objects from scratch.
- Real-World Analogy:
- Think of photocopying. You have an original document (the prototype), and you can make identical or slightly modified copies as needed.

5. ADAPTER PATTERN

- Definition
- The Adapter Pattern is a design pattern that allows incompatible interfaces to work together by providing a middle layer that translates one interface into another.
- Use Case
- Integrating legacy systems with new ones.
- Adapting third-party APIs to match your application's interface.
- Key Features
- Translates interfaces for compatibility.
- Enables seamless integration without changing components.
- Focuses on a single responsibility of translation.
- Real-World Analogy
- A universal power plug adapter converts one plug type to fit different sockets, just like the Adapter Pattern converts incompatible interfaces in software.

HOW THESE PATTERNS WORK TOGETHER IN YOUR PROJECT??

SINGLETON

Used for classes where only one instance should exist, like managing the medical clinic system or patient database.

FACTORY

Used for creating objects like patients, doctors, or appointments dynamically, depending on input data.

BUILDER

Helpful when constructing complex objects like patient records with optional fields (e.g., address, phone number).

PROTOTYPE

Used for duplicating objects like patient details to quickly create a new medical record with some modifications.

ADAPTER

The Adapter pattern allows incompatible systems to work together by converting interfaces, ensuring smooth integration without changing existing code.

DATABASE OVERVIEW:

PURPOSE:

THE DATABASE PROVIDES A SECURE AND CENTRALIZED STORAGE SOLUTION FOR MANAGING PATIENTS, DOCTORS, APPOINTMENTS, AND MEDICAL RECORDS.

KEY TABLES:

- USERLOGIN: HANDLES USER AUTHENTICATION AND ROLES.
- PATIENTS: STORES PATIENT DETAILS LIKE NAME, CONTACT, AND GENDER.
- DOCTORS: MAINTAINS DOCTOR INFORMATION, INCLUDING SPECIALIZATION AND CONTACT.
- APPOINTMENTS: MANAGES SCHEDULING BETWEEN PATIENTS AND DOCTORS.
- MEDICALRECORDS: RECORDS PATIENT VISIT DETAILS AND DOCTOR DIAGNOSES.

DATABASE TABLES

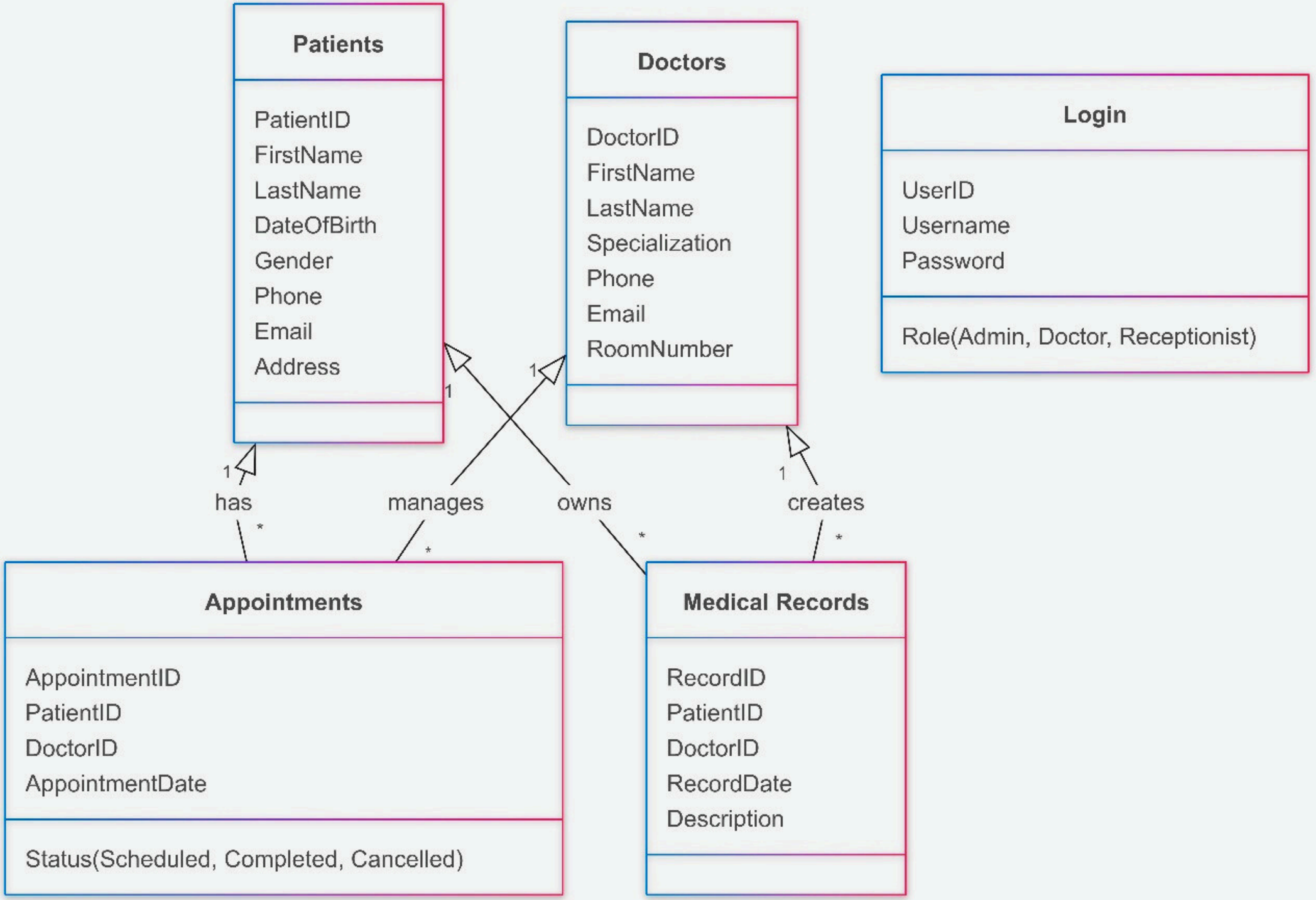
USERLOGIN TABLE: STORES LOGIN CREDENTIALS AND USER ROLES.
USERID: PRIMARY KEY
USERNAME, PASSWORD, ROLE: USER DETAILS

PATIENTS TABLE: STORES PERSONAL INFORMATION ABOUT PATIENTS.
PATIENTID: PRIMARY KEY
FIRSTNAME, LASTNAME, DATEOFBIRTH, ETC.

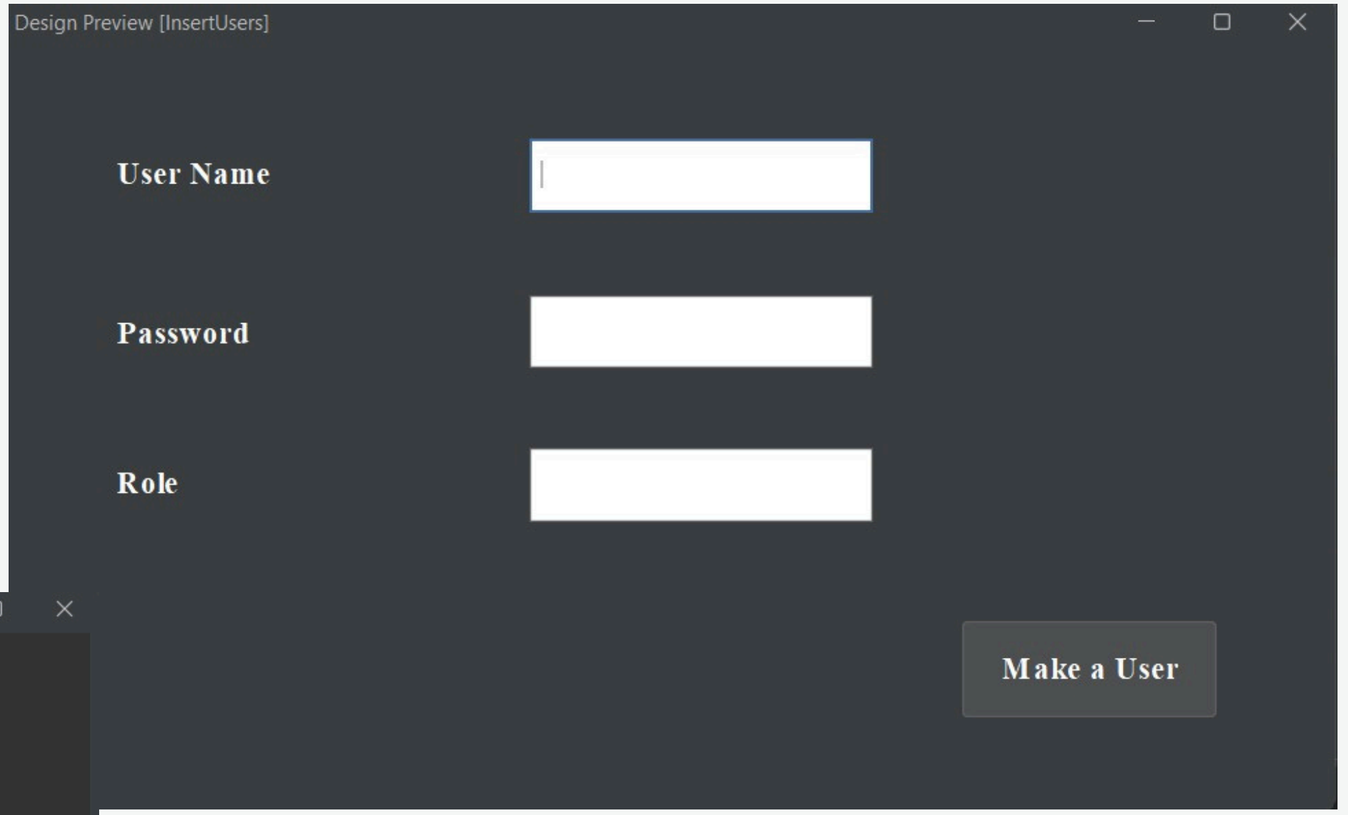
DOCTORS TABLE: STORES INFORMATION ABOUT DOCTORS.
DOCTORID: PRIMARY KEY
FIRSTNAME, SPECIALIZATION, ETC.

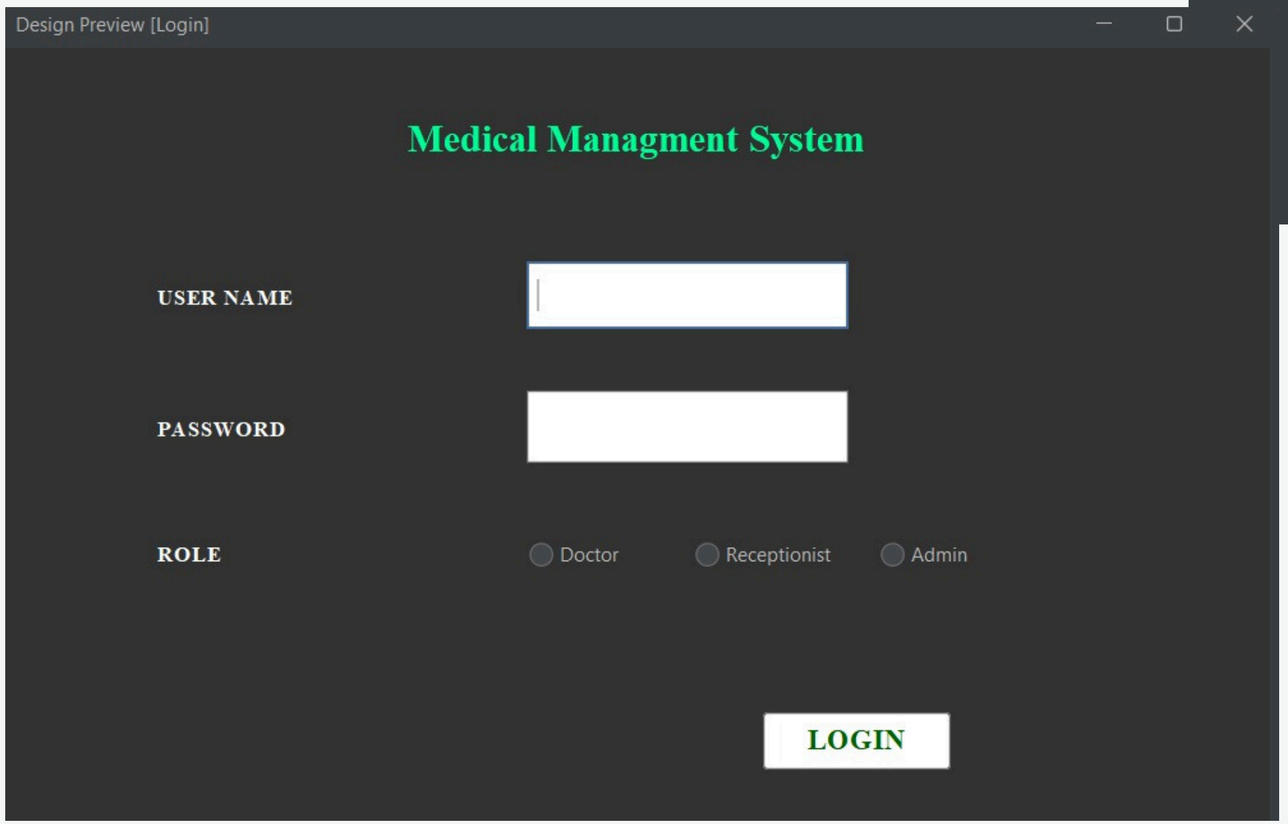
APPOINTMENTS TABLE: TRACKS APPOINTMENTS BETWEEN PATIENTS AND DOCTORS.
APPOINTMENTID: PRIMARY KEY
PATIENTID, DOCTORID: FOREIGN KEYS
APPOINTMENTDATE, STATUS

MEDICALRECORDS TABLE: STORES MEDICAL RECORDS FOR EACH PATIENT.
RECORDID: PRIMARY KEY
PATIENTID, DOCTORID: FOREIGN KEYS
RECORDDATE, DESCRIPTION

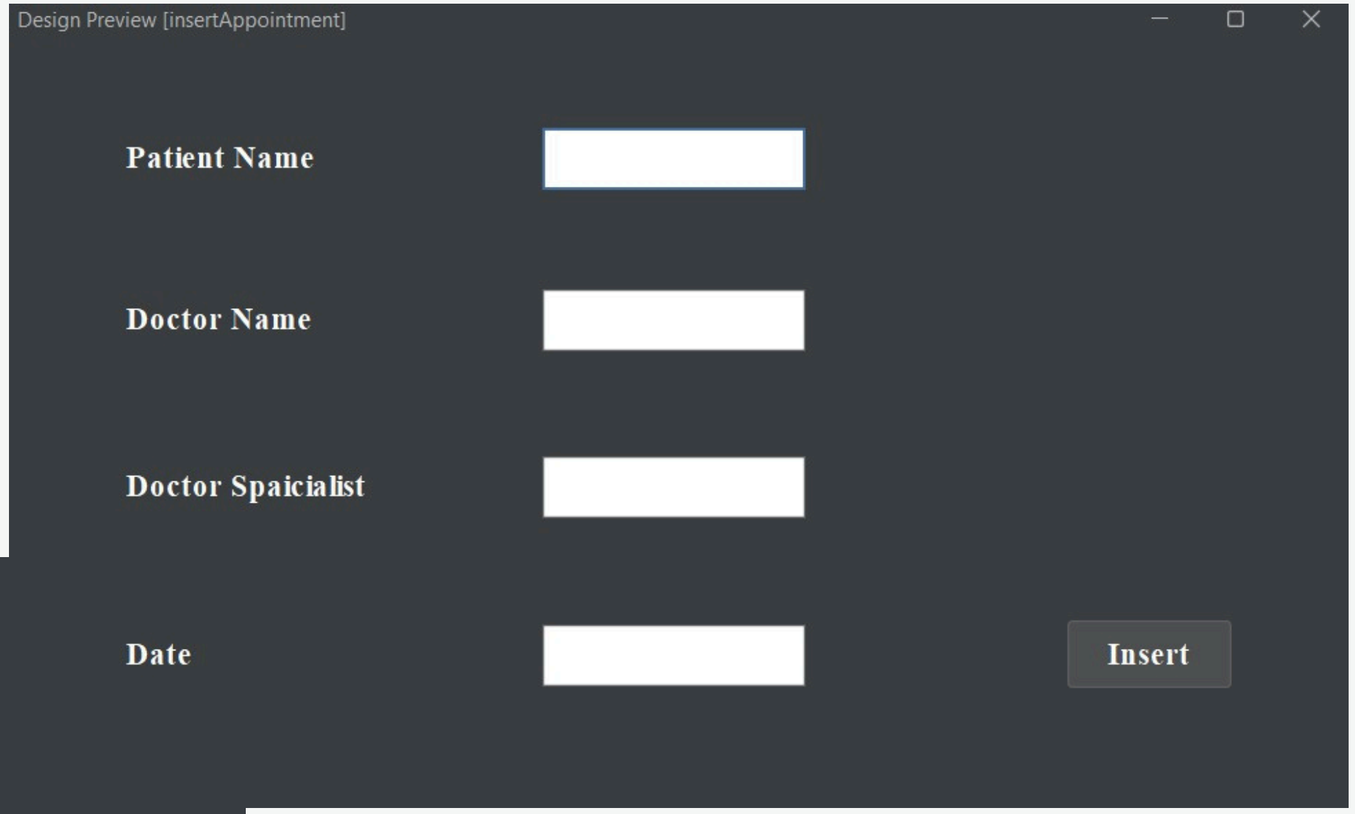

Patients
PatientID
FirstName
LastName
DateOfBirth
Gender
Phone
Email
Address
Doctors
DoctorID
FirstName
LastName
Specialization
Phone
Email
RoomNumber
Login
UserID
Username
Password
Role(Admin, Doctor, Receptionist)
1
1
1
1
has
manages
owns
creates
*
*
*
*
Appointments
AppointmentID
PatientID
DoctorID
AppointmentDate
Status(Scheduled, Completed, Cancelled)
Medical Records
RecordID
PatientID
DoctorID
RecordDate
Description

LOGIN

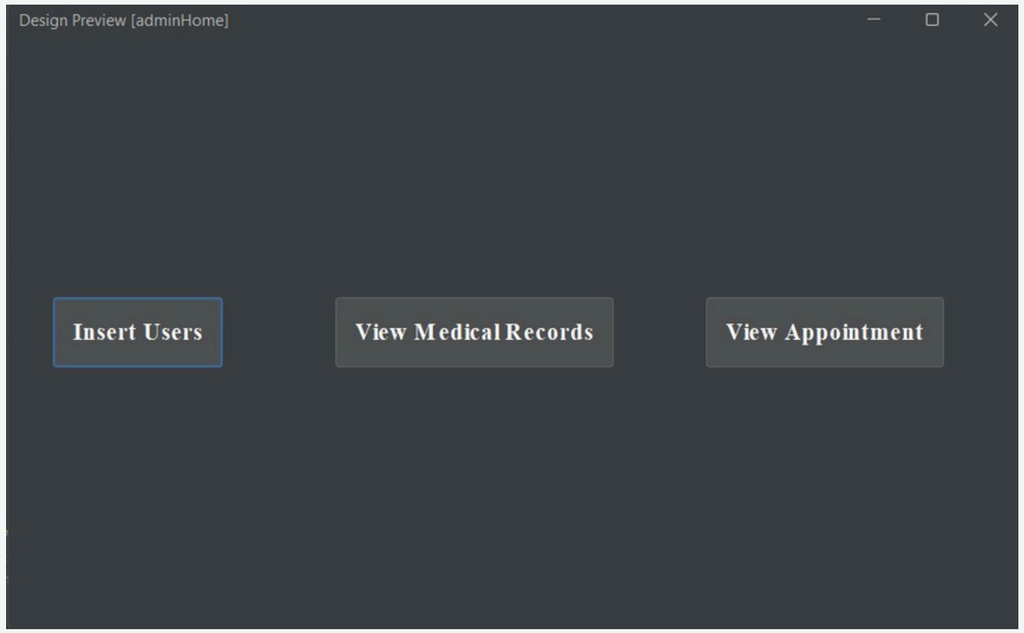

Design Preview [InsertUsers]
User Name
Password
Role
Make a User


Design Preview [Login]
Medical Managment System
USER NAME
PASSWORD
ROLE
Doctor
Receptionist
Admin
LOGIN

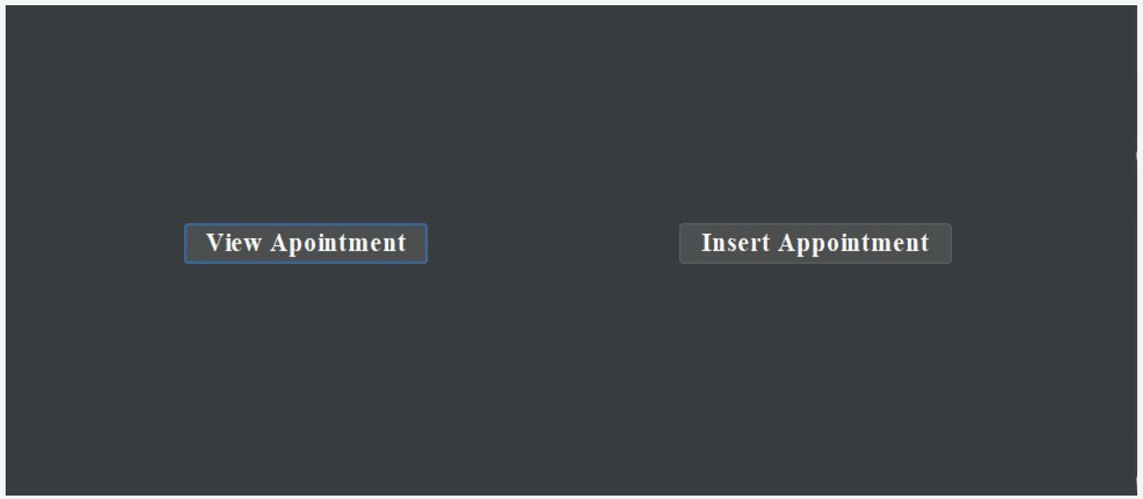
INSWRT

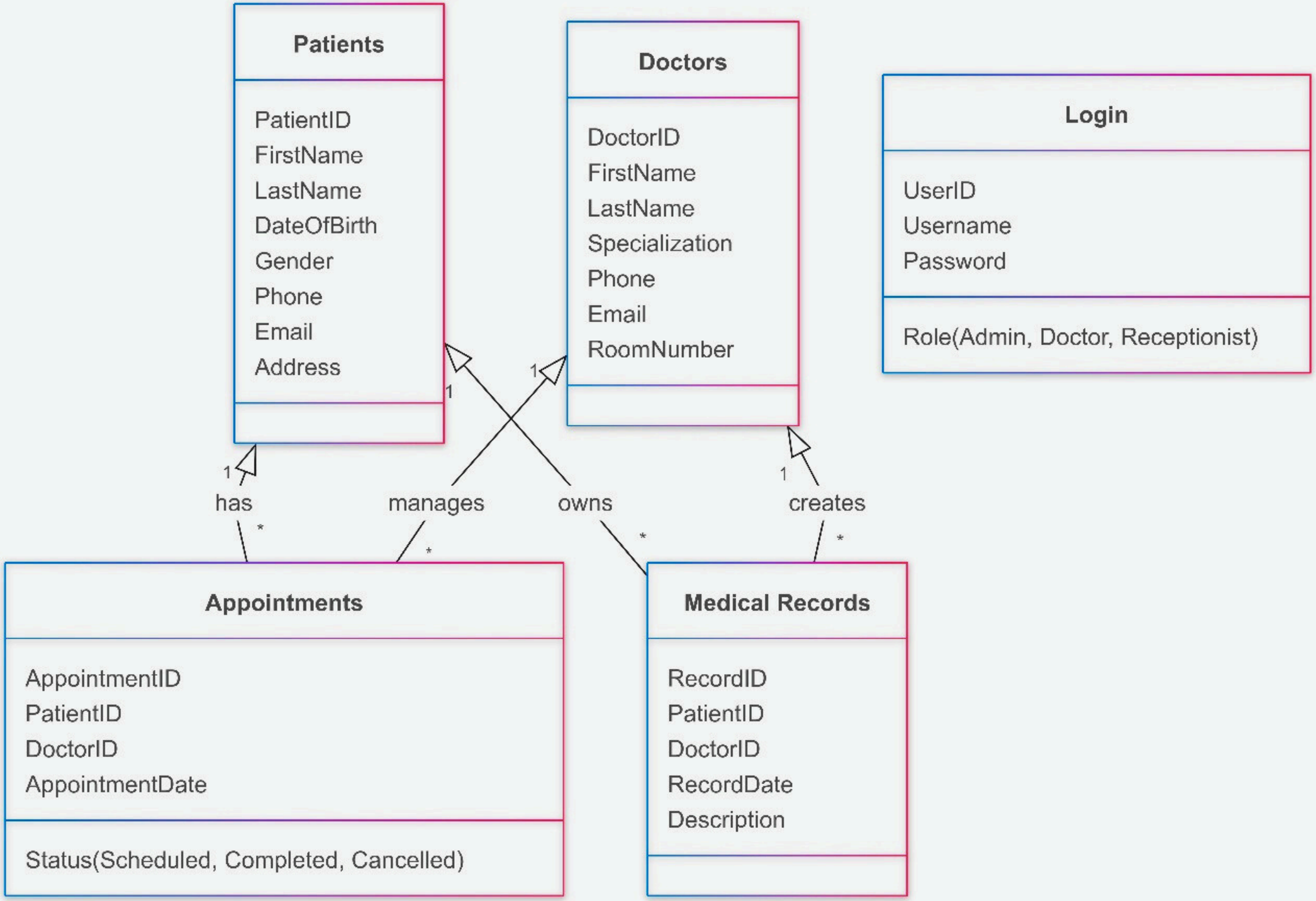


RECORD


Design Preview [adminHome]
Insert Users
View Medical Records
View Appointment


Design Preview [doctorHome]
View Appointment
View Medical Records
Insert Medical Records


View Apointment
Insert Appointment


Patients
PatientID
FirstName
LastName
DateOfBirth
Gender
Phone
Email
Address
Doctors
DoctorID
FirstName
LastName
Specialization
Phone
Email
RoomNumber
Login
UserID
Username
Password
Role(Admin, Doctor, Receptionist)
1
1
1
1
has
manages
owns
creates
*
*
*
*
Appointments
AppointmentID
PatientID
DoctorID
AppointmentDate
Status(Scheduled, Completed, Cancelled)
Medical Records
RecordID
PatientID
DoctorID
RecordDate
Description

OVERALL PROJECT ADVANTAGES

- IMPROVED FLEXIBILITY, MAINTAINABILITY, AND SCALABILITY BY COMBINING DESIGN PATTERNS AND RELATIONAL DATABASE SYSTEMS.
- CLEAR DATA STRUCTURE FOR MANAGING PATIENT RECORDS, DOCTORS, APPOINTMENTS, AND MEDICAL HISTORIES.

OVERALL PROJECT DISADVANTAGES

- INCREASED COMPLEXITY AND DEVELOPMENT TIME DUE TO INTEGRATING MULTIPLE DESIGN PATTERNS AND MANAGING DATABASE RELATIONSHIPS.
- RISK OF PERFORMANCE ISSUES WHEN HANDLING LARGE DATASETS OR COMPLEX QUERIES.

OUR TEAM

1. MAHMOUD ALSOUDY AHMAD 21-00911

2. RENAD ABDELRAHMAN 21-01500

3. SHAHD OMAR MAHMOUD 21-01596

4. ENAS ESSAM MOHAMED 21-00633

THANKS