HOSPITAL MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

GOKULAKKANNAN P	**231501053**
GOKULAKRISHNAN S	**231501054**
GUNAVAZHAGAN B	**231501055**
HARISH M	**231501059**

In partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2024 - 2025

BONAFIDE CERTIFICATE

Certified that this project report "**HOSPITAL MANAGEMENT SYSTEM**" is the bonafide work of **"GOKULAKKANNAN P (231501053), GOKULAKRISHNAN S (231501054), GUNAVAZHAGAN B (231501055), HARISH M (231501059)"** who carried out the project work under my supervision.

Submitted for the Practical Examination held on ________________________

SIGNATURE

Mr. U. Kumaran,
Assistant Professor (SS)
AIML,
Rajalakshmi Engineering College (Autonomous),
Thandalam, Chennai - 602 105

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This project presents a robust Hospital Management System built using DBMS principles to streamline essential hospital operations for administrators, doctors, and patients. The system enables Admin users to manage patient records, assign doctors to patients, and monitor billing. Through the Doctor module, doctors can access assigned patient information, update medical records, issue prescriptions, and generate bills for services provided. Patient records, appointment schedules, and billing details are securely stored and readily accessible.Developed with MySQL, the system incorporates advanced database management features like joins, views, and triggers to maintain data integrity and operational efficiency. The user interface, designed with Python’s Custom Tkinter, provides an intuitive experience with fixed-length pop- up windows for simplified navigation. This project effectively demonstrates the application of database concepts to enhance the efficiency of hospital management systems.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION:

Efficient hospital management is essential for delivering quality patient care and optimizing operational workflows. This project introduces a Hospital Management System (HMS) designed with database management principles to centralize the administration of key functions for hospital staff. The system supports roles for administrators and doctors, enabling streamlined management of patient records, appointments, doctor assignments, and billing operations.

The HMS is built on MySQL, utilizing advanced database features including joins, views, and triggers to maintain data consistency and integrity. The Admin Module allows administrators to manage patient information, assign doctors based on availability, and oversee billing records. Through the Doctor Module, doctors can securely access patient details, update records, provide prescriptions, and manage billing for services rendered.

The system interface, developed with Python's Custom Tkinter library, offers a user-friendly experience with fixed-length pop-up windows for efficient navigation. This project demonstrates the application of database management concepts within healthcare, offering a structured solution to enhance the effectiveness and organization of hospital operations.

1.2 OBJECTIVES:

- **Centralized Patient Management**: Consolidate patient records for easy access and organization.
- **Optimized Appointment Scheduling**: Allow admins to assign doctors based on availability and specialization.
- **Streamlined Billing**: Enable doctors to create and manage billing directly in the system.
- **Data Integrity**: Use MySQL features to ensure consistent, real-time data.
- **Secure Role-based Access**: Provide tailored access for admins and doctors.
- **User-friendly Interface**: Implement a Tkinter GUI with fixed-length pop-ups for efficient navigation.

1.3 MODULES:

- **Admin Module**: Manages patient records, assigns doctors, and oversees billing.
- **Doctor Module**: Accesses patient details, updates record, prescribes treatments, and generates bills.
- **Appointment Management Module**: Schedules and organizes patient appointments based on doctor availability.
- **Database Management Module**: Stores and maintains patient, doctor, and billing data using **MySQL.**
- **User Interface Module**: Provides a **Tkinter**-based GUI with fixed-length pop-ups for streamlined interaction.

2. SURVEY OF TECHNOLOGIES

2.1 Software Description

- MySQL: Manages relational data storage for patient, doctor, and billing records.
- Python: Primary programming language for implementing system logic, database interactions, and user interface.
- Tkinter: Python's standard GUI library for developing interactive applications with a user-friendly interface.
- Custom Tkinter: Enhances the standard Tkinter library for improved UI aesthetics and functionality.

2.2 Languages

2.2.1 SQL

Structured Query Language (SQL) is utilized to define, manipulate, and manage data in relational databases. In this project, SQL is responsible for:

- Creating and modifying tables for patient, doctor, and billing records.
- Retrieving patient information and appointment details through queries for display in the GUI.
- Ensuring data integrity with foreign keys and constraints to maintain accurate relationships between tables.

2.2.2 Python

Python serves as the primary programming language for this project, selected for its simplicity and extensive libraries for database management and user interface development.

- **MySQL Connector**: Facilitates communication between Python and the SQL database for executing queries and transactions.
- **Tkinter**: Provides a graphical user interface (GUI) for users to interact with the hospital management system effectively.

3. REQUIREMENTS AND ANALYSIS

3.1 Requirement Specification

- **Functional Requirements**:

  1. Admins can manage patient records and assign doctors to patients.
  2. Doctors can access patient details, update medical records, and generate billing information.
  3. Store patient, doctor, and billing records with timestamps in the SQL database.
  4. Provide a GUI to display and manage patient information, appointments, and billing records.

- **Non-functional Requirements**:

  1. Ensure fast and accurate data retrieval and updates with minimal latency.
  2. Maintain secure and reliable database storage to protect sensitive patient information.
  3. Design a scalable system capable of handling a growing number of patients and transactions.

3.2 Hardware and Software Requirements

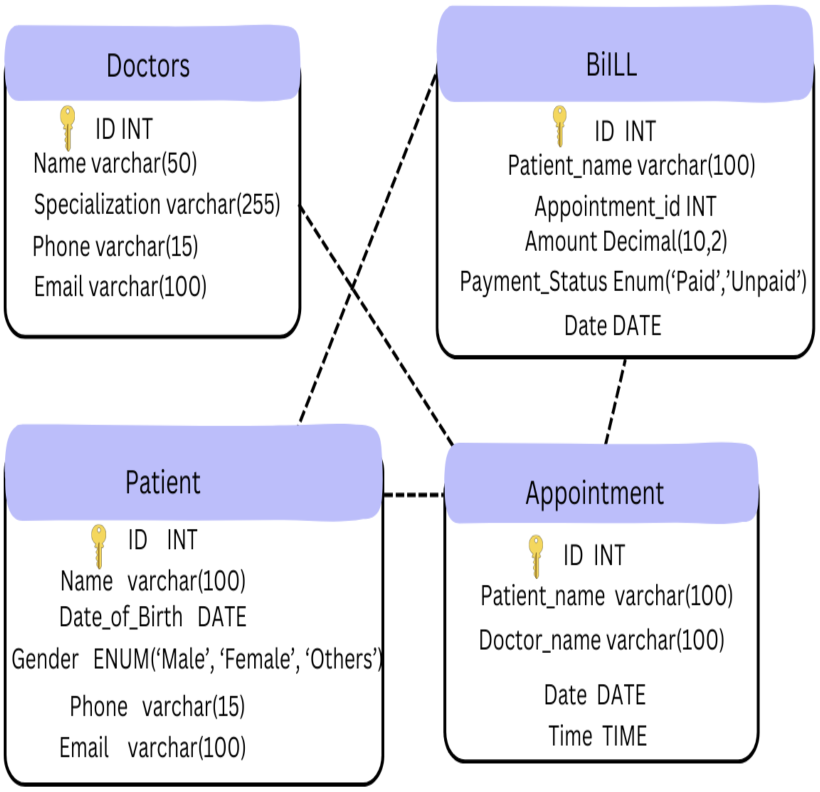
Hardware Requirements:

- Computer with Windows 10 or Linux operating system.
- Minimum 4 GB RAM and 500 GB storage.

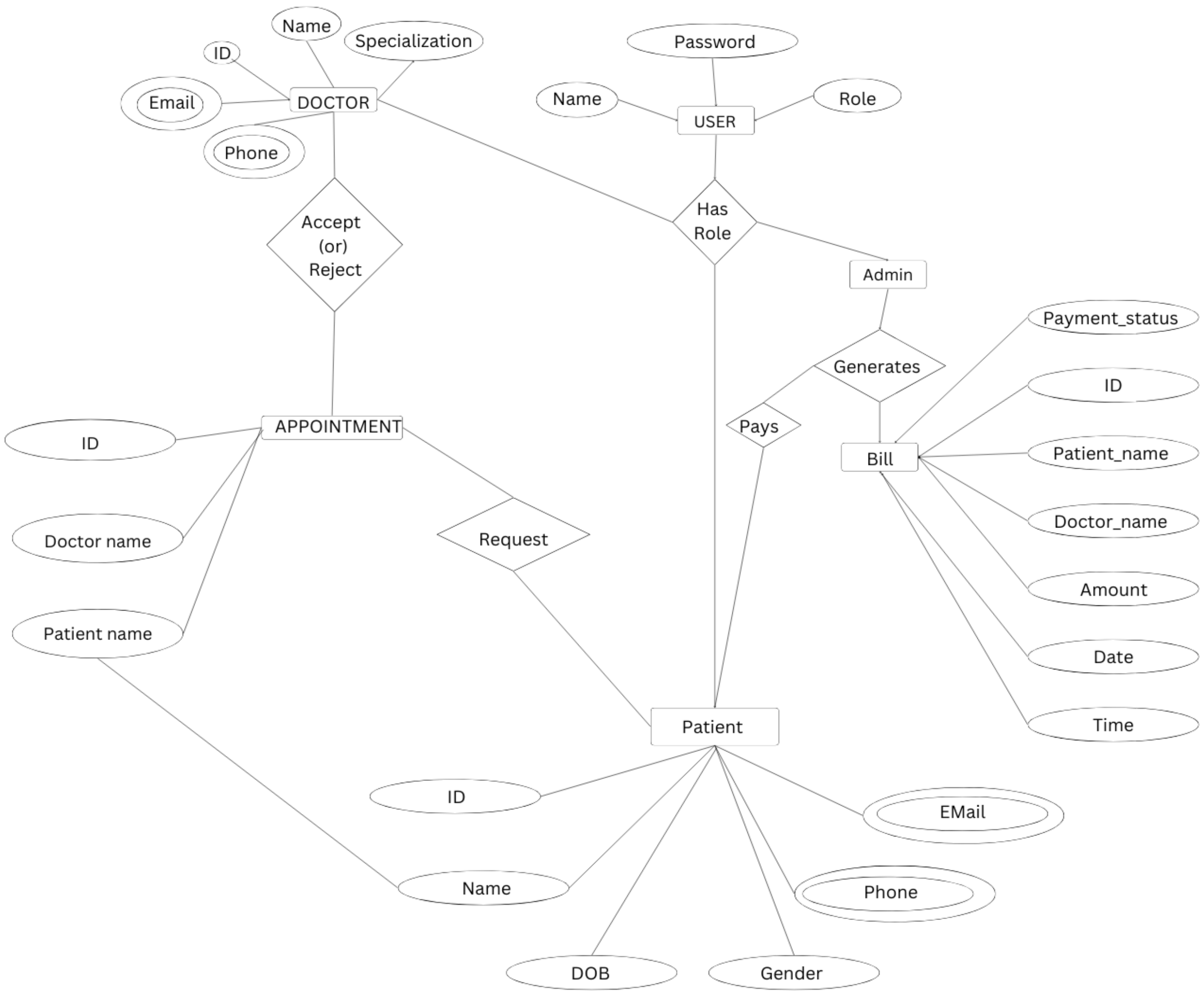
Software Requirements:

- Python 3.x
- MySQL Server
- Libraries: mysql-connector, custom tkinter

3.3 Architecture Diagram



3.4 ER Diagram



4. Program Code

PYTHON CODE

```
import customtkinter as ctk
import mysql.connector
from mysql.connector import Error
from datetime import datetime

# Initialize customtkinter
ctk.set_appearance_mode("System")
ctk.set_default_color_theme("blue")

# MySQL Database Connection
def create_connection():
    try:
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="MyRoot",
            database="user_db"
        )
        return conn
    except Error as e:
        print(f"Error: '{e}'")
        return None

# Helper function to execute a MySQL query with parameters

def execute_query(query, params=()):
    results = None
    conn = None
    cursor = None
    try:
        # Establish a new connection
        conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="MyRoot",
            database="user_db"
        )
        cursor = conn.cursor()
        cursor.execute(query, params)

        # Check if the query is a SELECT statement
        if query.strip().upper().startswith("SELECT"):
            results = cursor.fetchall()  # Fetch all results if it's a SELECT query
        else:
            conn.commit()  # Commit changes for INSERT, UPDATE, DELETE queries
    except mysql.connector.Error as err:
        print(f"Error: {err}")
    finally:
        if cursor is not None:
            cursor.close()  # Close the cursor if it was created
        if conn is not None:
            conn.close()    # Close the connection if it was created

    return results


# Function to check login credentials
def login():
    conn = create_connection()
    if conn is not None:
        cursor  =  conn.cursor()
        username = entry_username.get()
        password = entry_password.get()
        role = role_var.get()

        cursor.execute("SELECT * FROM users WHERE username = %s AND password = %s AND
role = %s", (username, password, role))
        result = cursor.fetchone()
        conn.close()

        if result:
            label_message.configure(text="Login Successful!", text_color="green")
            open_role_window(role)  # Open the role-specific window
        else:
            label_message.configure(text="Invalid Username, Password, or Role",
text_color="red")
    else:
        label_message.configure(text="Database Connection Failed", text_color="red")

# Function to open role-specific window
def open_role_window(role):
    global content_frame  # Make content_frame accessible globally
    role_window = ctk.CTkToplevel()
    role_window.title(f"{role} Dashboard")
    role_window.geometry("800x600")

    # Create sidebar frame with a distinct color
    sidebar_frame = ctk.CTkFrame(role_window, width=180, corner_radius=0,
fg_color="#2D2F3A")
    sidebar_frame.pack(side="left", fill="y")

    # Create main content frame (right side) with padding
    content_frame = ctk.CTkFrame(role_window, width=600, fg_color="grey")
    content_frame.pack(side="right", expand=True, fill="both", padx=10, pady=10)
```

```
# Function to update content based on the selected option/sub-option
def display_content(content):
    # Clear existing content
    for widget in content_frame.winfo_children():
        widget.destroy()

    # Add Appointments View
    if content == "BILL - ADD APPOINTMENTS":
        add_appointment_view(content_frame)

    # Update Appointments View
    elif content == "BILL - UPDATE APPOINTMENTS":
        update_appointment_view(content_frame)

    # Appointment Detail View
    elif content == "BILL - APPOINTMENT DETAIL":
        appointment_detail_view(content_frame)

    if content == "BILL - ADD DOCTOR":
        add_doctor_view(content_frame)

    # Update Doctors View
    elif content == "BILL - UPDATE DOCTOR":
        update_doctor_view(content_frame)

    # Doctor Detail View
    elif content == "BILL - DOCTOR DETAIL":
        doctor_detail_view(content_frame)

    if content == "BILL - ADD PATIENT":
        add_patient_view(content_frame)

    # Update Doctors View
    elif content == "BILL - UPDATE PATIENT":
        update_patient_view(content_frame)

    # Doctor Detail View
    elif content == "BILL - PATIENT DETAIL":
        patient_detail_view(content_frame)

    if content == "BILL - GENERATE BILL":
        generate_bill_view(content_frame)

    elif content == "BILL - VIEW BILL":
        view_bill_view(content_frame)

    elif content == "BILL - DELETE BILL":
        delete_bill_view(content_frame)

    # Define views for Doctors, Patients, and Bill sections here as needed
    # Admin sidebar with sub-options and styled buttons
    if role == "Admin":
        sidebar_options = {
            "APPOINTMENTS": ["ADD APPOINTMENTS", "UPDATE APPOINTMENTS", "APPOINTMENT
DETAIL"],
            "DOCTORS": ["ADD DOCTOR", "UPDATE DOCTOR", "DOCTOR DETAIL"],
            "PATIENTS": ["ADD PATIENT", "UPDATE PATIENT", "PATIENT DETAIL"],
            "BILL": ["GENERATE BILL", "VIEW BILL", "DELETE BILL"]
        }

        for main_option, sub_options in sidebar_options.items():
            # Main section header (without command, just a label for sub-options)
            main_label = ctk.CTkLabel(sidebar_frame, text=main_option,
text_color="#EAEAEA", fg_color="#2D2F3A", font=("Arial", 12, "bold"))
            main_label.pack(pady=(10, 0), padx=10, anchor="w")

            for sub_option in sub_options:
                sub_button = ctk.CTkButton(
                    sidebar_frame,
                    text=sub_option,
                    command=lambda opt=sub_option: display_content(f"{main_option} -
{opt}"),
                    fg_color="#3C3F51",
                    hover_color="#555A71",
                    text_color="#FFFFFF"
                )
                sub_button.pack(pady=2, padx=20, anchor="w")

    # Add a label in the content frame to show the initial dashboard text
    initial_label = ctk.CTkLabel(content_frame, text=f"Welcome to {role} Portal",
font=("Arial", 24))
    initial_label.pack(pady=40)

# Function to open registration window
def open_register_window():
    register_window = ctk.CTkToplevel()
    register_window.title("Register")
    register_window.geometry("400x400")

    def register_user():
        conn = create_connection()
        if conn is not None:
            cursor = conn.cursor()
            new_username = entry_new_username.get()
            new_password = entry_new_password.get()
            new_role  =  register_role_var.get()
            try:
                cursor.execute("INSERT INTO users (username, password, role) VALUES
(%s, %s, %s)", (new_username, new_password, new_role))
                conn.commit()
                label_register_message.configure(text="User Registered!", 
text_color="green")
            except Error as e:
                label_register_message.configure(text="Error: User already exists", 
text_color="red")
            conn.close()

    # Registration form
    label_register = ctk.CTkLabel(register_window, text="Register", font=("Arial", 
24))
    label_register.pack(pady=20)

    label_new_username = ctk.CTkLabel(register_window, text="New Username:")
    label_new_username.pack()
    entry_new_username = ctk.CTkEntry(register_window)
    entry_new_username.pack(pady=5)

    label_new_password = ctk.CTkLabel(register_window, text="New Password:")
    label_new_password.pack()
    entry_new_password = ctk.CTkEntry(register_window, show="*")
    entry_new_password.pack(pady=5)

    label_role = ctk.CTkLabel(register_window, text="Role:")
    label_role.pack()
    register_role_var = ctk.StringVar(value="Admin")
    role_options = ctk.CTkComboBox(register_window, variable=register_role_var, 
values=["Admin", "Doctor", "Pharmacist"])
    role_options.pack(pady=5)

    button_create_account = ctk.CTkButton(register_window, text="Create Account", 
command=register_user)
    button_create_account.pack(pady=10)

    label_register_message = ctk.CTkLabel(register_window, text="")
    label_register_message.pack()

# View for Adding an Appointment
def add_appointment_view(parent):
    ctk.CTkLabel(parent, text="Add Appointment", font=("Arial", 18)).pack(pady=10)

    # Input fields for patient name, doctor name, date, and time
    labels = ["Patient Name:", "Doctor Name:", "Date (YYYY-MM-DD):", "Appointment Time 
(HH:MM):"]
    entries = []
    for label_text in labels:
        ctk.CTkLabel(parent, text=label_text).pack()
        entry = ctk.CTkEntry(parent)
        entry.pack(pady=5)
        entries.append(entry)

    # Button to add the appointment to the database
    def add_appointment():
        patient_name, doctor_name, date, time = [e.get() for e in entries]

        # Check if patient exists; if not, display message
        patient_exists = execute_query(
            "SELECT COUNT(*) FROM patients WHERE name = %s", (patient_name,)
        )[0][0]

        if not patient_exists:
            ctk.CTkLabel(parent, text=f"Patient '{patient_name}' does not exist.
Please add patient information.", text_color="red").pack()
            return

        # Check if doctor exists; if not, display message
        doctor_exists = execute_query(
            "SELECT COUNT(*) FROM doctors WHERE name = %s", (doctor_name,)
        )[0][0]

        if not doctor_exists:
            ctk.CTkLabel(parent, text=f"Doctor '{doctor_name}' does not exist. Please
add doctor information.", text_color="red").pack()
            return

        # Insert the appointment now that patient and doctor exist in the database
        execute_query(
            "INSERT INTO appointments (patient_name, doctor_name, date, time) VALUES
(%s, %s, %s, %s)",
            (patient_name, doctor_name, date, time)
        )
        ctk.CTkLabel(parent, text="Appointment added successfully!",
text_color="green").pack()


    ctk.CTkButton(parent, text="Add Appointment",
command=add_appointment).pack(pady=10)

# View for Updating an Appointment
def update_appointment_view(parent):
    ctk.CTkLabel(parent, text="Update Appointment", font=("Arial", 18)).pack(pady=10)

    # Search bar to find an appointment
    search_entry = ctk.CTkEntry(parent, placeholder_text="Search Patient Name")
    search_entry.pack(pady=5)

    # Frame for search results
    search_results_frame = ctk.CTkFrame(parent)
    search_results_frame.pack(fill="x", padx=10, pady=5)

    # Fields to update patient name, doctor name, date, and time
    update_labels = ["Appointment ID:", "Patient Name:", "Doctor Name:", "Date (YYYY-
MM-DD):", "Appointment Time (HH:MM):"]
```

```
    update_entries = []
    for label_text in update_labels:
        ctk.CTkLabel(parent, text=label_text).pack()
        entry = ctk.CTkEntry(parent)
        entry.pack(pady=5)
        update_entries.append(entry)

    # Function to search for a patient and display results
    def search_appointment():
        patient_name = search_entry.get()
        results = execute_query("SELECT * FROM appointments WHERE patient_name = %s",
(patient_name,))

        # Clear previous results
        for widget in search_results_frame.winfo_children():
            widget.destroy()

        if results:
            for row in results:
                result_label = ctk.CTkLabel(search_results_frame, text=f"Appointment
ID: {row[0]}, Patient: {row[1]}, Doctor: {row[2]}, Date: {row[3]}, Time: {row[4]}")
                result_label.pack()

                # Adding button to fill fields with selected result
                update_button = ctk.CTkButton(search_results_frame, text="Select",
command=lambda appointment=row: fill_update_fields(appointment))
                update_button.pack(pady=5)

        else:
            ctk.CTkLabel(search_results_frame, text="No results found").pack()

    # Function to fill the update fields with selected appointment details
    def fill_update_fields(appointment):
        # Populate fields with current appointment details
        for entry, value in zip(update_entries[1:], appointment[1:]):  # Skip
appointment ID
            entry.delete(0, ctk.END)
            entry.insert(0, value)

        # Set the appointment ID in the first entry
        update_entries[0].delete(0, ctk.END)
        update_entries[0].insert(0, appointment[0])  # Appointment ID

    ctk.CTkButton(parent, text="Search Appointment",
command=search_appointment).pack(pady=5)

    # Button to update the appointment in the database
    def update_appointment():
        appointment_id = update_entries[0].get().strip()  # Get the appointment ID

        # Fetch the current appointment details
        current_appointment = execute_query(
            "SELECT patient_name, doctor_name, date, time FROM appointments WHERE id = 
%s",
            (appointment_id,)
        )

        if not current_appointment:
            ctk.CTkLabel(parent, text="Appointment ID not found.", 
text_color="red").pack()
            return

        # Unpack the current appointment details
        current_patient_name, current_doctor_name, current_date, current_time = 
current_appointment[0]

        # Get new values from the entry fields, or retain current values if the entry 
is empty
        new_patient_name = update_entries[1].get().strip() or current_patient_name
        new_doctor_name = update_entries[2].get().strip() or current_doctor_name
        new_date = update_entries[3].get().strip() or current_date
        new_time = update_entries[4].get().strip() or current_time

        # Check if new patient name exists in the patients table
        if new_patient_name != current_patient_name:  # Only check if it's a change
            patient_exists = execute_query(
                "SELECT COUNT(*) FROM patients WHERE name = %s",
                (new_patient_name,)
            )[0][0]

            if not patient_exists:
                ctk.CTkLabel(parent, text="New patient name does not exist in the 
database.", text_color="red").pack()
                return

        # Perform the update
        execute_query(
            "UPDATE appointments SET patient_name=%s, doctor_name=%s, date=%s, time=%s 
WHERE id=%s",
            (new_patient_name, new_doctor_name, new_date, new_time, appointment_id)
        )

        ctk.CTkLabel(parent, text="Appointment updated successfully!", 
text_color="green").pack()

    ctk.CTkButton(parent, text="Update Appointment", 
command=update_appointment).pack(pady=10)

# View for Appointment Detail
def appointment_detail_view(parent):
    ctk.CTkLabel(parent, text="Appointment Detail", font=("Arial", 18)).pack(pady=10)
    # Search bar to find an appointment
    search_entry = ctk.CTkEntry(parent, placeholder_text="Search Patient Name")
    search_entry.pack(pady=5)

    # Frame for displaying appointment details
    detail_frame = ctk.CTkFrame(parent)
    detail_frame.pack(fill="x", padx=10, pady=5)

    # Function to display all appointments
    def display_all_appointments():
        all_appointments = execute_query("SELECT * FROM appointments",)

        # Clear previous details
        for widget in detail_frame.winfo_children():
            widget.destroy()

        if all_appointments:
            for row in all_appointments:
                detail_label = ctk.CTkLabel(detail_frame, text=f"ID: {row[0]},
Patient: {row[1]}, Doctor: {row[2]}, Date: {row[3]}, Time: {row[4]}")
                detail_label.pack()
        else:
            ctk.CTkLabel(detail_frame, text="No appointments found").pack()

    # Function to search for an appointment and display its details
    def search_appointment_detail():
        patient_name = search_entry.get().strip()
        results = execute_query("SELECT * FROM appointments WHERE patient_name = %s",
(patient_name,))

        # Clear previous details
        for widget in detail_frame.winfo_children():
            widget.destroy()

        if results:
            for row in results:
                detail_label = ctk.CTkLabel(detail_frame, text=f"ID: {row[0]},
Patient: {row[1]}, Doctor: {row[2]}, Date: {row[3]}, Time: {row[4]}")
                detail_label.pack()
        else:
            ctk.CTkLabel(detail_frame, text="No results found").pack()

    # Button to trigger the search function
    ctk.CTkButton(parent, text="Search Appointment",
command=search_appointment_detail).pack(pady=5)

    # Initially display all appointments
    display_all_appointments()

# Add Doctor View
def add_doctor_view(parent):
    ctk.CTkLabel(parent, text="Add Doctor", font=("Arial", 18)).pack(pady=10)

    # Input fields for doctor details
    labels = ["Doctor Name:", "Specialization:", "Phone:", "Email:"]
    entries = []
    for label_text in labels:
        ctk.CTkLabel(parent, text=label_text).pack()
        entry = ctk.CTkEntry(parent)
        entry.pack(pady=5)
        entries.append(entry)

    # Button to add the doctor to the database
    def add_doctor():
        doctor_name, specialization, phone, email = [e.get() for e in entries]

        # Insert the doctor into the database
        execute_query(
            "INSERT INTO doctors (name, specialization, phone, email) VALUES (%s, %s,
%s, %s)",
            (doctor_name, specialization, phone, email)
        )
        ctk.CTkLabel(parent, text="Doctor added successfully!",
text_color="green").pack()

    ctk.CTkButton(parent, text="Add Doctor", command=add_doctor).pack(pady=10)

# Update Doctor View
def update_doctor_view(parent):
    ctk.CTkLabel(parent, text="Update Doctor", font=("Arial", 18)).pack(pady=10)

    # Search bar to find a doctor
    search_entry = ctk.CTkEntry(parent, placeholder_text="Search Doctor Name")
    search_entry.pack(pady=5)

    # Frame for search results
    search_results_frame = ctk.CTkFrame(parent)
    search_results_frame.pack(fill="x", padx=10, pady=5)

    # Fields to update doctor details
    update_labels = ["Doctor ID:", "Doctor Name:", "Specialization:", "Phone:",
"Email:"]
    update_entries = []
    for label_text in update_labels:
        ctk.CTkLabel(parent, text=label_text).pack()
        entry = ctk.CTkEntry(parent)
        entry.pack(pady=5)
        update_entries.append(entry)

    # Function to search for a doctor and display results
    def search_doctor():
        doctor_name = search_entry.get().strip()
        results = execute_query("SELECT * FROM doctors WHERE name = %s",
(doctor_name,))

        # Clear previous results
        for widget in search_results_frame.winfo_children():
            widget.destroy()

        if results:
            for row in results:
                result_label = ctk.CTkLabel(
                    search_results_frame,
                    text=f"Doctor ID: {row[0]}, Name: {row[1]}, Specialization:
{row[2]}, Phone: {row[3]}, Email: {row[4]}"
                )
                result_label.pack()

                # Adding button to fill fields with selected result
                update_button = ctk.CTkButton(search_results_frame, text="Select",
command=lambda doctor=row: fill_update_fields(doctor))
                update_button.pack(pady=5)

        else:
            ctk.CTkLabel(search_results_frame, text="No results found").pack()

    # Function to fill the update fields with selected doctor details
    def fill_update_fields(doctor):
        # Populate fields with current doctor details
        for entry, value in zip(update_entries[1:], doctor[1:]):  # Skip doctor ID
            entry.delete(0, ctk.END)
            entry.insert(0, value)

        # Set the doctor ID in the first entry
        update_entries[0].delete(0, ctk.END)
        update_entries[0].insert(0, doctor[0])  # Doctor ID

    ctk.CTkButton(parent, text="Search Doctor", command=search_doctor).pack(pady=5)

    # Button to update the doctor in the database
    def update_doctor():
        doctor_id = update_entries[0].get().strip()  # Get the doctor ID

        # Get new values from the entry fields
        new_doctor_name = update_entries[1].get().strip()
        new_specialization = update_entries[2].get().strip()
        new_phone = update_entries[3].get().strip()
        new_email = update_entries[4].get().strip()

        # Perform the update
        execute_query(
            "UPDATE doctors SET name=%s, specialization=%s, phone=%s, email=%s WHERE
id=%s",
            (new_doctor_name, new_specialization, new_phone, new_email, doctor_id)
        )

        ctk.CTkLabel(parent, text="Doctor updated successfully!",
text_color="green").pack()

    ctk.CTkButton(parent, text="Update Doctor", command=update_doctor).pack(pady=10)

# Doctor Details View
def doctor_detail_view(parent):
    ctk.CTkLabel(parent, text="Doctor Detail", font=("Arial", 18)).pack(pady=10)

    # Search bar to find a doctor
    search_entry = ctk.CTkEntry(parent, placeholder_text="Search Doctor Name")
    search_entry.pack(pady=5)

    # Frame for displaying doctor details
    detail_frame = ctk.CTkFrame(parent)
    detail_frame.pack(fill="x", padx=10, pady=5)

    # Function to display all doctors
    def display_all_doctors():
        all_doctors = execute_query("SELECT * FROM doctors")

        # Clear previous details
        for widget in detail_frame.winfo_children():
            widget.destroy()

        if all_doctors:
            for row in all_doctors:
                detail_label = ctk.CTkLabel(
                    detail_frame,
                    text=f"ID: {row[0]}, Name: {row[1]}, Specialization: {row[2]},
Phone: {row[3]}, Email: {row[4]}"
                )
                detail_label.pack()
        else:
            ctk.CTkLabel(detail_frame, text="No doctors found").pack()

    # Function to search for a doctor and display its details
    def search_doctor_detail():
        doctor_name = search_entry.get().strip()
        results = execute_query("SELECT * FROM doctors WHERE name = %s",
(doctor_name,))

        # Clear previous details
        for widget in detail_frame.winfo_children():
            widget.destroy()

        if results:
            for row in results:
                detail_label = ctk.CTkLabel(
                    detail_frame,
                    text=f"ID: {row[0]}, Name: {row[1]}, Specialization: {row[2]},
Phone: {row[3]}, Email: {row[4]}"
                )
                detail_label.pack()
        else:
            ctk.CTkLabel(detail_frame, text="No results found").pack()

    # Button to trigger the search function
    ctk.CTkButton(parent, text="Search Doctor",
command=search_doctor_detail).pack(pady=5)

    # Initially display all doctors
    display_all_doctors()

# Add Patient View
def add_patient_view(parent):
    ctk.CTkLabel(parent, text="Add Patient", font=("Arial", 18)).pack(pady=10)

    labels = ["Name:", "Date of Birth (YYYY-MM-DD):", "Gender:", "Phone:", "Email:"]
    entries = []
    for label_text in labels:
        ctk.CTkLabel(parent, text=label_text).pack()
        entry = ctk.CTkEntry(parent)
        entry.pack(pady=5)
        entries.append(entry)

    def add_patient():
        name, dob, gender, phone, email = [e.get().strip() for e in entries]

        if not all([name, dob, gender, phone, email]):
            ctk.CTkLabel(parent, text="Please fill in all fields.",
text_color="red").pack()
            return

        # Insert patient into the database
        query = "INSERT INTO patients (name, date_of_birth, gender, phone, email)
VALUES (%s, %s, %s, %s, %s)"
        execute_query(query, (name, dob, gender, phone, email))
        ctk.CTkLabel(parent, text="Patient added successfully!",
text_color="green").pack()

    ctk.CTkButton(parent, text="Add Patient", command=add_patient).pack(pady=10)

# Update Patient View
def update_patient_view(parent):
    ctk.CTkLabel(parent, text="Update Patient", font=("Arial", 18)).pack(pady=10)

    search_entry = ctk.CTkEntry(parent, placeholder_text="Enter Patient Name to
Search")
```

```
    search_entry.pack(pady=5)

    update_labels = ["Patient ID:", "Name:", "Date of Birth (YYYY-MM-DD):", "Gender:",
"Phone:", "Email:"]
    update_entries = []
    for label_text in update_labels:
        ctk.CTkLabel(parent, text=label_text).pack()
        entry = ctk.CTkEntry(parent)
        entry.pack(pady=5)
        update_entries.append(entry)
    update_entries[0].configure(state=ctk.DISABLED)  # Disable ID field

    def search_patient():
        name = search_entry.get().strip()
        result = execute_query("SELECT * FROM patients WHERE name = %s", (name,))

        if result:
            patient = result[0]
            for entry, value in zip(update_entries, patient):
                entry.configure(state=ctk.NORMAL)
                entry.delete(0, ctk.END)
                entry.insert(0, value)
            update_entries[0].configure(state=ctk.DISABLED)
        else:
            ctk.CTkLabel(parent, text="No patient found.", text_color="red").pack()

    ctk.CTkButton(parent, text="Search Patient", command=search_patient).pack(pady=5)

    def update_patient():
        patient_id = update_entries[0].get().strip()
        name, dob, gender, phone, email = [e.get().strip() for e in
update_entries[1:]]

        if not patient_id:
            ctk.CTkLabel(parent, text="No patient selected for update.",
text_color="red").pack()
            return

        query = "UPDATE patients SET name=%s, date_of_birth=%s, gender=%s, phone=%s,
email=%s WHERE id=%s"
        execute_query(query, (name, dob, gender, phone, email, patient_id))
        ctk.CTkLabel(parent, text="Patient updated successfully!",
text_color="green").pack()

    ctk.CTkButton(parent, text="Update Patient", command=update_patient).pack(pady=10)

# Patient Details View
def patient_detail_view(parent):
    ctk.CTkLabel(parent, text="Patient Details", font=("Arial", 18)).pack(pady=10)

    search_entry = ctk.CTkEntry(parent, placeholder_text="Search Patient Name")
    search_entry.pack(pady=5)

    detail_frame = ctk.CTkFrame(parent)
    detail_frame.pack(fill="x", padx=10, pady=5)

    def display_all_patients():
        all_patients = execute_query("SELECT * FROM patients")
        for widget in detail_frame.winfo_children():
            widget.destroy()
        if all_patients:
            for patient in all_patients:
                detail_label = ctk.CTkLabel(detail_frame, text=f"ID: {patient[0]},
Name: {patient[1]}, DOB: {patient[2]}, Gender: {patient[3]}, Phone: {patient[4]},
Email: {patient[5]}")
                detail_label.pack()
        else:
            ctk.CTkLabel(detail_frame, text="No patients found.").pack()

    def search_patient_detail():
        name = search_entry.get().strip()
        results = execute_query("SELECT * FROM patients WHERE name LIKE %s", ('%' +
name + '%',))

        for widget in detail_frame.winfo_children():
            widget.destroy()

        if results:
            for patient in results:
                detail_label = ctk.CTkLabel(detail_frame, text=f"ID: {patient[0]},
Name: {patient[1]}, DOB: {patient[2]}, Gender: {patient[3]}, Phone: {patient[4]},
Email: {patient[5]}")
                detail_label.pack()
        else:
            ctk.CTkLabel(detail_frame, text="No results found.",
text_color="red").pack()

    ctk.CTkButton(parent, text="Search Patient",
command=search_patient_detail).pack(pady=5)
    display_all_patients()



# Generate Bill View
def generate_bill_view(parent):
    ctk.CTkLabel(parent, text="Generate Bill", font=("Arial", 18)).pack(pady=10)

    # Input fields for bill details
    labels = ["Patient Name:", "Appointment ID:", "Amount:", "Date (YYYY-MM-DD):"]
    entries = []
    for label_text in labels:
        ctk.CTkLabel(parent, text=label_text).pack()
        entry = ctk.CTkEntry(parent)
        entry.pack(pady=5)
        entries.append(entry)

    # Dropdown for Payment Status (replacing the entry)
    ctk.CTkLabel(parent, text="Payment Status:").pack()
    payment_status_dropdown = ctk.CTkComboBox(parent, values=["Paid", "Unpaid"])
    payment_status_dropdown.pack(pady=5)

    # Button to generate the bill
    def generate_bill():
        patient_name, appointment_id, amount, date = [e.get().strip() for e in
entries]
        payment_status = payment_status_dropdown.get().strip()

        if not all([patient_name, appointment_id, amount, payment_status, date]):
            ctk.CTkLabel(parent, text="Please fill in all fields.",
text_color="red").pack()
            return

        # Insert bill into the database
        query = "INSERT INTO bills (patient_name, appointment_id, amount,
payment_status, date) VALUES (%s, %s, %s, %s, %s)"
        execute_query(query, (patient_name, appointment_id, amount, payment_status,
date))
        ctk.CTkLabel(parent, text="Bill generated successfully!",
text_color="green").pack()

    ctk.CTkButton(parent, text="Generate Bill", command=generate_bill).pack(pady=10)

# View Bill View
def view_bill_view(parent):
    ctk.CTkLabel(parent, text="View Bill", font=("Arial", 18)).pack(pady=10)

    search_entry = ctk.CTkEntry(parent, placeholder_text="Enter Patient Name to
Search")
    search_entry.pack(pady=5)

    detail_frame = ctk.CTkFrame(parent)
    detail_frame.pack(fill="x", padx=10, pady=5)

    # Function to display all bills
    def display_all_bills():
        all_bills = execute_query("SELECT * FROM bills")
        for widget in detail_frame.winfo_children():
            widget.destroy()
        if all_bills:
            for bill in all_bills:
                detail_label = ctk.CTkLabel(detail_frame, text=f"ID: {bill[0]},
Patient: {bill[1]}, Appointment ID: {bill[2]}, Amount: ${bill[3]:.2f}, Status:
{bill[4]}, Date: {bill[5]}")
```

```
                detail_label.pack()
        else:
            ctk.CTkLabel(detail_frame, text="No bills found.").pack()

    # Function to search bills by patient name
    def search_bill():
        patient_name = search_entry.get().strip()
        results = execute_query("SELECT * FROM bills WHERE patient_name LIKE %s", ('%'
+ patient_name + '%',))

        for widget in detail_frame.winfo_children():
            widget.destroy()

        if results:
            for bill in results:
                detail_label = ctk.CTkLabel(detail_frame, text=f"ID: {bill[0]},
Patient: {bill[1]}, Appointment ID: {bill[2]}, Amount: ${bill[3]:.2f}, Status:
{bill[4]}, Date: {bill[5]}")
                detail_label.pack()
        else:
            ctk.CTkLabel(detail_frame, text="No results found.",
text_color="red").pack()

    ctk.CTkButton(parent, text="Search Bill", command=search_bill).pack(pady=5)
    display_all_bills()

# Delete Bill View
def delete_bill_view(parent):
    ctk.CTkLabel(parent, text="Delete Bill", font=("Arial", 18)).pack(pady=10)

    # Search field to enter Bill ID for deletion
    delete_entry = ctk.CTkEntry(parent, placeholder_text="Enter Bill ID to Delete")
    delete_entry.pack(pady=5)

    # Function to delete a bill
    def delete_bill():
        bill_id = delete_entry.get().strip()

        if not bill_id:
            ctk.CTkLabel(parent, text="Please enter a Bill ID.",
text_color="red").pack()
            return

        # Execute delete query
        query = "DELETE FROM bills WHERE id = %s"
        execute_query(query, (bill_id,))
        ctk.CTkLabel(parent, text="Bill deleted successfully!",
text_color="green").pack()

    ctk.CTkButton(parent, text="Delete Bill", command=delete_bill).pack(pady=5)
```

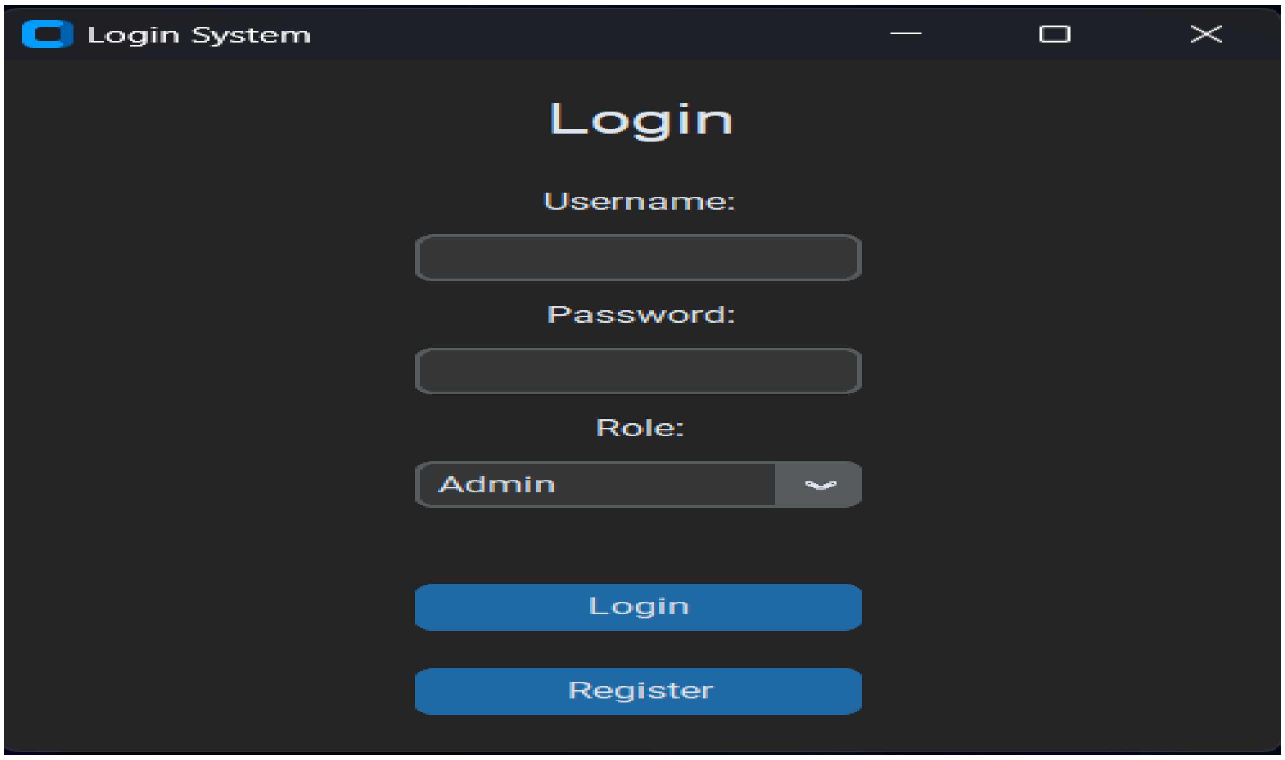
```
# Main application window
app = ctk.CTk()
app.title("Login System")
app.geometry("400x400")

# Login frame
label_title = ctk.CTkLabel(app, text="Login", font=("Arial", 24))
label_title.pack(pady=20)

label_username = ctk.CTkLabel(app, text="Username:")
label_username.pack()
entry_username = ctk.CTkEntry(app)
entry_username.pack(pady=5)

label_password = ctk.CTkLabel(app, text="Password:")
label_password.pack()
entry_password = ctk.CTkEntry(app, show="*")
entry_password.pack(pady=5)

label_role = ctk.CTkLabel(app, text="Role:")
label_role.pack()
role_var = ctk.StringVar(value="Admin")
role_options = ctk.CTkComboBox(app, variable=role_var, values=["Admin", "Doctor",
"Pharmacist"])
role_options.pack(pady=5)

label_message = ctk.CTkLabel(app, text="")
label_message.pack()

button_login = ctk.CTkButton(app, text="Login", command=login)
button_login.pack(pady=10)

button_register = ctk.CTkButton(app, text="Register", command=open_register_window)
button_register.pack(pady=10)

app.mainloop()
```

MYSQL QUERIES

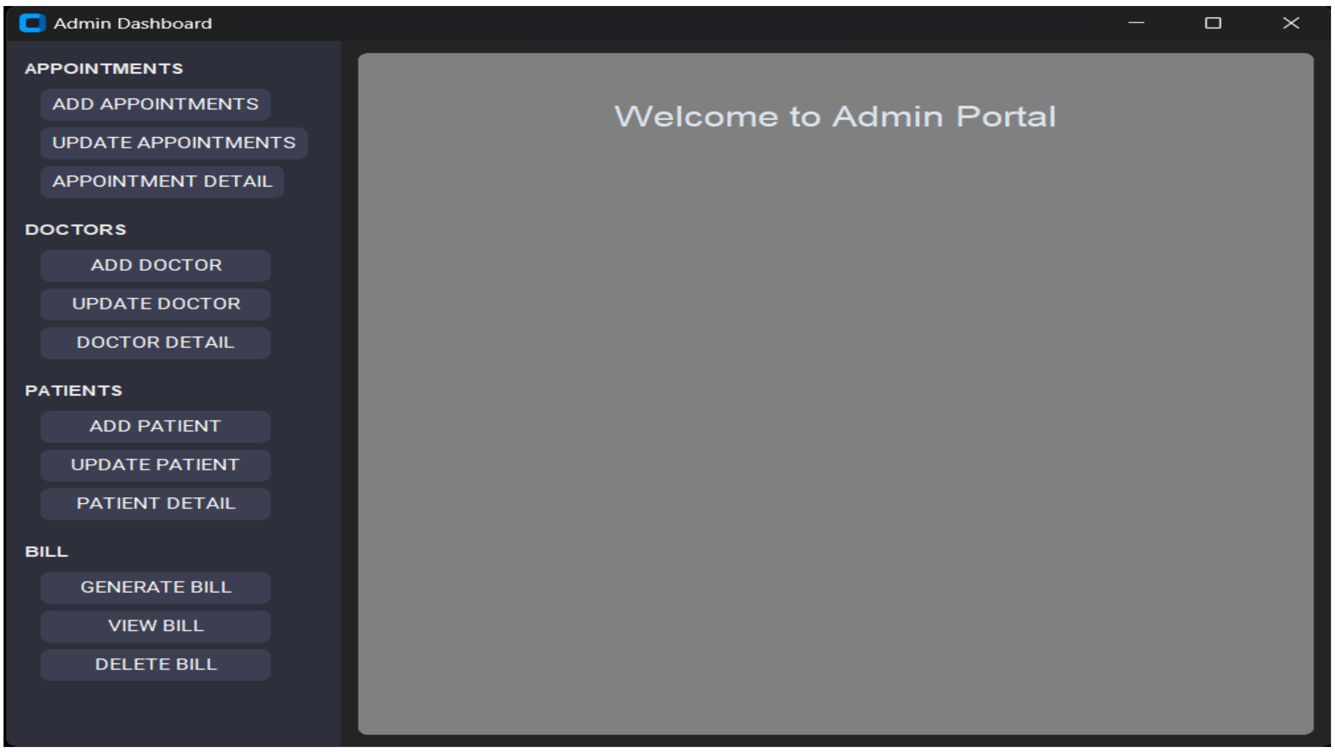
```
-- Create database
CREATE DATABASE IF NOT EXISTS user_db;
USE user_db;

-- Table for storing user information (for login)
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('Admin', 'Doctor', 'Pharmacist') NOT NULL
);

-- Table for storing doctor information
CREATE TABLE IF NOT EXISTS doctors (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    specialization VARCHAR(100),
    phone VARCHAR(15),
    email VARCHAR(100) UNIQUE
);

-- Table for storing patient information
CREATE TABLE IF NOT EXISTS patients (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    date_of_birth DATE,
    gender ENUM('Male', 'Female', 'Other'),
    phone VARCHAR(15),
    email VARCHAR(100) UNIQUE
);

-- Table for storing appointments (including patient_name and doctor_name)
CREATE TABLE IF NOT EXISTS appointments (
    id INT AUTO_INCREMENT PRIMARY KEY,
    patient_name VARCHAR(100) NOT NULL,
    doctor_name VARCHAR(100) NOT NULL,
    date NOT NULL,
    time NOT NULL,
    FOREIGN KEY (patient_name) REFERENCES patients(name) ON DELETE CASCADE,
    FOREIGN KEY (doctor_name) REFERENCES doctors(name) ON DELETE CASCADE
);

-- Table for storing billing information
CREATE TABLE IF NOT EXISTS bills (
    id INT AUTO_INCREMENT PRIMARY KEY,
    patient_name VARCHAR(100) NOT NULL,
    appointment_id INT NOT NULL,
    amount DECIMAL(10, 2) NOT NULL,
    payment_status ENUM('Paid', 'Unpaid') DEFAULT 'Unpaid',
        date NOT NULL,
        FOREIGN KEY (patient_name) REFERENCES patients(name) ON DELETE CASCADE,
        FOREIGN KEY (appointment_id) REFERENCES appointments(id) ON DELETE CASCADE
    );
```

PROJECT SCREENSHOTS

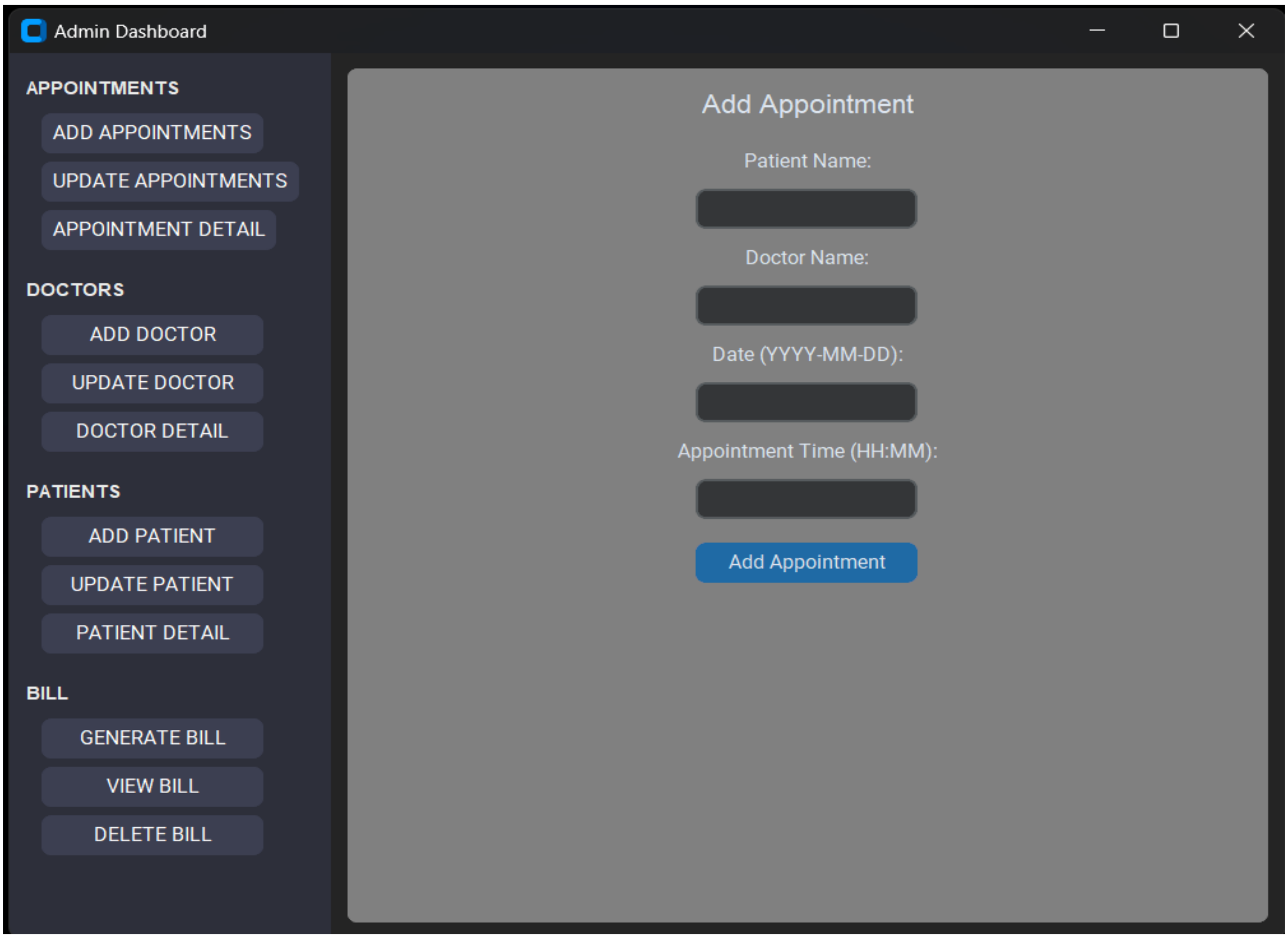
LOGIN



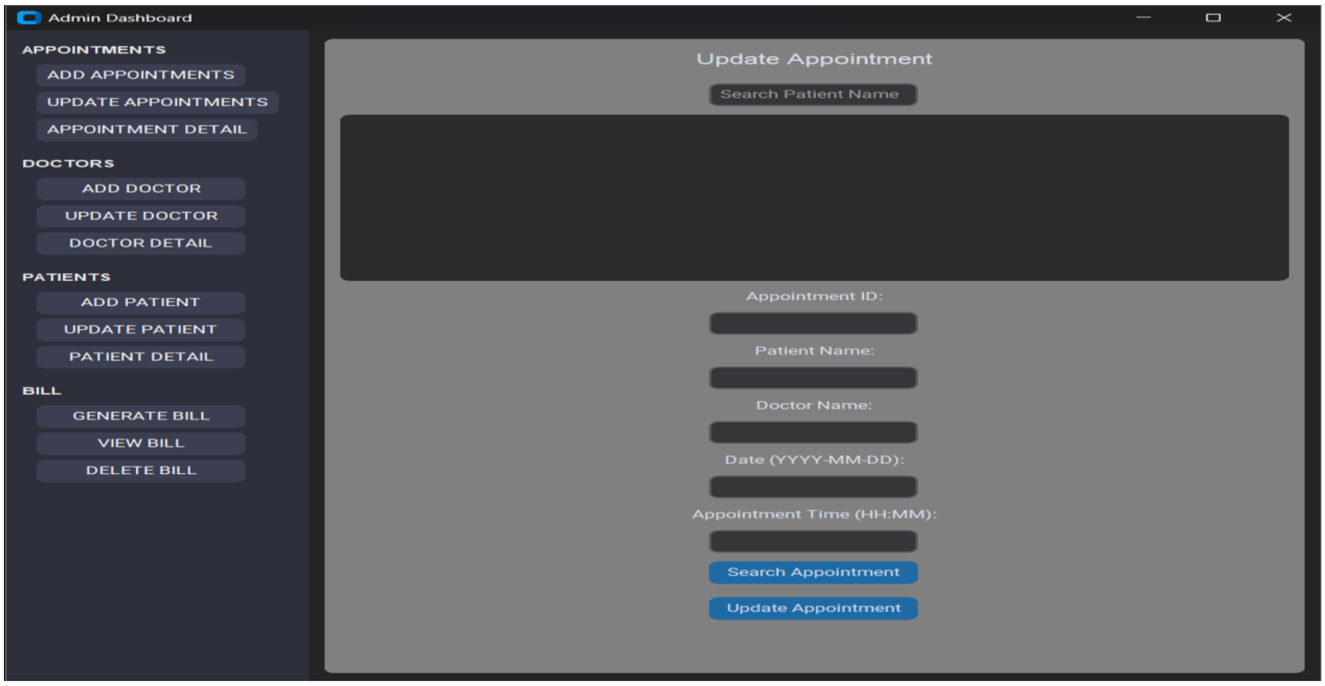
ADMIN DASHBOARD



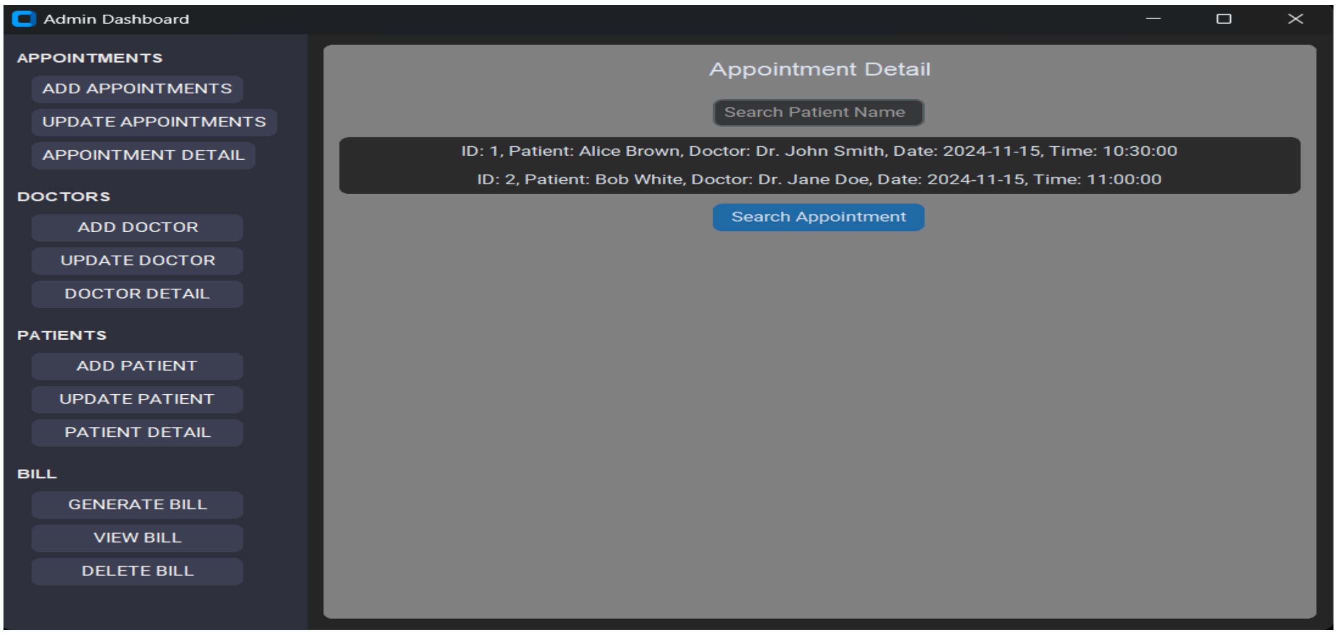
ADD APPOINTMENT



UPDATE APPOINTMENT

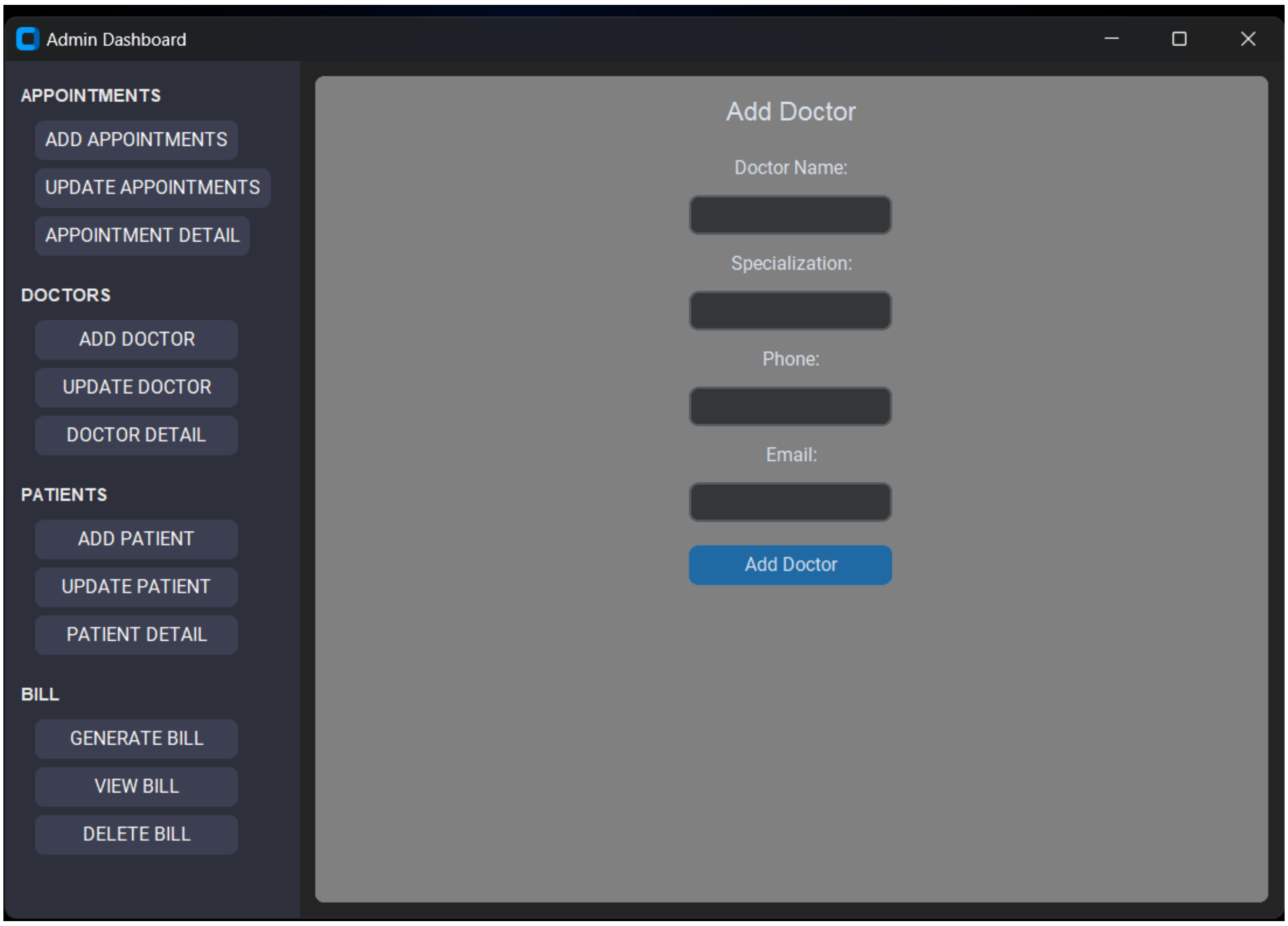


APPOINTMENT DETAIL

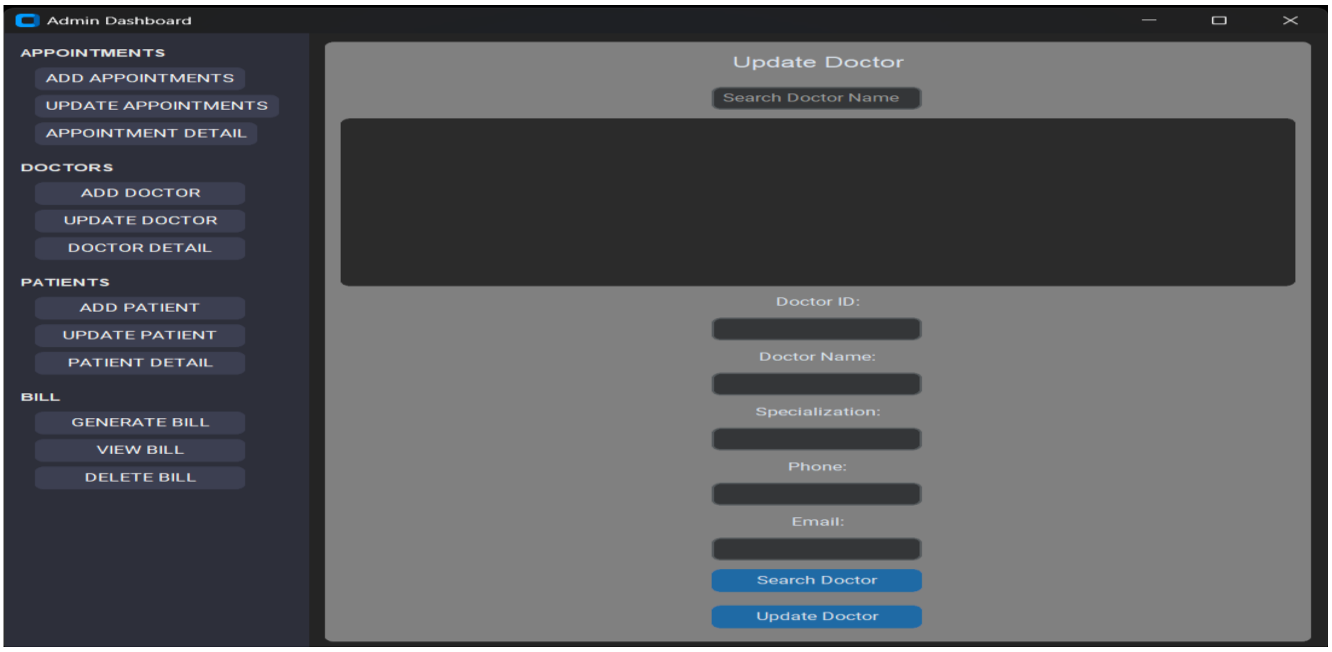


ID: 1, Patient: Alice Brown, Doctor: Dr. John Smith, Date: 2024-11-15, Time: 10:30:00
ID: 2, Patient: Bob White, Doctor: Dr. Jane Doe, Date: 2024-11-15, Time: 11:00:00

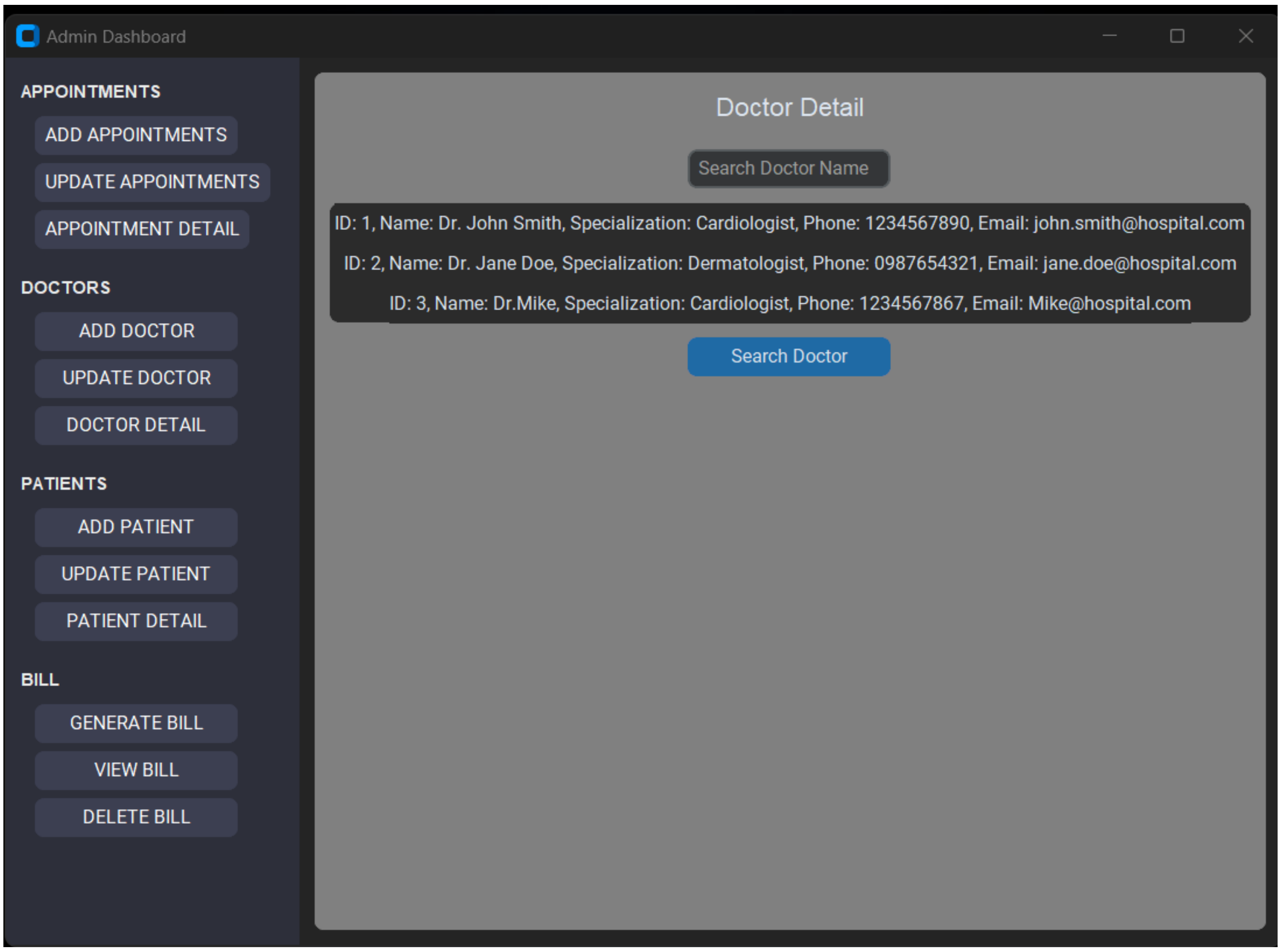
ADD DOCTOR



UPDATE DOCTOR

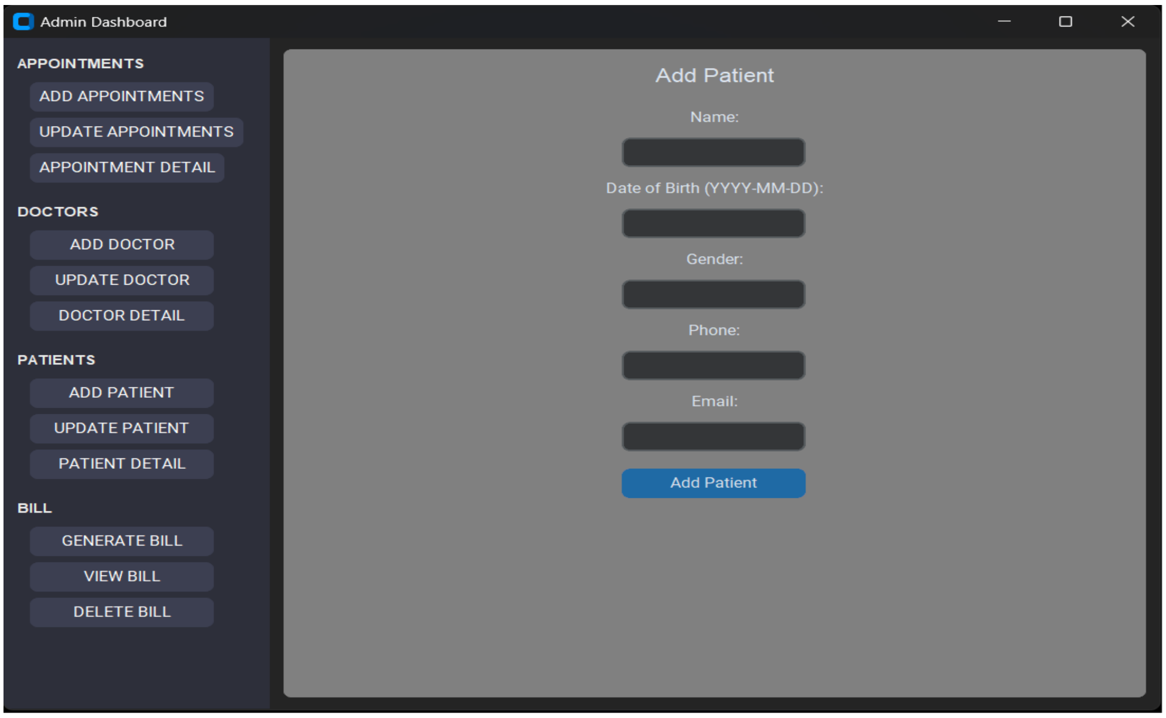


DOCTOR DETAIL

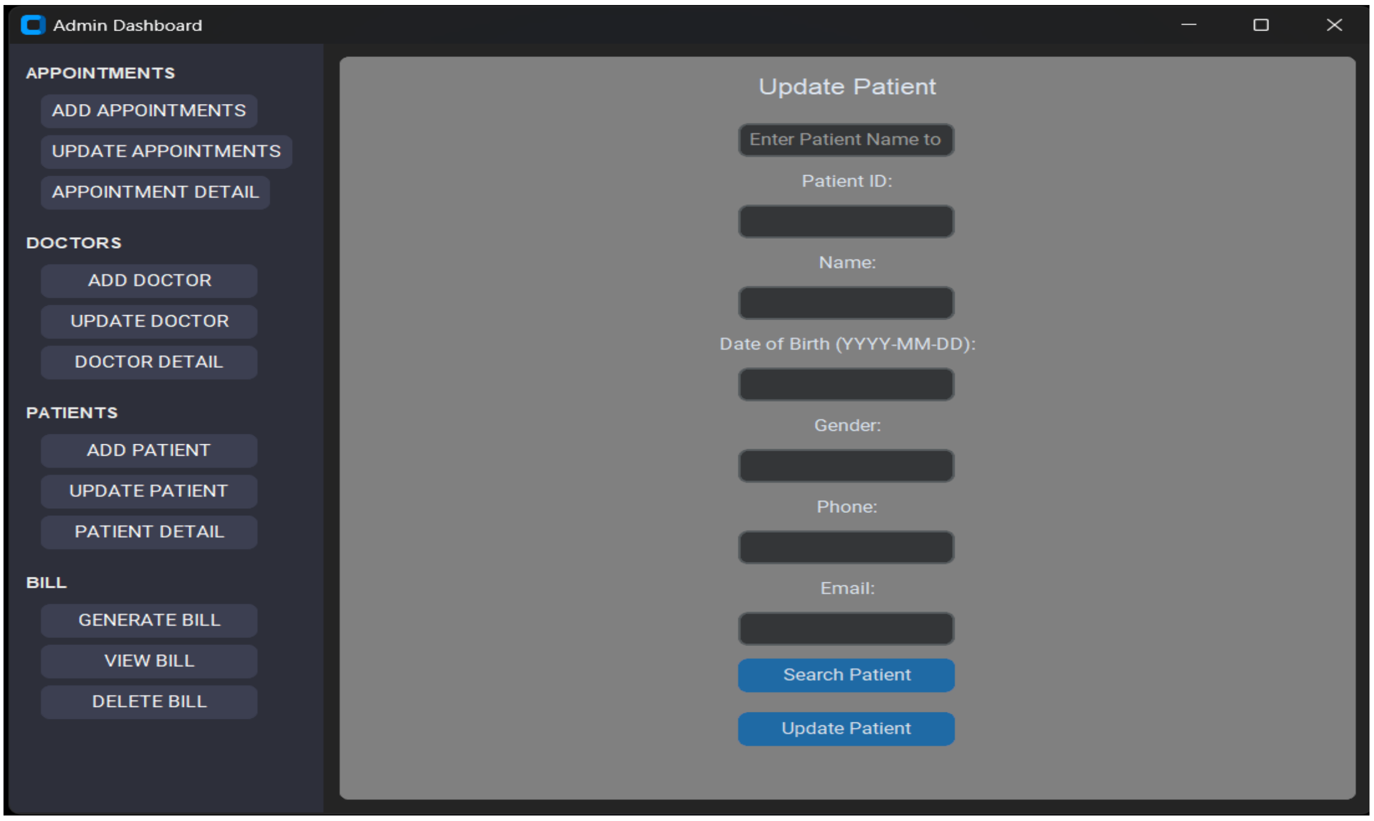


```
ID: 1, Name: Dr. John Smith, Specialization: Cardiologist, Phone: 1234567890, Email: john.smith@hospital.com
ID: 2, Name: Dr. Jane Doe, Specialization: Dermatologist, Phone: 0987654321, Email: jane.doe@hospital.com
ID: 3, Name: Dr.Mike, Specialization: Cardiologist, Phone: 1234567867, Email: Mike@hospital.com
```

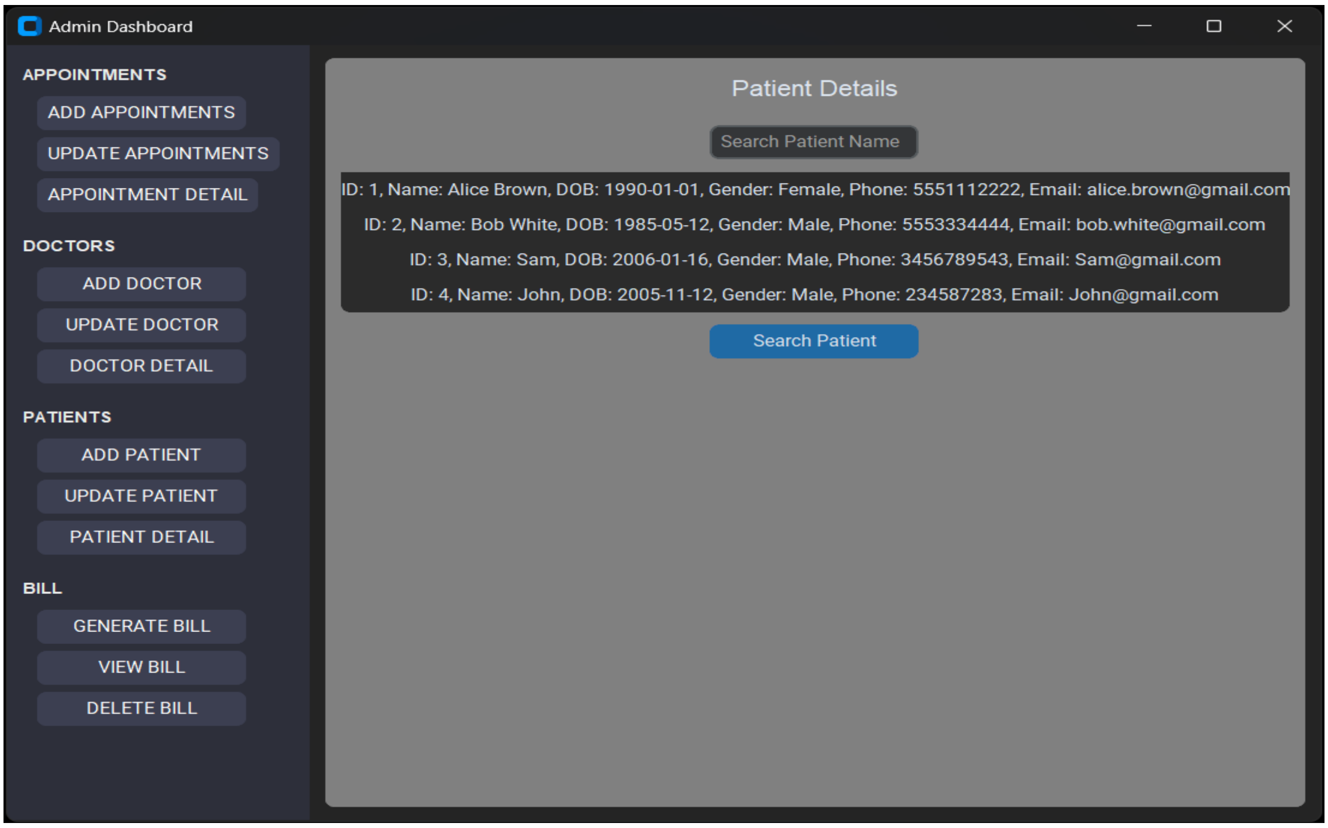
ADD PATIENT



UPDATE PATIENT

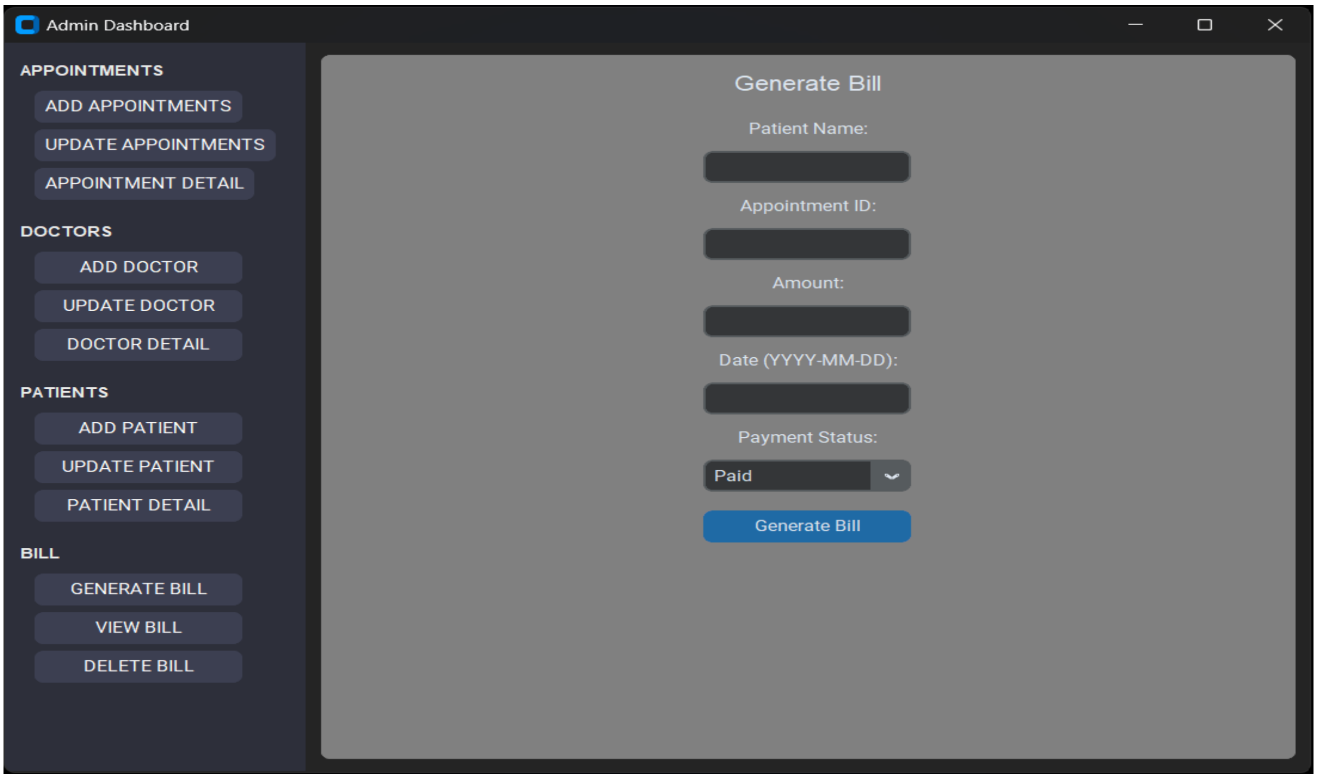


PATIENT DETAIL



ID: 1, Name: Alice Brown, DOB: 1990-01-01, Gender: Female, Phone: 5551112222, Email: alice.brown@gmail.com
ID: 2, Name: Bob White, DOB: 1985-05-12, Gender: Male, Phone: 5553334444, Email: bob.white@gmail.com
ID: 3, Name: Sam, DOB: 2006-01-16, Gender: Male, Phone: 3456789543, Email: Sam@gmail.com
ID: 4, Name: John, DOB: 2005-11-12, Gender: Male, Phone: 234587283, Email: John@gmail.com

GENERATE BILL



Patient Name:

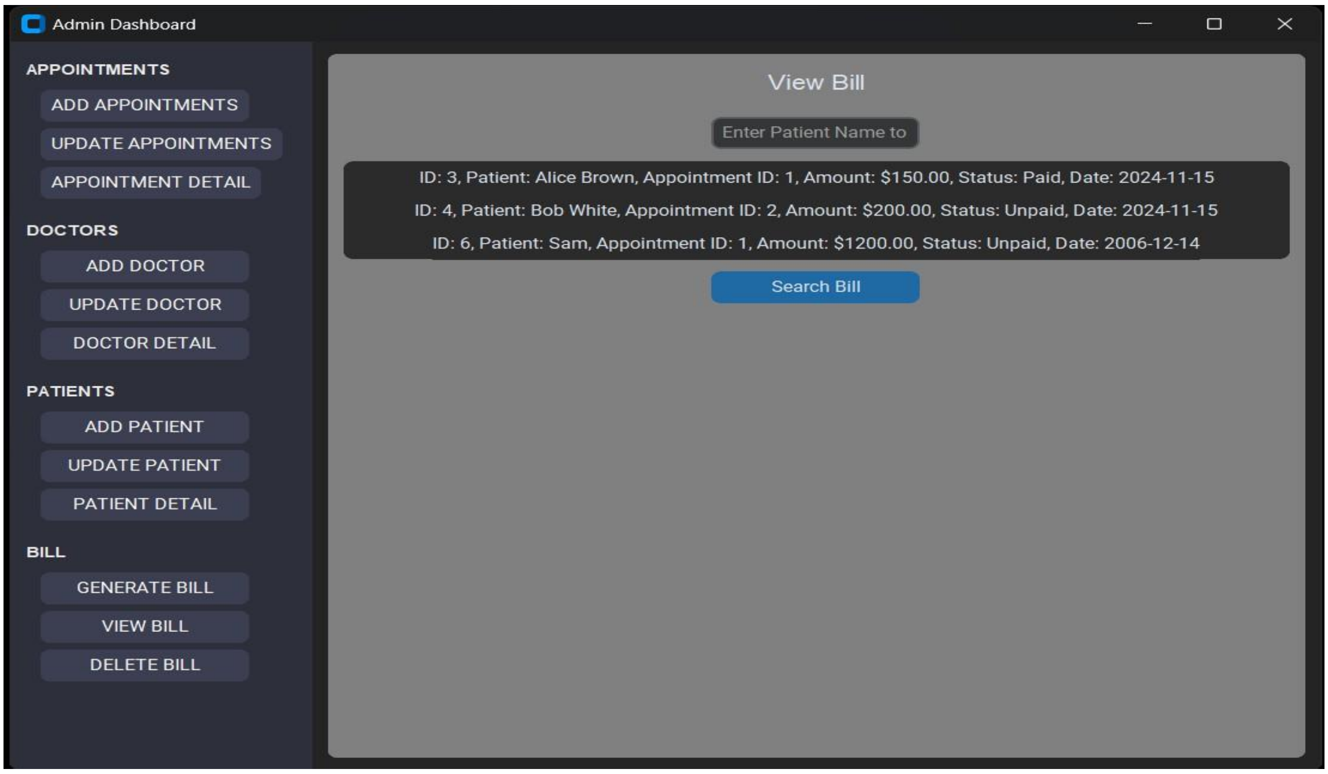
Appointment ID:

Amount:

Date (YYYY-MM-DD):

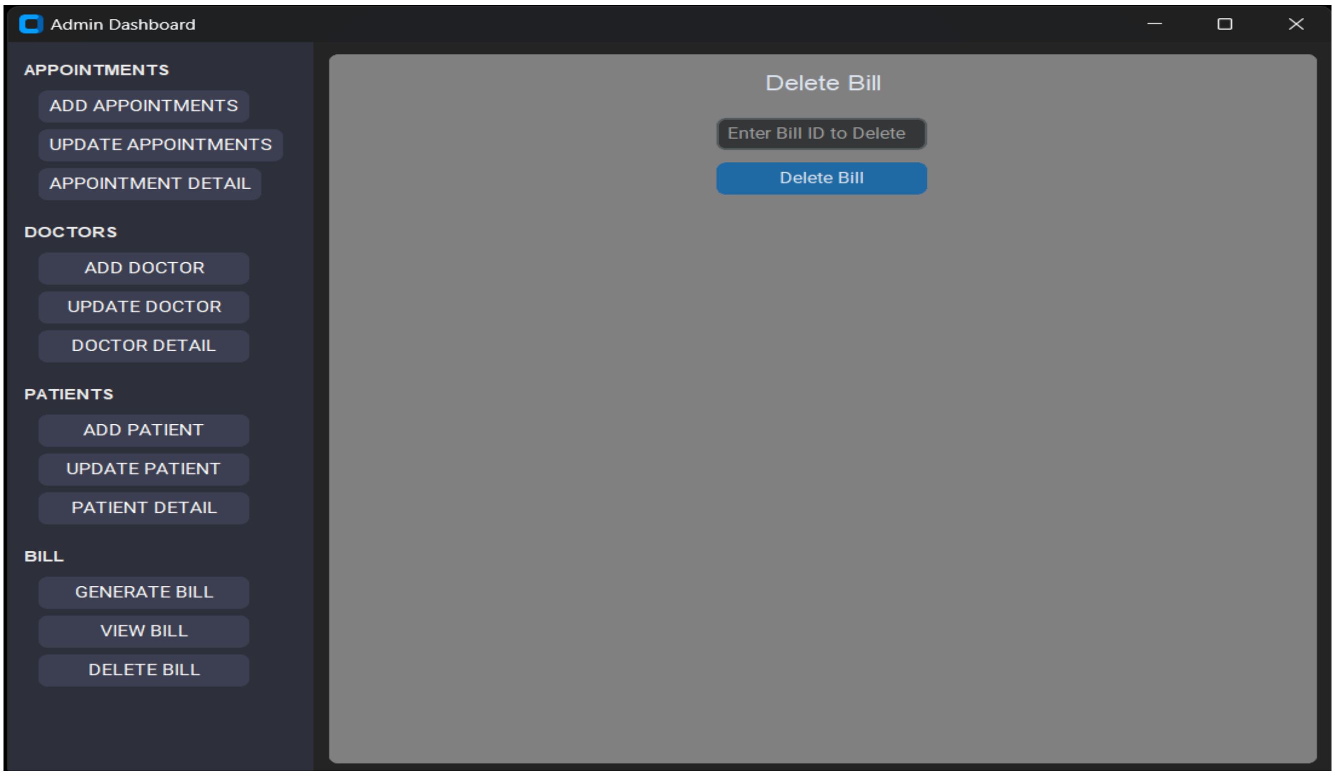
Payment Status: Paid

VIEW BILL



```
ID: 3, Patient: Alice Brown, Appointment ID: 1, Amount: $150.00, Status: Paid, Date: 2024-11-15
ID: 4, Patient: Bob White, Appointment ID: 2, Amount: $200.00, Status: Unpaid, Date: 2024-11-15
ID: 6, Patient: Sam, Appointment ID: 1, Amount: $1200.00, Status: Unpaid, Date: 2006-12-14
```

DELETE BILL



5. RESULTS AND DISCUSSION

5.1 Observations

- The system efficiently manages patient records, doctor assignments, and billing operations with accurate timestamps.
- The GUI offers a user-friendly experience, allowing admins and doctors to navigate through patient information, appointments, and billing records effectively.
- Data integrity is maintained with SQL constraints, ensuring consistency across patient and billing records.
- Role-based access provides secure and distinct functionalities for admins and doctors.

5.2 Limitations

- Limited scalability for larger hospital networks without further optimization.
- Real-time data processing may slow down with a high volume of concurrent users.
- The system relies on manual data entry, which may lead to occasional input errors.

5.3 Future Improvements

- **Mobile App Integration**: Develop a mobile app to allow remote access to patient and billing records.
- **Cloud Database**: Transition to a cloud-based database for enhanced scalability and remote access.
- **Automated Data Entry**: Incorporate scanning tools to reduce manual entry and minimize errors.

6. CONCLUSION

The Hospital Management System effectively centralizes core hospital operations, providing a reliable, user-friendly platform for managing patient information, doctor assignments, and billing. Leveraging SQL for database management, the system ensures data integrity, security, and real-time accessibility for authorized users. The role-based structure enhances operational efficiency, allowing admins and doctors to perform specific tasks seamlessly while safeguarding sensitive information. Through a well-designed GUI, the system offers an intuitive experience that streamlines interactions and reduces manual workload. This project demonstrates the successful integration of database management principles into healthcare administration, showcasing potential for further expansion and scalability.

7. REFERENCES

1. Tkinter Documentation. *Python Standard Library*, Python Software Foundation, 2023. Available at: https://docs.python.org/3/library/tkinter.html

2. Oracle MySQL Documentation. *MySQL 8.0 Reference Manual*, Oracle Corporation, 2023. Available at: https://dev.mysql.com/doc/

3. Rossum, G. van, *Python Programming Language*, Python Software Foundation, 2023. Available at: https://www.python.org/doc/

4. Simform, "How to Build a Hospital Management System," 2023. Available at: https://www.simform.com/

5. Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Boston: Pearson.

6. https://www.researchgate.net/publication/367460409_The_Hospital_Management_System

7. https://journals.indexcopernicus.com/api/file/viewByFileId/651165.pdf

8. https://github.com/GokulakkannanP/Hospital-management-system-DBMS-mini-project

9. https://github.com/GokulakrishnanAIML-A/Java/tree/main/dbms%20output

10. https://github.com/Gunavazhagan-B/Hospital-Management-System

11. https://github.com/HarishM10/Hospital-Management-System