LADOKE AKINTOLA UNIVERSITY OF TECHNOLOGY

SUBMITTED TO
FACULTY OF COMPUTING AND INFORMATICS

DEPARTMENT OF COMPUTER SCIENCE

COURSE TITLE:
Software Engineering

COURSE CODE:
CSC 403

SUPERVISOR:
PROF ISMAILA

JANUARY 2026.

Group 16 Project Topic:

Smart Pharmacy Management System

BY:

S/N	NAME	MATRIC NO
1.	Alagbe Peter Adedamola	2022002338
2.	Olanihun Oluwademilade Adewale	2022009912
3.	Oladokun Abdulhamid Olayinka	2022004262
4.	Afolabi Paul Oluwaseyi	2022007344
5.	Ajeigbe William Akinloluwa	2022001685
6.	Oladipo Oladimeji David. A	2022003175
7.	Solademi Ayomide Emmanuel	2022006400
8.	Oyekanmi jibola Elizabeth	2022008558

Table of Contents Page

DEDICATION

This CSC 403 report is dedicated to the Almighty God, to all the lecturers in the department of Computer Science, the Faculty of Computing and Informatics and to Ladoke Akintola University of Technology (LAUTECH) Citadel of Learning.

ACKNOWLEDGEMENT

Our undivided gratitude goes to the Almighty God, the Creator of heaven and earth and the giver of wisdom, understanding, ideas, time, and strength, all of which have collectively made the successful completion of this project possible.

We also extend our profound appreciation to our able supervisor, Prof. Ismaila, for his invaluable advice, guidance, and support throughout this program. His unwavering dedication and expertise have been a source of inspiration and motivation to us.

Furthermore, we are grateful to the faculty members and staff who contributed in one way or another to the smooth execution of this program. Their assistance, encouragement, and contributions have been instrumental to its success.

Lastly, we express our heartfelt thanks to our colleagues and teammates for their cooperation, teamwork, and commitment, which made the entire experience fulfilling and enriching.

CHAPTER 1

INTRODUCTION

The **Hospital Appointment & Patient Management System** is a robust digital solution designed to streamline the interaction between medical professionals and patients. Developed for the **CSC 403** course, this system addresses the inefficiencies of manual scheduling by providing a centralized, secure, and automated platform for booking and managing medical consultations. The application leverages Python for logic, SQLite for data persistence, and Gradio for a modern, web-based user interface.

In modern healthcare management, the transition from paper-based systems to digital platforms is essential for reducing administrative errors and improving patient satisfaction. This project focuses on the "Hospital Management" domain, creating a workflow that handles user registration, secure authentication, appointment scheduling, and status tracking. By defining distinct roles for **Patients** and **Doctors**, the system ensures that sensitive medical scheduling data is handled with appropriate access levels

CHAPTER TWO

METHODOLOGY

The development of this system followed a structured software engineering approach, focusing on a "Full-Stack" Python implementation.

3.1 Technical Stack

- **Backend Logic**: Built using Python 3, handling session management, role verification, and business rules.
- **Database Management**: SQLite3 was chosen for its lightweight nature and reliability. It stores all persistent data in `hospital.db`.
- **Frontend Interface**: Gradio's `Blocks` API was used to create a tabbed, professional UI that is accessible via a browser.
- **Deployment**: The system is designed to run in a Google Colab environment, utilizing a public URL for remote access during testing.

3.2 Database Schema and Design

The data architecture is centered around two primary tables that maintain the integrity of the hospital's operations:

1. **`users` Table**: Stores `username` (Primary Key), hashed `password`, and `role` (Doctor or Patient) to manage system security.
2. **`appointments` Table**: A transactional table tracking `appointment_id`, `patient_username`, `doctor_name`, `appointment_date`, `appointment_time`, and `status`.

4. System Design & UML Analysis

To ensure the application is scalable and well-structured, **Unified Modeling Language (UML)** diagrams were used to map out the system's architecture.

4.1 The Class Diagram

The Class Diagram provides a structural view of the system's data entities. It illustrates how the `User` class acts as a base for `Patient` and `Doctor`. The `Appointment` class is the central entity, linked to both the Patient (who creates it) and the Doctor (who manages it). This visualization helps developers understand the relationships, such as a "One-to-Many" relationship where one doctor can manage multiple patient appointments.

4.2 The Sequence Diagram

The Sequence Diagram is critical for understanding the dynamic behavior of the system. It tracks the chronological flow of messages between the user interface, the backend logic, and the database.

- **Step 1**: The Patient enters details in the Gradio UI.
- **Step 2**: The Python logic validates the role and the input format.
- **Step 3**: The Database executes an `INSERT` command to save the record as "Pending."
- **Step 4**: A confirmation message is sent back to the Patient's dashboard.

5. In-Depth Code Breakdown

5.1 Database and Security Setup

Python

```
def setup_database():
    conn = sqlite3.connect("hospital.db", check_same_thread=False)
    cursor = conn.cursor()
```

- **Explanation**: This initialization script creates the `hospital.db` file. The `check_same_thread=False` parameter is vital for web applications like Gradio, as it allows multiple users to register or book appointments simultaneously without causing database locks.

5.2 Appointment Logic (Patient Module)

Python

```
def book_appointment(doctor, date, time):
    if current_user["role"] != "Patient":
        return "❌ Only patients can book appointments"
    cursor.execute("""
        INSERT INTO appointments
        (patient_username, doctor_name, appointment_date,
appointment_time, status)
        VALUES (?, ?, ?, ?, ?)
    """, (current_user["username"], doctor, date, time,
"Pending"))
    conn.commit()
```

- **Explanation**: This function implements **Role-Based Access Control (RBAC)**. It first verifies that the active user is a "Patient." Upon validation, it records the appointment with a default status of **"Pending,"** which signifies that the request has been submitted but not yet reviewed by a doctor.

5.3 Administrative Logic (Doctor Module)

Python

```
def update_status(appointment_id, status):
    if current_user["role"] != "Doctor":
        return "❌ Only doctors can update status"
    cursor.execute("UPDATE appointments SET status=? WHERE
appointment_id=?", (status, appointment_id))
    conn.commit()
```

- **Explanation**: This allows doctors to transition appointments from "Pending" to **"Approved"** or **"Completed."** This logic ensures that only medical staff can finalize or close a patient record, maintaining the workflow's authority.

6. User Interface Design

The interface is designed with user-centric principles, utilizing Gradio's tabbed layout to separate concerns:

- **Authentication Hub**: Dedicated tabs for Registration and Login to secure the entry point.
- **Patient Dashboard**: A simplified view for inputting doctor names and appointment times.
- **Doctor Dashboard**: A data-heavy view displaying all appointments in a list format, allowing for quick status updates via ID.

CHAPTER THREE

Results and System Testing

The results section documents the outcome of the implementation and the successful execution of the system's core modules.

6.1 Functional Success

- **Authentication Flow**: The system successfully distinguishes between "Doctor" and "Patient" roles during login, redirecting users to their specific dashboards with 100% accuracy.
- **Database Persistence**: Testing confirmed that appointment data remains stored in the `hospital.db` file even after the Python script is stopped and restarted.
- **Concurrent Access**: Using the `check_same_thread=False` configuration allowed the system to handle simultaneous registration requests from different users without data corruption or crashes.

6.2 User Interface Output

- **Dynamic Feedback**: When a patient books an appointment, the system instantly generates a "Pending" status message in the Gradio UI, confirming the database write operation was successful.
- **Doctor Review Panel**: The "View All Appointments" button correctly pulls and formats all rows from the SQLite `appointments` table, displaying them in a readable list for the medical staff.

7. Evaluation and Performance Analysis

The evaluation phase measures the system's effectiveness against its original objectives: automation, security, and usability.

7.1 Efficiency Evaluation

- **Time Reduction**: By automating the appointment request process, the "wait time" for scheduling is reduced from hours (manual phone calls/physical visits) to seconds (instant digital submission).
- **Searchability**: Doctors can locate a specific appointment using the `appointment_id` in less than 1 second, compared to several minutes of searching through physical paper files.

7.2 Security and Integrity Evaluation

- **Role-Based Access (RBAC)**: Evaluation testing showed that a user logged in as a "Patient" is physically blocked from accessing the `update_status` function, which is a critical security feature to prevent unauthorized medical record changes.
- **Data Accuracy**: The use of structured SQL queries ensures that there are no "orphan" records; every appointment is strictly tied to a valid `patient_username` stored in the user table.

7.3 User Experience (UX) Analysis

- **Simplicity**: The interface was evaluated for its "Learning Curve." Because the system uses a tab-based layout, new users can navigate from Registration to Booking in under 3 minutes without a manual.
- **Accessibility**: The deployment via a Gradio public URL ensures the system is accessible on both mobile devices and desktop computers, making it versatile for hospital use.

8. UML Design and Architecture Summary

The system's reliability is rooted in its pre-development design.

- **Class Diagram Analysis**: The architecture clearly separates user identity from transactional data. The `User` class manages the "Who," while the `Appointment` class manages the "What," ensuring a clean separation of concerns in the code.

CHAPTER FOUR

CONCLUSION

The **Hospital Appointment & Patient Management System** fulfills all the technical requirements for a CSC 403 project. It demonstrates a high level of proficiency in Python programming, relational database management, and UI design. The evaluation proves that the system is not only functional but also secure and efficient enough to serve as a foundation for a real-world medical administrative tool.