LADOKE AKINTOLA UNIVERSITY TECHNOLOGY

SUBMITTED TO
FACULTY OF COMPUTING AND INFORMATICS
DEPARTMENT OF COMPUTER SCIENCE
GROUP 6

COURSE TITLE:
Software Engineering

COURSE CODE:
CSC 403

SUPERVISOR:
PROF ISMAILA

JANUARY 2026.

Project Topic:
Smart Course Registration & Advising System

BY:

S/N	NAME	MATRIC NO
1.	Azeez Abeebllahi Adedolapo	2022005001
2.	Adegbindin Abdullahi Adewole	2022002145
3.	Adetunji Ismail Ademola	2022006119
4.	Abisoye Victor Oyetayo	2022010479
5.	Inaboya Samuel Boluwatife	2022009628
6.	Owolabi Zainab Dolapo	2022003940
7.	Akinwale Abdulmalik Olayide	2022003018
8.	Oyadiran Samuel Olawale	2022008305

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 **Smart Course Registration & Advising System** is a digital management platform designed to automate university course enrollment and academic advising. Traditional registration processes often suffer from manual errors, such as students enrolling in advanced courses without completing the necessary foundations.

This system solves these issues by:

1. Implementing **Role-Based Access Control (RBAC)** for Students and Advisors.
2. Enforcing **Automated Prerequisite Checks** to ensure academic integrity.
3. Providing a **Centralized Database** for course catalogs and registration statuses.
4. Offering a user-friendly interface for real-time interaction between students and faculty.

CHAPTER TWO

METHODOLOGY

The system is built using a three-tier architecture: **User Interface (Gradio)**, **Application Logic (Python)**, and **Database (SQLite)**.

Component	Technology	Purpose
Interface	Gradio	Creates the web-based dashboard for users.
Database	SQLite3	Stores user credentials, course details, and enrollment logs.
Logic	Python 3	Handles session management and prerequisite validation.
Environment	Google Colab	Provides the execution environment and public URL hosting.

2.2 Database Schema

The database (`course_system.db`) consists of three relational tables:

1. **`users`**: Manages authentication (Username, Password, Role).
2. **`courses`**: The academic catalog (Code, Title, Prerequisite).
3. **`registrations`**: Tracks student requests (Student Name, Course Code, Status: Pending/Approved).

3. Comprehensive Line-by-Line Code Explanation

3.1 Initialization and Setup

3. Comprehensive Line-by-Line Code Explanation

3.1 Initialization and Setup

Python
```
import gradio as gr
import sqlite3
```

- **Explanation**: Imports the necessary libraries for the UI and the database.

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

- **Explanation**: Establishes a connection to the database. `check_same_thread=False` is critical because Gradio's multi-threaded nature requires multiple threads to access the SQLite connection simultaneously.

Python
```
   cursor.execute("CREATE TABLE IF NOT EXISTS users (...)")
    cursor.execute("CREATE TABLE IF NOT EXISTS courses (...)")
    cursor.execute("CREATE TABLE IF NOT EXISTS registrations (...)")
```

- **Explanation**: These lines initialize the tables. The `registrations` table is the "link" between users and courses.

3.2 Session Management

Python
```
current_user = {"username": None, "role": None}
```

- **Explanation**: This global dictionary acts as a temporary session storage. It tracks who is currently using the app and what their "Role" is (Student vs. Advisor).

3.3 Core Logic Functions

Prerequisite Validation (The "Smart" Logic)

Python

```
def register_course(course_code):
    ...
    prerequisite = course[0]
    if prerequisite:
        cursor.execute("SELECT * FROM registrations WHERE student=? AND course_code=? AND status='Approved'",
                       (current_user["username"], prerequisite))
        if not cursor.fetchone():
            return f"Prerequisite {prerequisite} not satisfied"
```

- **Explanation**: This is the most important part of the code. Before a student can register, the system checks if the course has a prerequisite. If it does, it queries the database to see if the student has an "Approved" status for that specific prerequisite. If not, the registration is blocked.

Advisor Controls

Python

```
def add_course(code, title, prerequisite):
    if current_user["role"] != "Advisor":
        return "Only advisors can add courses"
```

- **Explanation**: Implements a security check. Even if the UI shows the "Add Course" tab to a student, the backend logic will reject the request if the role is not "Advisor."

3.4 The Gradio Interface (UI)

Python

```
with gr.Blocks(theme=gr.themes.Soft(), title="Smart Course Registration System") as app:
```

- **Explanation**: Sets up the visual container with a "Soft" theme.

Python
```
gr.Button("Login").click(lambda x, y: (login_user(x, y), user_status()), [u, p], [o, status])
```

- **Explanation**: Uses a `lambda` function to trigger two events with one click: logging the user in and updating the "User Status" markdown at the top of the page.

CHAPTER THREE

RESULTS AND EVALUATION

Upon execution in Google Colab, the system generates a local and public URL.

1. **Advisor Perspective**: The advisor can populate the database with courses. For example, they can add "CSC101" and then add "CSC201" with "CSC101" as a prerequisite.
2. **Student Perspective**: A student attempting to register for "CSC201" will be rejected until they register for and are "Approved" in "CSC101."
3. **Workflow**:
    a. Student Registers Status is "Pending".
    b. Advisor Reviews Approves student.
    c. Student is now cleared for higher-level courses.

CHAPTER FOUR

CONCLUSION AND UML DIAGRAM ANALYSIS

The **Smart Course Registration & Advising System** demonstrates the effective integration of relational databases with interactive web interfaces to solve complex administrative challenges.

4.1 The Role of UML Diagrams in This Project

- To maintain the system's integrity and facilitate future scaling, **Unified Modeling Language (UML)** diagrams are essential components of the project's documentation:
- **Use Case Diagram**: This diagram would illustrate the system's boundaries, showing "Student" and "Advisor" as actors. It defines their interactions, such as the student "Registering for Course" and the advisor "Approving Request".
- **Class Diagram**: This provides a structural view of the system's data model. It would define the `User`, `Course`, and `Registration` classes, their attributes (e.g., `course_code`, `prerequisite`), and their relationships (e.g., a "One-to-Many" relationship between a student and their registrations).
- **Sequence Diagram**: This is vital for visualizing the "Prerequisite Check" logic. it would detail the chronological steps: the student sends a registration request, the logic layer queries the `courses` table for prerequisites, the database returns the student's history, and the system finally returns a success or error message.
- Integrating these UML perspectives ensures that the system logic remains transparent and that any developer joining the project can immediately understand the underlying data flow and rules.