CREATE DATABASE menu\_management;

USE menu\_management;

CREATE TABLE departments (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) UNIQUE NOT NULL

);

CREATE TABLE years (

id INT AUTO\_INCREMENT PRIMARY KEY,

department\_id INT,

year\_name VARCHAR(50) NOT NULL,

FOREIGN KEY (department\_id) REFERENCES departments(id) ON DELETE CASCADE

);

CREATE TABLE subjects (

id INT AUTO\_INCREMENT PRIMARY KEY,

year\_id INT,

subject\_name VARCHAR(255) NOT NULL,

FOREIGN KEY (year\_id) REFERENCES years(id) ON DELETE CASCADE

);

CREATE TABLE excel\_files (

id INT AUTO\_INCREMENT PRIMARY KEY,

subject\_id INT,

file\_name VARCHAR(255),

uploaded\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

file\_data LONGBLOB,

FOREIGN KEY (subject\_id) REFERENCES subjects(id) ON DELETE CASCADE

);

INSERT INTO departments (name) VALUES ('Computer Science'), ('Electrical'), ('Mechanical');

INSERT INTO years (department\_id, year\_name)

VALUES (1, 'First Year'), (1, 'Second Year'), (2, 'First Year'), (3, 'Final Year');

INSERT INTO subjects (year\_id, subject\_name)

VALUES (1, 'Data Structures'), (1, 'Python'), (2, 'DBMS'), (3, 'Circuit Theory');

import sys

import mysql.connector

import pandas as pd

import os

from PyQt6.QtWidgets import (

    QApplication, QMainWindow, QMenu, QInputDialog, QMessageBox, QFileDialog

)

from PyQt6.QtGui import QFont, QAction

from PyQt6.QtGui import QPalette, QBrush, QPixmap

from PyQt6.QtWidgets import (

    QApplication, QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout, QMessageBox, QFrame

)

from PyQt6.QtGui import QFont, QPixmap

from PyQt6.QtCore import Qt, QSize

import sys

# MySQL Database Connection

conn = mysql.connector.connect(

    host="localhost",

    user="root",

    password="Radha@1474",

    database="menu\_management"

)

cursor = conn.cursor()

# ✅ Credentials

CORRECT\_USERNAME = "admin"

CORRECT\_PASSWORD = "password123"

class LoginWindow(QWidget):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Question paper generator")

        self.resize(1024, 768)

        # ✅ Main Layout

        main\_layout = QVBoxLayout()

        main\_layout.setContentsMargins(0, 0, 0, 0)

        main\_layout.setSpacing(0)

        self.setLayout(main\_layout)

        # ✅ Create Background First

        self.background\_label = QLabel(self)

        self.background\_label.setScaledContents(True)

        # ✅ Card-like Login Frame

        login\_frame = QFrame(self)

        login\_frame.setStyleSheet("""

            background-color: white;

            border-radius: 12px;

            padding: 30px;

            border: 1px solid #ccc;

        """)

        frame\_layout = QVBoxLayout()

        frame\_layout.setSpacing(15)

        login\_frame.setLayout(frame\_layout)

        # ✅ Shop Name

        shop\_label = QLabel("        ADCET         ")

        shop\_label.setFont(QFont("Arial", 16, QFont.Weight.Bold))

        shop\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

        shop\_label.setStyleSheet("color: #222;")

        frame\_layout.addWidget(shop\_label)

        # ✅ Owner Name

        owner\_label = QLabel("COE")

        owner\_label.setFont(QFont("Arial", 12))

        owner\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

        owner\_label.setStyleSheet("color: #666;")

        frame\_layout.addWidget(owner\_label)

        # ✅ Username Input

        self.username\_input = QLineEdit()

        self.username\_input.setPlaceholderText("Enter Username")

        self.username\_input.setFont(QFont("Arial", 12))

        self.username\_input.setStyleSheet(self.input\_style())

        frame\_layout.addWidget(self.username\_input)

        # ✅ Password Input

        self.password\_input = QLineEdit()

        self.password\_input.setPlaceholderText("Enter Password")

        self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

        self.password\_input.setFont(QFont("Arial", 12))

        self.password\_input.setStyleSheet(self.input\_style())

        frame\_layout.addWidget(self.password\_input)

        # ✅ Login Button

        self.login\_button = QPushButton("Login")

        self.login\_button.setFont(QFont("Arial", 14, QFont.Weight.Bold))

        self.login\_button.setStyleSheet(self.button\_style())

        self.login\_button.clicked.connect(self.validate\_login)

        frame\_layout.addWidget(self.login\_button)

        # ✅ Add Login Frame to Main Layout

        main\_layout.addWidget(login\_frame, alignment=Qt.AlignmentFlag.AlignCenter)

        # ✅ Apply Background

        self.set\_background()

    def set\_background(self):

        """ Set a full-screen background image using QLabel. """

        background\_image = QPixmap("exam.jpg")  # Load background image

        if not background\_image.isNull():

            self.background\_label.setPixmap(background\_image.scaled(

                self.size(), Qt.AspectRatioMode.IgnoreAspectRatio, Qt.TransformationMode.SmoothTransformation

            ))

            self.background\_label.setGeometry(0, 0, self.width(), self.height())  # Fill screen

        # ✅ Ensure UI elements stay in front of the background

        self.background\_label.lower()

    def resizeEvent(self, event):

        """ Ensure background resizes dynamically when window is resized. """

        self.set\_background()

        super().resizeEvent(event)

    def input\_style(self):

        return """

            QLineEdit {

                border: 2px solid #007BFF;

                border-radius: 5px;

                padding: 8px;

                font-size: 14px;

                background-color: white;

            }

            QLineEdit:focus {

                border-color: #0056b3;

            }

        """

    def button\_style(self):

        return """

            QPushButton {

                background-color: #007BFF; color: white;

                border-radius: 8px; padding: 10px;

            }

            QPushButton:hover {

                background-color: #0056b3;

            }

        """

    def validate\_login(self):

        username = self.username\_input.text()

        password = self.password\_input.text()

        if username == CORRECT\_USERNAME and password == CORRECT\_PASSWORD:

            self.open\_dashboard()

        else:

            QMessageBox.warning(self, "Error", "Invalid Username or Password!")

    def open\_dashboard(self):

        self.dashboard = Dashboard()

        self.dashboard.show()

        self.close()

####################################################################################

class Dashboard(QMainWindow):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.setWindowTitle("Academic Dashboard")

        self.setGeometry(100, 100, 1000, 600)

        # ✅ Initialize background label

        self.background\_label = QLabel(self)

        self.set\_background()

        self.initUI()

    def initUI(self):

        self.menu\_bar = self.menuBar()

        self.menu\_bar.setFont(QFont("Arial", 12))

        # Admin Menu

        admin\_menu = self.menu\_bar.addMenu("Admin")

        add\_dept\_action = QAction("Add Department", self)

        add\_dept\_action.triggered.connect(self.add\_department)

        admin\_menu.addAction(add\_dept\_action)

        del\_dept\_action = QAction("Delete Department", self)

        del\_dept\_action.triggered.connect(self.delete\_department)

        admin\_menu.addAction(del\_dept\_action)

        self.load\_departments()

    def set\_background(self):

        """ Set a full-screen background image using QLabel. """

        background\_image = QPixmap("3.jpg")  # ✅ Replace with your actual image path

        if not background\_image.isNull():

            self.background\_label.setPixmap(background\_image.scaled(

                self.size(), Qt.AspectRatioMode.IgnoreAspectRatio, Qt.TransformationMode.SmoothTransformation

            ))

            self.background\_label.setGeometry(0, 0, self.width(), self.height())  # Fill screen

            self.background\_label.setScaledContents(True)  # ✅ Ensure image scales properly

        # ✅ Ensure UI elements stay in front of the background

        self.background\_label.lower()

    def resizeEvent(self, event):

        """ Ensure the background image resizes dynamically with the window. """

        self.set\_background()

        super().resizeEvent(event)

    def load\_departments(self):

        cursor.execute("SELECT id, name FROM departments")

        departments = cursor.fetchall()

        for dept\_id, dept\_name in departments:

            dept\_menu = self.menu\_bar.addMenu(dept\_name)

            self.load\_years(dept\_menu, dept\_id)

            add\_year\_action = QAction("Add Year", self)

            add\_year\_action.triggered.connect(lambda \_, d=dept\_id: self.add\_year(d))

            dept\_menu.addAction(add\_year\_action)

            del\_year\_action = QAction("Delete Year", self)

            del\_year\_action.triggered.connect(lambda \_, d=dept\_id: self.delete\_year(d))

            dept\_menu.addAction(del\_year\_action)

    def add\_department(self):

        """ Add a new department """

        dept\_name, ok = QInputDialog.getText(self, "Add Department", "Enter Department Name:")

        if ok and dept\_name:

            cursor.execute("INSERT INTO departments (name) VALUES (%s)", (dept\_name,))

            conn.commit()

            QMessageBox.information(self, "Success", "Department added successfully!")

            self.menu\_bar.clear()

            self.initUI()

    def delete\_department(self):

        """ Delete a department """

        cursor.execute("SELECT id, name FROM departments")

        departments = cursor.fetchall()

        dept\_names = [dept[1] for dept in departments]

        if not dept\_names:

            QMessageBox.warning(self, "Warning", "No departments available to delete.")

            return

        dept\_name, ok = QInputDialog.getItem(self, "Delete Department", "Select Department:", dept\_names, 0, False)

        if ok and dept\_name:

            dept\_id = next(dept[0] for dept in departments if dept[1] == dept\_name)

            cursor.execute("DELETE FROM departments WHERE id = %s", (dept\_id,))

            conn.commit()

            QMessageBox.information(self, "Success", "Department deleted successfully!")

            self.menu\_bar.clear()

            self.initUI()

    def load\_years(self, dept\_menu, dept\_id):

        cursor.execute("SELECT id, year\_name FROM years WHERE department\_id = %s", (dept\_id,))

        years = cursor.fetchall()

        for year\_id, year\_name in years:

            year\_menu = QMenu(year\_name, self)

            dept\_menu.addMenu(year\_menu)

            self.load\_subjects(year\_menu, year\_id)

            add\_subject\_action = QAction("Add Subject", self)

            add\_subject\_action.triggered.connect(lambda \_, y=year\_id: self.add\_subject(y))

            year\_menu.addAction(add\_subject\_action)

            del\_subject\_action = QAction("Delete Subject", self)

            del\_subject\_action.triggered.connect(lambda \_, y=year\_id: self.delete\_subject(y))

            year\_menu.addAction(del\_subject\_action)

    def add\_year(self, dept\_id):

        """ Add a new academic year to a department """

        year\_name, ok = QInputDialog.getText(self, "Add Year", "Enter Year Name:")

        if ok and year\_name:

            cursor.execute("SELECT COUNT(\*) FROM years WHERE department\_id = %s AND year\_name = %s", (dept\_id, year\_name))

            if cursor.fetchone()[0] > 0:

                QMessageBox.warning(self, "Error", "Year already exists in this department!")

            else:

                cursor.execute("INSERT INTO years (department\_id, year\_name) VALUES (%s, %s)", (dept\_id, year\_name))

                conn.commit()

                QMessageBox.information(self, "Success", "Year added successfully!")

                self.menu\_bar.clear()

                self.initUI()

    def delete\_year(self, dept\_id):

        """ Delete a year from a department """

        cursor.execute("SELECT id, year\_name FROM years WHERE department\_id = %s", (dept\_id,))

        years = cursor.fetchall()

        year\_names = [year[1] for year in years]

        if not year\_names:

            QMessageBox.warning(self, "Warning", "No years available to delete.")

            return

        year\_name, ok = QInputDialog.getItem(self, "Delete Year", "Select Year:", year\_names, 0, False)

        if ok and year\_name:

            year\_id = next(year[0] for year in years if year[1] == year\_name)

            cursor.execute("DELETE FROM years WHERE id = %s", (year\_id,))

            conn.commit()

            QMessageBox.information(self, "Success", "Year deleted successfully!")

            self.menu\_bar.clear()

            self.initUI()

    def load\_subjects(self, year\_menu, year\_id):

        cursor.execute("SELECT id, subject\_name FROM subjects WHERE year\_id = %s", (year\_id,))

        subjects = cursor.fetchall()

        for subj\_id, subj\_name in subjects:

            subject\_menu = QMenu(subj\_name, self)

            year\_menu.addMenu(subject\_menu)

            upload\_qb\_action = QAction("Upload Question Bank (Excel)", self)

            upload\_qb\_action.triggered.connect(lambda \_, s=subj\_id: self.upload\_question\_bank(s))

            subject\_menu.addAction(upload\_qb\_action)

            self.load\_uploaded\_files(subject\_menu, subj\_id)

    def add\_subject(self, year\_id):

        """ Add a subject to a year """

        subject\_name, ok = QInputDialog.getText(self, "Add Subject", "Enter Subject Name:")

        if ok and subject\_name:

            cursor.execute("SELECT COUNT(\*) FROM subjects WHERE year\_id = %s AND subject\_name = %s", (year\_id, subject\_name))

            if cursor.fetchone()[0] > 0:

                QMessageBox.warning(self, "Error", "Subject already exists for this year!")

            else:

                cursor.execute("INSERT INTO subjects (year\_id, subject\_name) VALUES (%s, %s)", (year\_id, subject\_name))

                conn.commit()

                QMessageBox.information(self, "Success", "Subject added successfully!")

                self.menu\_bar.clear()

                self.initUI()

    def delete\_subject(self, year\_id):

        """ Delete a subject from a year """

        cursor.execute("SELECT id, subject\_name FROM subjects WHERE year\_id = %s", (year\_id,))

        subjects = cursor.fetchall()

        subject\_names = [sub[1] for sub in subjects]

        if not subject\_names:

            QMessageBox.warning(self, "Warning", "No subjects available to delete.")

            return

        subject\_name, ok = QInputDialog.getItem(self, "Delete Subject", "Select Subject:", subject\_names, 0, False)

        if ok and subject\_name:

            subject\_id = next(sub[0] for sub in subjects if sub[1] == subject\_name)

            cursor.execute("DELETE FROM subjects WHERE id = %s", (subject\_id,))

            conn.commit()

            QMessageBox.information(self, "Success", "Subject deleted successfully!")

            self.menu\_bar.clear()

            self.initUI()

    def load\_uploaded\_files(self, subject\_menu, subject\_id):

        cursor.execute("SELECT id, file\_name FROM excel\_files WHERE subject\_id = %s", (subject\_id,))

        files = cursor.fetchall()

        for file\_id, file\_name in files:

            file\_menu = QMenu(file\_name, self)

            subject\_menu.addMenu(file\_menu)

            open\_file\_action = QAction("Open File", self)

            open\_file\_action.triggered.connect(lambda \_, f=file\_id: self.open\_uploaded\_file(f))

            file\_menu.addAction(open\_file\_action)

            del\_file\_action = QAction("Delete File", self)

            del\_file\_action.triggered.connect(lambda \_, f=file\_id: self.delete\_uploaded\_file(f))

            file\_menu.addAction(del\_file\_action)

    def upload\_question\_bank(self, subject\_id):

        file\_path, \_ = QFileDialog.getOpenFileName(

            self, "Open Question Bank", "",

            "Excel Files (\*.csv \*.xls \*.xlsx \*.xlsb \*.xlsm);;All Files (\*)"

        )

        if not file\_path:

            return

        try:

            with open(file\_path, "rb") as file:

                file\_data = file.read()

            cursor.execute(

                "INSERT INTO excel\_files (subject\_id, file\_name, file\_data) VALUES (%s, %s, %s)",

                (subject\_id, file\_path.split("/")[-1], file\_data)

            )

            conn.commit()

            QMessageBox.information(self, "Success", "Question bank uploaded successfully!")

            self.menu\_bar.clear()

            self.initUI()

        except Exception as e:

            QMessageBox.critical(self, "Error", f"Error uploading file: {str(e)}")

    def open\_uploaded\_file(self, file\_id):

        cursor.execute("SELECT file\_name, file\_data FROM excel\_files WHERE id = %s", (file\_id,))

        file = cursor.fetchone()

        if not file:

            QMessageBox.warning(self, "Error", "File not found in database.")

            return

        file\_name, file\_data = file

        file\_path = os.path.join(os.getcwd(), file\_name)

        try:

            with open(file\_path, "wb") as f:

                f.write(file\_data)

            os.system(f'start excel "{file\_path}"')  # Open in Excel

        except Exception as e:

            QMessageBox.critical(self, "Error", f"Could not open file: {str(e)}")

    def delete\_uploaded\_file(self, file\_id):

        cursor.execute("SELECT file\_name FROM excel\_files WHERE id = %s", (file\_id,))

        file = cursor.fetchone()

        if not file:

            QMessageBox.warning(self, "Error", "File not found in database.")

            return

        reply = QMessageBox.question(self, "Delete File", f"Are you sure you want to delete '{file[0]}'?", QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

        if reply == QMessageBox.StandardButton.Yes:

            cursor.execute("DELETE FROM excel\_files WHERE id = %s", (file\_id,))

            conn.commit()

            QMessageBox.information(self, "Success", "File deleted successfully!")

            self.menu\_bar.clear()

            self.initUI()

if \_\_name\_\_ == "\_\_main\_\_":

    app = QApplication(sys.argv)

    login\_window = LoginWindow()

    login\_window.show()

    login\_window.showMaximized()

    sys.exit(app.exec())