import wx

import cv2

import qrcode

import json

import datetime

import re

import hashlib

import secrets

import sqlite3

import os

from pathlib import Path

import csv

import sys

import shutil

from datetime import timedelta

import calendar

def get\_database\_path():

"""Get the appropriate database path for both development and bundled environments"""

# Detect if running from PyInstaller bundle

if getattr(sys, 'frozen', False):

base\_path = sys.\_MEIPASS

else:

base\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))

# Use writable path for the DB (AppData on Windows, ~/.config on Linux, ~/Library/Application Support on macOS)

if sys.platform == "win32":

appdata\_path = os.path.join(os.getenv('APPDATA'), 'SecureQRLoginSystem')

elif sys.platform == "darwin": # macOS

appdata\_path = os.path.join(os.path.expanduser('~'), 'Library', 'Application Support', 'SecureQRLoginSystem')

else: # Linux and other Unix-like

appdata\_path = os.path.join(os.path.expanduser('~'), '.config', 'SecureQRLoginSystem')

os.makedirs(appdata\_path, exist\_ok=True)

db\_path = os.path.join(appdata\_path, 'secure\_qr\_login.db')

return db\_path

def get\_qr\_codes\_path():

"""Get the appropriate path for storing QR code images"""

if getattr(sys, 'frozen', False):

base\_path = sys.\_MEIPASS

else:

base\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))

# Use writable path for QR codes

if sys.platform == "win32":

appdata\_path = os.path.join(os.getenv('APPDATA'), 'SecureQRLoginSystem', 'qr\_codes')

elif sys.platform == "darwin": # macOS

appdata\_path = os.path.join(os.path.expanduser('~'), 'Library', 'Application Support', 'SecureQRLoginSystem', 'qr\_codes')

else: # Linux and other Unix-like

appdata\_path = os.path.join(os.path.expanduser('~'), '.config', 'SecureQRLoginSystem', 'qr\_codes')

os.makedirs(appdata\_path, exist\_ok=True)

return appdata\_path

class DatabaseManager:

def \_\_init\_\_(self, db\_path=None):

if db\_path is None:

self.db\_path = get\_database\_path()

else:

self.db\_path = db\_path

self.init\_database()

def init\_database(self):

"""Initialize database tables with proper error handling"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

try:

# First, try to get table info to check current schema

cursor.execute("SELECT name FROM sqlite\_master WHERE type='table' AND name='users'")

table\_exists = cursor.fetchone()

if table\_exists:

# Table exists, check schema

cursor.execute("PRAGMA table\_info(users)")

columns = [column[1] for column in cursor.fetchall()]

if 'adress' in columns and 'address' not in columns:

# Need to migrate from old schema

print("Migrating database schema from 'adress' to 'address'...")

self.migrate\_database(conn, cursor)

elif 'address' not in columns:

# Table exists but missing address column, add it

print("Adding missing 'address' column...")

cursor.execute("ALTER TABLE users ADD COLUMN address TEXT NOT NULL DEFAULT ''")

# If 'address' already exists, do nothing

else:

# Table doesn't exist, create with correct schema

self.create\_tables(conn, cursor)

except Exception as e:

print(f"Error during database initialization: {e}")

# If anything fails, try to create fresh tables

try:

cursor.execute("DROP TABLE IF EXISTS users")

cursor.execute("DROP TABLE IF EXISTS login\_history")

cursor.execute("DROP TABLE IF EXISTS attendance")

cursor.execute("DROP TABLE IF EXISTS holidays")

cursor.execute("DROP TABLE IF EXISTS events")

self.create\_tables(conn, cursor)

except Exception as e2:

print(f"Failed to create fresh tables: {e2}")

conn.commit()

conn.close()

def create\_tables(self, conn, cursor):

"""Create database tables with correct schema"""

# Users table - corrected schema

cursor.execute('''

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

user\_id TEXT UNIQUE NOT NULL,

name TEXT NOT NULL,

address TEXT NOT NULL,

email TEXT UNIQUE NOT NULL,

phone TEXT UNIQUE NOT NULL,

password\_hash TEXT NOT NULL,

salt TEXT NOT NULL,

qr\_code\_data TEXT NOT NULL,

is\_active BOOLEAN DEFAULT 1,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

''')

# Login history table

cursor.execute('''

CREATE TABLE IF NOT EXISTS login\_history (

id INTEGER PRIMARY KEY AUTOINCREMENT,

user\_id TEXT NOT NULL,

action TEXT NOT NULL,

timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

method TEXT NOT NULL,

FOREIGN KEY (user\_id) REFERENCES users (user\_id)

)

''')

# Attendance table

cursor.execute('''

CREATE TABLE IF NOT EXISTS attendance (

id INTEGER PRIMARY KEY AUTOINCREMENT,

user\_id TEXT NOT NULL,

date DATE NOT NULL,

status TEXT NOT NULL, -- 'present', 'leave', 'holiday', 'absent'

login\_time TIMESTAMP,

logout\_time TIMESTAMP,

hours\_worked DECIMAL(4,2),

notes TEXT,

FOREIGN KEY (user\_id) REFERENCES users (user\_id),

UNIQUE(user\_id, date)

)

''')

# Holidays table

cursor.execute('''

CREATE TABLE IF NOT EXISTS holidays (

id INTEGER PRIMARY KEY AUTOINCREMENT,

date DATE NOT NULL UNIQUE,

description TEXT NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

''')

# Events table

cursor.execute('''

CREATE TABLE IF NOT EXISTS events (

id INTEGER PRIMARY KEY AUTOINCREMENT,

title TEXT NOT NULL,

description TEXT,

event\_date DATE NOT NULL,

event\_type TEXT NOT NULL, -- 'holiday', 'event', 'meeting', 'celebration'

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

''')

# Create indexes

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_user\_id ON users(user\_id)')

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_email ON users(email)')

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_phone ON users(phone)')

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_history\_timestamp ON login\_history(timestamp)')

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_history\_user\_id ON login\_history(user\_id)')

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_attendance\_date ON attendance(date)')

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_attendance\_user\_date ON attendance(user\_id, date)')

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_holidays\_date ON holidays(date)')

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_events\_date ON events(event\_date)')

def migrate\_database(self, conn, cursor):

"""Migrate from old schema to new schema"""

try:

# Create temporary table with new schema

cursor.execute('''

CREATE TABLE users\_new (

id INTEGER PRIMARY KEY AUTOINCREMENT,

user\_id TEXT UNIQUE NOT NULL,

name TEXT NOT NULL,

address TEXT NOT NULL,

email TEXT UNIQUE NOT NULL,

phone TEXT UNIQUE NOT NULL,

password\_hash TEXT NOT NULL,

salt TEXT NOT NULL,

qr\_code\_data TEXT NOT NULL,

is\_active BOOLEAN DEFAULT 1,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

)

''')

# Copy data from old table to new table

cursor.execute('''

INSERT INTO users\_new

(id, user\_id, name, address, email, phone, password\_hash, salt, qr\_code\_data, is\_active, created\_at)

SELECT id, user\_id, name, adress, email, phone, password\_hash, salt, qr\_code\_data, is\_active, created\_at

FROM users

''')

# Drop old table and rename new table

cursor.execute('DROP TABLE users')

cursor.execute('ALTER TABLE users\_new RENAME TO users')

# Recreate indexes

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_user\_id ON users(user\_id)')

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_email ON users(email)')

cursor.execute('CREATE INDEX IF NOT EXISTS idx\_phone ON users(phone)')

print("Database migration completed successfully")

except Exception as e:

print(f"Migration failed: {e}")

# If migration fails, try the simpler approach

try:

cursor.execute("ALTER TABLE users RENAME COLUMN adress TO address")

print("Successfully renamed column using ALTER TABLE")

except Exception as e2:

print(f"Column rename also failed: {e2}")

raise

def add\_user(self, user\_data):

"""Add a new user to the database"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

try:

cursor.execute('''

INSERT INTO users (user\_id, name, address, email, phone, password\_hash, salt, qr\_code\_data, is\_active)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)

''', (

user\_data['user\_id'],

user\_data['name'],

user\_data['address'],

user\_data['email'],

user\_data['phone'],

user\_data['password\_hash'],

user\_data['salt'],

user\_data['qr\_code\_data'],

user\_data.get('is\_active', True)

))

conn.commit()

return True, "User added successfully"

except sqlite3.IntegrityError as e:

return False, f"User with this ID, email or phone already exists: {str(e)}"

except Exception as e:

return False, f"Database error: {str(e)}"

finally:

conn.close()

def update\_user\_password(self, user\_id, new\_password\_hash, new\_salt):

"""Update user password"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

try:

cursor.execute('''

UPDATE users

SET password\_hash = ?, salt = ?

WHERE user\_id = ?

''', (new\_password\_hash, new\_salt, user\_id))

conn.commit()

return cursor.rowcount > 0

except Exception as e:

return False

finally:

conn.close()

def get\_user\_by\_id(self, user\_id):

"""Get user by user ID"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

cursor.execute('SELECT \* FROM users WHERE user\_id = ?', (user\_id,))

user = cursor.fetchone()

conn.close()

if user:

return {

'id': user[0],

'user\_id': user[1],

'name': user[2],

'address': user[3],

'email': user[4],

'phone': user[5],

'password\_hash': user[6],

'salt': user[7],

'qr\_code\_data': user[8],

'is\_active': bool(user[9]),

'created\_at': user[10]

}

return None

def get\_user\_by\_qr\_data(self, qr\_data):

"""Get user by QR code data"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

cursor.execute('SELECT \* FROM users WHERE qr\_code\_data = ?', (qr\_data,))

user = cursor.fetchone()

conn.close()

if user:

return {

'id': user[0],

'user\_id': user[1],

'name': user[2],

'address': user[3],

'email': user[4],

'phone': user[5],

'password\_hash': user[6],

'salt': user[7],

'qr\_code\_data': user[8],

'is\_active': bool(user[9]),

'created\_at': user[10]

}

return None

def get\_all\_users(self):

"""Get all users from database (excluding sensitive data)"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

try:

cursor.execute('''

SELECT user\_id, name, address, email, phone, is\_active, created\_at

FROM users

ORDER BY user\_id

''')

except sqlite3.OperationalError as e:

if "no such column: address" in str(e):

# Fallback for old schema

cursor.execute('''

SELECT user\_id, name, adress, email, phone, is\_active, created\_at

FROM users

ORDER BY user\_id

''')

else:

raise

users = cursor.fetchall()

conn.close()

user\_list = []

for user in users:

user\_list.append({

'user\_id': user[0],

'name': user[1],

'address': user[2],

'email': user[3],

'phone': user[4],

'is\_active': bool(user[5]),

'created\_at': user[6]

})

return user\_list

def user\_exists(self, user\_id, email, phone):

"""Check if user with given ID, email or phone already exists"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

cursor.execute('''

SELECT user\_id FROM users

WHERE user\_id = ? OR email = ? OR phone = ?

''', (user\_id, email, phone))

result = cursor.fetchone()

conn.close()

return result is not None

def get\_last\_user\_id(self, prefix="EMP"):

"""Get the last user ID with the given prefix"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

cursor.execute('''

SELECT user\_id FROM users

WHERE user\_id LIKE ?

ORDER BY user\_id DESC

LIMIT 1

''', (f"{prefix}%",))

result = cursor.fetchone()

conn.close()

return result[0] if result else None

def add\_login\_record(self, user\_id, action, method):

"""Add login/logout record to history"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

cursor.execute('''

INSERT INTO login\_history (user\_id, action, method)

VALUES (?, ?, ?)

''', (user\_id, action, method))

conn.commit()

conn.close()

def get\_login\_history(self, user\_id=None, start\_date=None, end\_date=None):

"""Get login history with optional filters"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

query = '''

SELECT lh.user\_id, u.name, lh.action, lh.timestamp, lh.method

FROM login\_history lh

JOIN users u ON lh.user\_id = u.user\_id

'''

params = []

conditions = []

if user\_id:

conditions.append("lh.user\_id = ?")

params.append(user\_id)

if start\_date:

conditions.append("DATE(lh.timestamp) >= ?")

params.append(start\_date)

if end\_date:

conditions.append("DATE(lh.timestamp) <= ?")

params.append(end\_date)

if conditions:

query += " WHERE " + " AND ".join(conditions)

query += " ORDER BY lh.timestamp DESC"

cursor.execute(query, params)

history = cursor.fetchall()

conn.close()

return history

# Attendance and Leave Management Methods

def mark\_attendance(self, user\_id, date, status, login\_time=None, logout\_time=None, hours\_worked=0, notes=""):

"""Mark attendance for a user on a specific date"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

try:

cursor.execute('''

INSERT OR REPLACE INTO attendance

(user\_id, date, status, login\_time, logout\_time, hours\_worked, notes)

VALUES (?, ?, ?, ?, ?, ?, ?)

''', (user\_id, date, status, login\_time, logout\_time, hours\_worked, notes))

conn.commit()

return True

except Exception as e:

print(f"Error marking attendance: {e}")

return False

finally:

conn.close()

def get\_attendance(self, user\_id=None, start\_date=None, end\_date=None):

"""Get attendance records with optional filters"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

query = '''

SELECT a.user\_id, u.name, a.date, a.status, a.login\_time, a.logout\_time, a.hours\_worked, a.notes

FROM attendance a

JOIN users u ON a.user\_id = u.user\_id

'''

params = []

conditions = []

if user\_id:

conditions.append("a.user\_id = ?")

params.append(user\_id)

if start\_date:

conditions.append("a.date >= ?")

params.append(start\_date)

if end\_date:

conditions.append("a.date <= ?")

params.append(end\_date)

if conditions:

query += " WHERE " + " AND ".join(conditions)

query += " ORDER BY a.date DESC, u.name"

cursor.execute(query, params)

attendance = cursor.fetchall()

conn.close()

return attendance

def get\_attendance\_summary(self, start\_date, end\_date):

"""Get attendance summary for all users in date range"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

query = '''

SELECT

u.user\_id,

u.name,

COUNT(a.date) as total\_days,

SUM(CASE WHEN a.status = 'present' THEN 1 ELSE 0 END) as present\_days,

SUM(CASE WHEN a.status = 'leave' THEN 1 ELSE 0 END) as leave\_days,

SUM(CASE WHEN a.status = 'holiday' THEN 1 ELSE 0 END) as holiday\_days,

SUM(CASE WHEN a.status = 'absent' THEN 1 ELSE 0 END) as absent\_days

FROM users u

LEFT JOIN attendance a ON u.user\_id = a.user\_id AND a.date BETWEEN ? AND ?

WHERE u.is\_active = 1

GROUP BY u.user\_id, u.name

ORDER BY u.name

'''

cursor.execute(query, (start\_date, end\_date))

summary = cursor.fetchall()

conn.close()

return summary

def add\_holiday(self, date, description):

"""Add a holiday"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

try:

cursor.execute('''

INSERT OR REPLACE INTO holidays (date, description)

VALUES (?, ?)

''', (date, description))

conn.commit()

return True

except Exception as e:

print(f"Error adding holiday: {e}")

return False

finally:

conn.close()

def get\_holidays(self, start\_date=None, end\_date=None):

"""Get holidays with optional date range"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

query = "SELECT date, description FROM holidays"

params = []

if start\_date and end\_date:

query += " WHERE date BETWEEN ? AND ?"

params.extend([start\_date, end\_date])

elif start\_date:

query += " WHERE date >= ?"

params.append(start\_date)

elif end\_date:

query += " WHERE date <= ?"

params.append(end\_date)

query += " ORDER BY date"

cursor.execute(query, params)

holidays = cursor.fetchall()

conn.close()

return holidays

def is\_holiday(self, date):

"""Check if a date is a holiday"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

cursor.execute('SELECT date FROM holidays WHERE date = ?', (date,))

result = cursor.fetchone()

conn.close()

return result is not None

def mark\_unmarked\_dates\_as\_leave(self, user\_id, start\_date, end\_date, leave\_type="leave"):

"""Mark unmarked dates in range as leave for a user"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

try:

# Get all dates in range

current\_date = datetime.datetime.strptime(start\_date, '%Y-%m-%d').date()

end\_date\_obj = datetime.datetime.strptime(end\_date, '%Y-%m-%d').date()

while current\_date <= end\_date\_obj:

date\_str = current\_date.strftime('%Y-%m-%d')

# Check if date is already marked

cursor.execute('SELECT id FROM attendance WHERE user\_id = ? AND date = ?', (user\_id, date\_str))

existing = cursor.fetchone()

# Check if date is holiday

is\_holiday = self.is\_holiday(date\_str)

if not existing and not is\_holiday:

# Mark as leave

cursor.execute('''

INSERT INTO attendance (user\_id, date, status, notes)

VALUES (?, ?, ?, ?)

''', (user\_id, date\_str, leave\_type, "Auto-marked as leave"))

current\_date += timedelta(days=1)

conn.commit()

return True

except Exception as e:

print(f"Error marking unmarked dates as leave: {e}")

return False

finally:

conn.close()

# Events Management Methods

def add\_event(self, title, description, event\_date, event\_type):

"""Add an event to the calendar"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

try:

cursor.execute('''

INSERT INTO events (title, description, event\_date, event\_type)

VALUES (?, ?, ?, ?)

''', (title, description, event\_date, event\_type))

conn.commit()

return True

except Exception as e:

print(f"Error adding event: {e}")

return False

finally:

conn.close()

def get\_events(self, start\_date=None, end\_date=None, event\_type=None):

"""Get events with optional filters"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

query = "SELECT id, title, description, event\_date, event\_type FROM events"

params = []

conditions = []

if start\_date and end\_date:

conditions.append("event\_date BETWEEN ? AND ?")

params.extend([start\_date, end\_date])

elif start\_date:

conditions.append("event\_date >= ?")

params.append(start\_date)

elif end\_date:

conditions.append("event\_date <= ?")

params.append(end\_date)

if event\_type:

conditions.append("event\_type = ?")

params.append(event\_type)

if conditions:

query += " WHERE " + " AND ".join(conditions)

query += " ORDER BY event\_date"

cursor.execute(query, params)

events = cursor.fetchall()

conn.close()

return events

def delete\_event(self, event\_id):

"""Delete an event"""

conn = sqlite3.connect(self.db\_path)

cursor = conn.cursor()

try:

cursor.execute('DELETE FROM events WHERE id = ?', (event\_id,))

conn.commit()

return cursor.rowcount > 0

except Exception as e:

print(f"Error deleting event: {e}")

return False

finally:

conn.close()

class PasswordResetDialog(wx.Dialog):

def \_\_init\_\_(self, parent, user\_name, user\_id):

# Pass the parent and give it a descriptive title

super().\_\_init\_\_(parent, title=f"Reset Password for {user\_name}", size=(400, 300))

self.user\_id = user\_id

self.user\_name = user\_name

self.init\_ui()

def init\_ui(self):

vbox = wx.BoxSizer(wx.VERTICAL)

instruction = wx.StaticText(self, label=f"Reset password for {self.user\_name} ({self.user\_id}):")

vbox.Add(instruction, 0, wx.ALL | wx.EXPAND, 10)

vbox.Add(wx.StaticText(self, label="New Password:"), 0, wx.LEFT | wx.RIGHT | wx.TOP, 10)

self.new\_password = wx.TextCtrl(self, style=wx.TE\_PASSWORD, size=(300, -1))

vbox.Add(self.new\_password, 0, wx.LEFT | wx.RIGHT | wx.BOTTOM | wx.EXPAND, 10)

vbox.Add(wx.StaticText(self, label="Confirm Password:"), 0, wx.LEFT | wx.RIGHT | wx.TOP, 10)

self.confirm\_password = wx.TextCtrl(self, style=wx.TE\_PASSWORD, size=(300, -1))

vbox.Add(self.confirm\_password, 0, wx.LEFT | wx.RIGHT | wx.BOTTOM | wx.EXPAND, 10)

btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

ok\_btn = wx.Button(self, label="OK", id=wx.ID\_OK)

cancel\_btn = wx.Button(self, label="Cancel", id=wx.ID\_CANCEL)

ok\_btn.Bind(wx.EVT\_BUTTON, self.on\_ok)

cancel\_btn.Bind(wx.EVT\_BUTTON, self.on\_cancel)

btn\_sizer.Add(ok\_btn, 0, wx.ALL, 5)

btn\_sizer.Add(cancel\_btn, 0, wx.ALL, 5)

vbox.Add(btn\_sizer, 0, wx.CENTER | wx.ALL, 10)

self.SetSizer(vbox)

vbox.Fit(self) # <-- This is critical for sizing the dialog correctly

self.Centre()

def on\_ok(self, event):

new\_pwd = self.new\_password.GetValue()

confirm\_pwd = self.confirm\_password.GetValue()

if not new\_pwd:

wx.MessageBox("Password cannot be empty", "Error", wx.OK | wx.ICON\_ERROR)

return

if new\_pwd != confirm\_pwd:

wx.MessageBox("Passwords do not match", "Error", wx.OK | wx.ICON\_ERROR)

return

self.EndModal(wx.ID\_OK)

def on\_cancel(self, event):

self.EndModal(wx.ID\_CANCEL)

def get\_password(self):

return self.new\_password.GetValue()

class SecurityManager:

@staticmethod

def hash\_password(password, salt=None):

"""Hash password with salt"""

if salt is None:

salt = secrets.token\_hex(16)

password\_hash = hashlib.pbkdf2\_hmac(

'sha256',

password.encode('utf-8'),

salt.encode('utf-8'),

100000 # Number of iterations

).hex()

return password\_hash, salt

@staticmethod

def verify\_password(password, password\_hash, salt):

"""Verify password against stored hash"""

new\_hash, \_ = SecurityManager.hash\_password(password, salt)

return new\_hash == password\_hash

@staticmethod

def validate\_phone(phone):

"""Validate phone number - exactly 10 digits"""

return re.match(r'^\d{10}$', phone) is not None

@staticmethod

def validate\_email(email):

"""Validate email format"""

return re.match(r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$', email) is not None

class IDGenerator:

@staticmethod

def generate\_employee\_id(db\_manager, prefix="ALLY"):

"""Generate automatic employee ID"""

last\_id = db\_manager.get\_last\_user\_id(prefix)

if last\_id:

# Extract the numeric part and increment

match = re.match(rf"^{prefix}(\d+)$", last\_id)

if match:

number = int(match.group(1))

new\_number = number + 1

else:

new\_number = 1

else:

new\_number = 1

return f"{prefix}{new\_number:03d}" # 3-digit number with leading zeros

class QRHandler:

@staticmethod

def generate\_qr\_code(user\_id, name, address, email, phone):

"""Generate QR code data and image for a user"""

qr\_data = json.dumps({

"user\_id": user\_id,

"name": name,

"address": address,

"email": email,

"phone": phone,

"timestamp": datetime.datetime.now().isoformat()

})

# Generate QR code image

qr = qrcode.QRCode(

version=1,

error\_correction=qrcode.constants.ERROR\_CORRECT\_L,

box\_size=10,

border=4,

)

qr.add\_data(qr\_data)

qr.make(fit=True)

img = qr.make\_image(fill\_color="black", back\_color="white")

return qr\_data, img

@staticmethod

def scan\_qr\_code(frame):

"""Scan QR code from camera frame using OpenCV's QRCodeDetector"""

try:

qr\_decoder = cv2.QRCodeDetector()

data, bbox, \_ = qr\_decoder.detectAndDecode(frame)

if data and len(data) > 0:

try:

qr\_data = json.loads(data)

return qr\_data

except json.JSONDecodeError:

return data

return None

except Exception as e:

print(f"QR scanning error: {e}")

return None

def get\_calendar\_data(self, year, month):

"""Get all holidays and events for a specific month"""

# Calculate start and end dates for the month

start\_date = f"{year}-{month:02d}-01"

last\_day = calendar.monthrange(year, month)[1]

end\_date = f"{year}-{month:02d}-{last\_day:02d}"

holidays = self.get\_holidays(start\_date, end\_date)

events = self.get\_events(start\_date, end\_date)

return {

'holidays': holidays,

'events': events

}

class PasswordResetDialog(wx.Dialog):

def \_\_init\_\_(self, parent, user\_name, user\_id):

super().\_\_init\_\_(parent, title=f"Reset Password for {user\_name}", size=(400, 300))

self.user\_id = user\_id

self.user\_name = user\_name

self.init\_ui()

def init\_ui(self):

vbox = wx.BoxSizer(wx.VERTICAL)

instruction = wx.StaticText(self, label=f"Reset password for {self.user\_name} ({self.user\_id}):")

vbox.Add(instruction, 0, wx.ALL | wx.EXPAND, 10)

vbox.Add(wx.StaticText(self, label="New Password:"), 0, wx.LEFT | wx.RIGHT | wx.TOP, 10)

self.new\_password = wx.TextCtrl(self, style=wx.TE\_PASSWORD, size=(300, -1))

vbox.Add(self.new\_password, 0, wx.LEFT | wx.RIGHT | wx.BOTTOM | wx.EXPAND, 10)

vbox.Add(wx.StaticText(self, label="Confirm Password:"), 0, wx.LEFT | wx.RIGHT | wx.TOP, 10)

self.confirm\_password = wx.TextCtrl(self, style=wx.TE\_PASSWORD, size=(300, -1))

vbox.Add(self.confirm\_password, 0, wx.LEFT | wx.RIGHT | wx.BOTTOM | wx.EXPAND, 10)

btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

ok\_btn = wx.Button(self, label="OK", id=wx.ID\_OK)

cancel\_btn = wx.Button(self, label="Cancel", id=wx.ID\_CANCEL)

ok\_btn.Bind(wx.EVT\_BUTTON, self.on\_ok)

cancel\_btn.Bind(wx.EVT\_BUTTON, self.on\_cancel)

btn\_sizer.Add(ok\_btn, 0, wx.ALL, 5)

btn\_sizer.Add(cancel\_btn, 0, wx.ALL, 5)

vbox.Add(btn\_sizer, 0, wx.CENTER | wx.ALL, 10)

self.SetSizer(vbox)

vbox.Fit(self) # <-- Fix for button display/dialog sizing

self.Centre()

def on\_ok(self, event):

new\_pwd = self.new\_password.GetValue()

confirm\_pwd = self.confirm\_password.GetValue()

if not new\_pwd:

wx.MessageBox("Password cannot be empty", "Error", wx.OK | wx.ICON\_ERROR)

return

if new\_pwd != confirm\_pwd:

wx.MessageBox("Passwords do not match", "Error", wx.OK | wx.ICON\_ERROR)

return

self.EndModal(wx.ID\_OK)

def on\_cancel(self, event):

self.EndModal(wx.ID\_CANCEL)

def get\_password(self):

return self.new\_password.GetValue()

class CalendarPanel(wx.Panel):

def \_\_init\_\_(self, parent, db\_manager):

super().\_\_init\_\_(parent)

self.db\_manager = db\_manager

self.current\_date = datetime.datetime.now()

self.selected\_date = None

self.init\_ui()

self.refresh\_calendar()

def init\_ui(self):

vbox = wx.BoxSizer(wx.VERTICAL)

# Title

title = wx.StaticText(self, label="Company Calendar", style=wx.ALIGN\_CENTER)

title\_font = wx.Font(16, wx.FONTFAMILY\_DEFAULT, wx.FONTSTYLE\_NORMAL, wx.FONTWEIGHT\_BOLD)

title.SetFont(title\_font)

vbox.Add(title, 0, wx.ALL | wx.CENTER, 10)

# Navigation controls

nav\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.prev\_month\_btn = wx.Button(self, label="← Previous")

self.month\_label = wx.StaticText(self, label="", style=wx.ALIGN\_CENTER)

self.next\_month\_btn = wx.Button(self, label="Next →")

self.today\_btn = wx.Button(self, label="Today")

self.prev\_month\_btn.Bind(wx.EVT\_BUTTON, self.on\_prev\_month)

self.next\_month\_btn.Bind(wx.EVT\_BUTTON, self.on\_next\_month)

self.today\_btn.Bind(wx.EVT\_BUTTON, self.on\_today)

nav\_sizer.Add(self.prev\_month\_btn, 0, wx.RIGHT, 10)

nav\_sizer.Add(self.month\_label, 1, wx.ALIGN\_CENTER)

nav\_sizer.Add(self.next\_month\_btn, 0, wx.LEFT, 10)

nav\_sizer.Add(self.today\_btn, 0, wx.LEFT, 20)

vbox.Add(nav\_sizer, 0, wx.EXPAND | wx.ALL, 10)

# Calendar grid

self.calendar\_grid = wx.GridSizer(7, 7, 5, 5) # 7x7 for days header + 6 weeks

vbox.Add(self.calendar\_grid, 1, wx.EXPAND | wx.ALL, 10)

# Event management section

event\_box = wx.StaticBox(self, label="Add Event/Holiday")

event\_sizer = wx.StaticBoxSizer(event\_box, wx.VERTICAL)

event\_grid = wx.FlexGridSizer(4, 2, 10, 10)

event\_grid.AddGrowableCol(1, 1)

event\_grid.Add(wx.StaticText(self, label="Date:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.event\_date = wx.TextCtrl(self, size=(120, -1))

event\_grid.Add(self.event\_date, 0, wx.EXPAND)

event\_grid.Add(wx.StaticText(self, label="Title:\*"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.event\_title = wx.TextCtrl(self, size=(200, -1))

event\_grid.Add(self.event\_title, 0, wx.EXPAND)

event\_grid.Add(wx.StaticText(self, label="Type:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.event\_type = wx.ComboBox(self, choices=["Holiday", "Event", "Meeting", "Celebration"], style=wx.CB\_READONLY)

self.event\_type.SetSelection(0)

event\_grid.Add(self.event\_type, 0, wx.EXPAND)

event\_grid.Add(wx.StaticText(self, label="Description:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.event\_desc = wx.TextCtrl(self, style=wx.TE\_MULTILINE, size=(-1, 60))

event\_grid.Add(self.event\_desc, 0, wx.EXPAND)

event\_sizer.Add(event\_grid, 0, wx.EXPAND | wx.ALL, 10)

# Buttons

btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.add\_event\_btn = wx.Button(self, label="Add Event/Holiday", size=(150, 35))

self.delete\_event\_btn = wx.Button(self, label="Delete Selected", size=(120, 35))

self.add\_event\_btn.Bind(wx.EVT\_BUTTON, self.on\_add\_event)

self.delete\_event\_btn.Bind(wx.EVT\_BUTTON, self.on\_delete\_event)

btn\_sizer.Add(self.add\_event\_btn, 0, wx.ALL, 5)

btn\_sizer.Add(self.delete\_event\_btn, 0, wx.ALL, 5)

event\_sizer.Add(btn\_sizer, 0, wx.CENTER)

vbox.Add(event\_sizer, 0, wx.EXPAND | wx.ALL, 10)

# Events list for the month

events\_box = wx.StaticBox(self, label="Events This Month")

events\_sizer = wx.StaticBoxSizer(events\_box, wx.VERTICAL)

self.events\_list = wx.ListCtrl(self, style=wx.LC\_REPORT | wx.BORDER\_SUNKEN)

self.events\_list.InsertColumn(0, "Date", width=80)

self.events\_list.InsertColumn(1, "Type", width=80)

self.events\_list.InsertColumn(2, "Title", width=150)

self.events\_list.InsertColumn(3, "Description", width=200)

events\_sizer.Add(self.events\_list, 1, wx.EXPAND | wx.ALL, 5)

vbox.Add(events\_sizer, 1, wx.EXPAND | wx.ALL, 10)

self.SetSizer(vbox)

def refresh\_calendar(self):

"""Refresh the calendar display"""

# Clear existing calendar

self.calendar\_grid.Clear()

# Update month label

self.month\_label.SetLabel(self.current\_date.strftime("%B %Y"))

# Add day headers

days = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]

for day in days:

header = wx.StaticText(self, label=day, style=wx.ALIGN\_CENTER)

header.SetBackgroundColour(wx.Colour(200, 200, 200))

header.SetMinSize((80, 25))

self.calendar\_grid.Add(header, 0, wx.EXPAND)

# Get first day of month and number of days

year = self.current\_date.year

month = self.current\_date.month

first\_day = datetime.date(year, month, 1)

last\_day = datetime.date(year, month, calendar.monthrange(year, month)[1])

# Get calendar data

calendar\_data = self.db\_manager.get\_calendar\_data(year, month)

holidays = {holiday[0]: holiday[1] for holiday in calendar\_data['holidays']}

events = {}

for event in calendar\_data['events']:

events[event[3]] = event # event[3] is event\_date

# Fill in blank days before first day of month

start\_weekday = first\_day.weekday() # Monday=0, Sunday=6

# Convert to Sunday=0, Monday=1, ..., Saturday=6

start\_weekday = (start\_weekday + 1) % 7

for \_ in range(start\_weekday):

self.calendar\_grid.Add(wx.StaticText(self, label=""), 0, wx.EXPAND)

# Add days of the month

current\_day = first\_day

while current\_day <= last\_day:

day\_str = str(current\_day.day)

date\_str = current\_day.strftime("%Y-%m-%d")

# Create day button

day\_btn = wx.Button(self, label=day\_str, size=(80, 60))

day\_btn.date = date\_str

day\_btn.Bind(wx.EVT\_BUTTON, self.on\_day\_click)

# Color coding

if current\_day == datetime.date.today():

day\_btn.SetBackgroundColour(wx.Colour(173, 216, 230)) # Light blue for today

# Check if it's a holiday

if date\_str in holidays:

day\_btn.SetBackgroundColour(wx.Colour(255, 200, 200)) # Light red for holidays

day\_btn.SetToolTip(f"Holiday: {holidays[date\_str]}")

# Check if there are events

if date\_str in events:

event = events[date\_str]

day\_btn.SetBackgroundColour(wx.Colour(200, 230, 200)) # Light green for events

day\_btn.SetToolTip(f"{event[4]}: {event[1]}")

self.calendar\_grid.Add(day\_btn, 0, wx.EXPAND)

current\_day += timedelta(days=1)

# Fill remaining empty cells

while self.calendar\_grid.GetItemCount() < 49: # 7x7 grid

self.calendar\_grid.Add(wx.StaticText(self, label=""), 0, wx.EXPAND)

self.Layout()

# Refresh events list

self.refresh\_events\_list()

def refresh\_events\_list(self):

"""Refresh the events list for the current month"""

self.events\_list.DeleteAllItems()

year = self.current\_date.year

month = self.current\_date.month

start\_date = f"{year}-{month:02d}-01"

last\_day = calendar.monthrange(year, month)[1]

end\_date = f"{year}-{month:02d}-{last\_day:02d}"

events = self.db\_manager.get\_events(start\_date, end\_date)

for event in events:

index = self.events\_list.InsertItem(self.events\_list.GetItemCount(), event[3]) # date

self.events\_list.SetItem(index, 1, event[4]) # type

self.events\_list.SetItem(index, 2, event[1]) # title

self.events\_list.SetItem(index, 3, event[2] or "") # description

self.events\_list.SetItemData(index, event[0]) # store event ID

# Color code based on event type

if event[4] == "Holiday":

self.events\_list.SetItemTextColour(index, wx.Colour(255, 0, 0))

elif event[4] == "Event":

self.events\_list.SetItemTextColour(index, wx.Colour(0, 0, 255))

elif event[4] == "Meeting":

self.events\_list.SetItemTextColour(index, wx.Colour(128, 0, 128))

def on\_prev\_month(self, event):

"""Go to previous month"""

self.current\_date = self.current\_date.replace(day=1)

self.current\_date -= timedelta(days=1)

self.current\_date = self.current\_date.replace(day=1)

self.refresh\_calendar()

def on\_next\_month(self, event):

"""Go to next month"""

self.current\_date = self.current\_date.replace(day=28) + timedelta(days=4)

self.current\_date = self.current\_date.replace(day=1)

self.refresh\_calendar()

def on\_today(self, event):

"""Go to current month"""

self.current\_date = datetime.datetime.now()

self.refresh\_calendar()

def on\_day\_click(self, event):

"""Handle day button click"""

btn = event.GetEventObject()

self.selected\_date = btn.date

self.event\_date.SetValue(self.selected\_date)

# Highlight selected day

for i in range(self.calendar\_grid.GetItemCount()):

item = self.calendar\_grid.GetItem(i)

if item.GetWindow() and hasattr(item.GetWindow(), 'date'):

if item.GetWindow().date == self.selected\_date:

item.GetWindow().SetBackgroundColour(wx.Colour(255, 255, 0)) # Yellow for selected

else:

# Reset to original colors

if item.GetWindow().date == datetime.date.today().strftime("%Y-%m-%d"):

item.GetWindow().SetBackgroundColour(wx.Colour(173, 216, 230))

elif self.db\_manager.is\_holiday(item.GetWindow().date):

item.GetWindow().SetBackgroundColour(wx.Colour(255, 200, 200))

else:

item.GetWindow().SetBackgroundColour(wx.NullColour)

def on\_add\_event(self, event):

"""Add a new event or holiday"""

date = self.event\_date.GetValue().strip()

title = self.event\_title.GetValue().strip()

event\_type = self.event\_type.GetStringSelection()

description = self.event\_desc.GetValue().strip()

if not date or not title:

wx.MessageBox("Please enter both date and title", "Error", wx.OK | wx.ICON\_ERROR)

return

if not self.validate\_date\_format(date):

wx.MessageBox("Please enter a valid date in YYYY-MM-DD format", "Error", wx.OK | wx.ICON\_ERROR)

return

# If it's a holiday, add to holidays table as well

if event\_type == "Holiday":

success = self.db\_manager.add\_holiday(date, title)

if not success:

wx.MessageBox("Error adding holiday", "Error", wx.OK | wx.ICON\_ERROR)

return

# Add to events table

success = self.db\_manager.add\_event(title, description, date, event\_type)

if success:

wx.MessageBox(f"{event\_type} added successfully", "Success", wx.OK | wx.ICON\_INFORMATION)

self.event\_title.Clear()

self.event\_desc.Clear()

self.refresh\_calendar()

else:

wx.MessageBox(f"Error adding {event\_type.lower()}", "Error", wx.OK | wx.ICON\_ERROR)

def on\_delete\_event(self, event):

"""Delete selected event"""

selected\_index = self.events\_list.GetFirstSelected()

if selected\_index == -1:

wx.MessageBox("Please select an event to delete", "Error", wx.OK | wx.ICON\_ERROR)

return

event\_id = self.events\_list.GetItemData(selected\_index)

event\_date = self.events\_list.GetItemText(selected\_index)

event\_type = self.events\_list.GetItemText(selected\_index, 1)

event\_title = self.events\_list.GetItemText(selected\_index, 2)

confirm = wx.MessageBox(

f"Are you sure you want to delete the {event\_type.lower()} '{event\_title}' on {event\_date}?",

"Confirm Delete",

wx.YES\_NO | wx.ICON\_QUESTION

)

if confirm == wx.YES:

# If it's a holiday, remove from holidays table as well

if event\_type == "Holiday":

conn = sqlite3.connect(self.db\_manager.db\_path)

cursor = conn.cursor()

try:

cursor.execute('DELETE FROM holidays WHERE date = ? AND description = ?', (event\_date, event\_title))

conn.commit()

except Exception as e:

print(f"Error deleting holiday: {e}")

finally:

conn.close()

# Delete from events table

success = self.db\_manager.delete\_event(event\_id)

if success:

wx.MessageBox("Event deleted successfully", "Success", wx.OK | wx.ICON\_INFORMATION)

self.refresh\_calendar()

else:

wx.MessageBox("Error deleting event", "Error", wx.OK | wx.ICON\_ERROR)

def validate\_date\_format(self, date\_str):

"""Validate YYYY-MM-DD date format"""

try:

datetime.datetime.strptime(date\_str, '%Y-%m-%d')

return True

except ValueError:

return False

class UserManager:

def \_\_init\_\_(self, db\_manager):

self.db\_manager = db\_manager

self.logged\_in\_users = set()

def add\_user(self, name, address, email, phone, password, prefix="ALLY"):

"""Add a new user with validation"""

# Validate inputs

if not all([name, address, email, phone, password]):

return False, None, "All fields are required"

if not SecurityManager.validate\_email(email):

return False, None, "Invalid email format"

if not SecurityManager.validate\_phone(phone):

return False, None, "Phone number must be exactly 10 digits"

# Generate user ID

user\_id = IDGenerator.generate\_employee\_id(self.db\_manager, prefix)

# Check for existing user

if self.db\_manager.user\_exists(user\_id, email, phone):

return False, None, "User with this ID, email or phone already exists"

# Hash password

password\_hash, salt = SecurityManager.hash\_password(password)

# Generate QR code

qr\_data, qr\_image = QRHandler.generate\_qr\_code(user\_id, name, address, email, phone)

# Prepare user data

user\_data = {

"user\_id": user\_id,

"name": name,

"address": address,

"email": email,

"phone": phone,

"password\_hash": password\_hash,

"salt": salt,

"qr\_code\_data": qr\_data,

"is\_active": True

}

# Save to database

success, message = self.db\_manager.add\_user(user\_data)

if success:

# Save QR code image to appropriate location

qr\_dir = Path(get\_qr\_codes\_path())

qr\_image.save(qr\_dir / f"{user\_id}.png")

return True, user\_id, f"User {name} added successfully with ID: {user\_id}"

else:

return False, None, f"Failed to add user: {message}"

def reset\_user\_password(self, user\_id, new\_password):

"""Reset user password (admin function)"""

user = self.db\_manager.get\_user\_by\_id(user\_id)

if not user:

return False, "User not found"

# Hash new password

password\_hash, salt = SecurityManager.hash\_password(new\_password)

# Update in database

success = self.db\_manager.update\_user\_password(user\_id, password\_hash, salt)

if success:

return True, f"Password reset successfully for {user['name']} ({user\_id})"

else:

return False, "Failed to reset password"

def verify\_manual\_login(self, user\_id, password):

"""Verify manual login credentials"""

user = self.db\_manager.get\_user\_by\_id(user\_id)

if not user:

return False, "User not found"

if not user.get('is\_active', True):

return False, "User account is disabled"

if SecurityManager.verify\_password(password, user['password\_hash'], user['salt']):

return True, "Login successful"

else:

return False, "Invalid password"

def get\_all\_users(self):

"""Get all users with login status"""

try:

users = self.db\_manager.get\_all\_users()

user\_list = []

for user in users:

status = "Logged In" if user['user\_id'] in self.logged\_in\_users else "Logged Out"

user\_list.append({

'user\_id': user['user\_id'],

'name': user['name'],

'address': user['address'],

'email': user['email'],

'phone': user['phone'],

'status': status,

'created\_at': user['created\_at']

})

return user\_list

except Exception as e:

print(f"Error getting users: {e}")

return []

def is\_user\_logged\_in(self, user\_id):

"""Check if user is currently logged in"""

return user\_id in self.logged\_in\_users

def get\_user\_by\_id(self, user\_id):

"""Get user details by user ID"""

return self.db\_manager.get\_user\_by\_id(user\_id)

def logout\_user(self, user\_id):

"""Logout user without password verification"""

if user\_id in self.logged\_in\_users:

self.logged\_in\_users.remove(user\_id)

self.db\_manager.add\_login\_record(user\_id, "logout", "manual\_admin")

return True, f"User {user\_id} logged out successfully"

else:

return False, f"User {user\_id} is not logged in"

class LoginManager:

def \_\_init\_\_(self, db\_manager, user\_manager):

self.db\_manager = db\_manager

self.user\_manager = user\_manager

def handle\_qr\_login(self, qr\_data):

"""Handle login/logout via QR code scanning"""

if isinstance(qr\_data, str):

try:

qr\_data = json.loads(qr\_data)

except:

return False, "Invalid QR code format"

user\_id = qr\_data.get('user\_id')

user = self.db\_manager.get\_user\_by\_qr\_data(json.dumps(qr\_data))

if not user:

return False, "Unknown QR code"

if not user.get('is\_active', True):

return False, "User account is disabled"

current\_time = datetime.datetime.now()

today = current\_time.strftime('%Y-%m-%d')

if self.user\_manager.is\_user\_logged\_in(user\_id):

# Logout user

logout\_time = current\_time.strftime('%Y-%m-%d %H:%M:%S')

# Calculate hours worked

login\_record = self.get\_last\_login\_record(user\_id)

if login\_record:

login\_time = datetime.datetime.strptime(login\_record[3], '%Y-%m-%d %H:%M:%S')

hours\_worked = round((current\_time - login\_time).total\_seconds() / 3600, 2)

else:

hours\_worked = 0

# Update attendance record

self.db\_manager.mark\_attendance(

user\_id, today, "present",

login\_time=login\_record[3] if login\_record else None,

logout\_time=logout\_time,

hours\_worked=hours\_worked

)

self.user\_manager.logged\_in\_users.remove(user\_id)

self.db\_manager.add\_login\_record(user\_id, "logout", "qr\_code")

return True, f"{user['name']} ({user\_id}) logged out successfully"

else:

# Login user

login\_time = current\_time.strftime('%Y-%m-%d %H:%M:%S')

# Check if today is holiday

if self.db\_manager.is\_holiday(today):

status = "holiday"

else:

status = "present"

# Create attendance record

self.db\_manager.mark\_attendance(

user\_id, today, status,

login\_time=login\_time

)

self.user\_manager.logged\_in\_users.add(user\_id)

self.db\_manager.add\_login\_record(user\_id, "login", "qr\_code")

return True, f"{user['name']} ({user\_id}) logged in successfully"

def get\_last\_login\_record(self, user\_id):

"""Get the last login record for a user"""

conn = sqlite3.connect(self.db\_manager.db\_path)

cursor = conn.cursor()

cursor.execute('''

SELECT user\_id, action, timestamp

FROM login\_history

WHERE user\_id = ? AND action = 'login'

ORDER BY timestamp DESC

LIMIT 1

''', (user\_id,))

record = cursor.fetchone()

conn.close()

return record

def handle\_manual\_login(self, user\_id, password):

"""Handle manual login"""

success, message = self.user\_manager.verify\_manual\_login(user\_id, password)

if success:

if not self.user\_manager.is\_user\_logged\_in(user\_id):

self.user\_manager.logged\_in\_users.add(user\_id)

self.db\_manager.add\_login\_record(user\_id, "login", "manual")

# Mark attendance

today = datetime.datetime.now().strftime('%Y-%m-%d')

login\_time = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

if self.db\_manager.is\_holiday(today):

status = "holiday"

else:

status = "present"

self.db\_manager.mark\_attendance(user\_id, today, status, login\_time=login\_time)

return True, message

else:

return False, "User is already logged in"

else:

return False, message

def handle\_manual\_logout(self, user\_id, password):

"""Handle manual logout"""

success, message = self.user\_manager.verify\_manual\_login(user\_id, password)

if success:

if self.user\_manager.is\_user\_logged\_in(user\_id):

current\_time = datetime.datetime.now()

today = current\_time.strftime('%Y-%m-%d')

logout\_time = current\_time.strftime('%Y-%m-%d %H:%M:%S')

# Calculate hours worked

login\_record = self.get\_last\_login\_record(user\_id)

if login\_record:

login\_time = datetime.datetime.strptime(login\_record[3], '%Y-%m-%d %H:%M:%S')

hours\_worked = round((current\_time - login\_time).total\_seconds() / 3600, 2)

else:

hours\_worked = 0

# Update attendance record

self.db\_manager.mark\_attendance(

user\_id, today, "present",

login\_time=login\_record[3] if login\_record else None,

logout\_time=logout\_time,

hours\_worked=hours\_worked

)

self.user\_manager.logged\_in\_users.remove(user\_id)

self.db\_manager.add\_login\_record(user\_id, "logout", "manual")

return True, message.replace("Login", "Logout")

else:

return False, "User is already logged out"

else:

return False, message

def admin\_logout\_user(self, user\_id):

"""Admin can logout any user without password"""

return self.user\_manager.logout\_user(user\_id)

class PasswordResetDialog(wx.Dialog):

def \_\_init\_\_(self, parent, user\_id):

# 1. FIX: Initialize the wx.Dialog class

super().\_\_init\_\_(parent, title="Reset Password", size=(400, 300))

self.user\_id = user\_id

self.init\_ui()

def init\_ui(self):

vbox = wx.BoxSizer(wx.VERTICAL)

# Instructions

# 2. FIX: Use self.user\_id, which is passed in \_\_init\_\_

instruction = wx.StaticText(self, label=f"Reset password for {self.user\_id}:")

vbox.Add(instruction, 0, wx.ALL | wx.EXPAND, 10)

# New Password

vbox.Add(wx.StaticText(self, label="New Password:"), 0, wx.LEFT | wx.RIGHT | wx.TOP, 10)

self.new\_password = wx.TextCtrl(self, style=wx.TE\_PASSWORD, size=(300, -1))

vbox.Add(self.new\_password, 0, wx.LEFT | wx.RIGHT | wx.BOTTOM | wx.EXPAND, 10)

# Confirm Password

vbox.Add(wx.StaticText(self, label="Confirm Password:"), 0, wx.LEFT | wx.RIGHT | wx.TOP, 10)

self.confirm\_password = wx.TextCtrl(self, style=wx.TE\_PASSWORD, size=(300, -1))

vbox.Add(self.confirm\_password, 0, wx.LEFT | wx.RIGHT | wx.BOTTOM | wx.EXPAND, 10)

# Buttons

btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

# Use a standard ID for the OK button

ok\_btn = wx.Button(self, label="OK", id=wx.ID\_OK)

cancel\_btn = wx.Button(self, label="Cancel", id=wx.ID\_CANCEL)

ok\_btn.Bind(wx.EVT\_BUTTON, self.on\_ok)

cancel\_btn.Bind(wx.EVT\_BUTTON, self.on\_cancel)

btn\_sizer.Add(ok\_btn, 0, wx.ALL, 5)

btn\_sizer.Add(cancel\_btn, 0, wx.ALL, 5)

vbox.Add(btn\_sizer, 0, wx.CENTER | wx.ALL, 10)

# 3. FIX: Use self.SetSizer(vbox) AND self.Fit() to make the layout manager size the dialog correctly.

self.SetSizer(vbox)

vbox.Fit(self) # Ensure the dialog box fits the content

self.Centre()

def on\_ok(self, event):

new\_pwd = self.new\_password.GetValue()

confirm\_pwd = self.confirm\_password.GetValue()

if not new\_pwd:

wx.MessageBox("Password cannot be empty", "Error", wx.OK | wx.ICON\_ERROR)

return

if new\_pwd != confirm\_pwd:

wx.MessageBox("Passwords do not match", "Error", wx.OK | wx.ICON\_ERROR)

return

self.EndModal(wx.ID\_OK)

def on\_cancel(self, event):

self.EndModal(wx.ID\_CANCEL)

def get\_password(self):

return self.new\_password.GetValue()

class UserPanel(wx.Panel):

def \_\_init\_\_(self, parent, login\_manager):

super().\_\_init\_\_(parent)

self.login\_manager = login\_manager

self.init\_ui()

def init\_ui(self):

vbox = wx.BoxSizer(wx.VERTICAL)

# Title

title = wx.StaticText(self, label="User Login/Logout Panel", style=wx.ALIGN\_CENTER)

title\_font = wx.Font(16, wx.FONTFAMILY\_DEFAULT, wx.FONTSTYLE\_NORMAL, wx.FONTWEIGHT\_BOLD)

title.SetFont(title\_font)

vbox.Add(title, 0, wx.ALL | wx.CENTER, 10)

# Manual login section

manual\_box = wx.StaticBox(self, label="Manual Login/Logout")

manual\_sizer = wx.StaticBoxSizer(manual\_box, wx.VERTICAL)

grid = wx.FlexGridSizer(3, 2, 10, 10)

grid.AddGrowableCol(1, 1)

grid.Add(wx.StaticText(self, label="Employee ID:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.manual\_id\_input = wx.TextCtrl(self, size=(200, -1))

grid.Add(self.manual\_id\_input, 0, wx.EXPAND)

grid.Add(wx.StaticText(self, label="Password:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.manual\_password\_input = wx.TextCtrl(self, style=wx.TE\_PASSWORD, size=(200, -1))

grid.Add(self.manual\_password\_input, 0, wx.EXPAND)

manual\_sizer.Add(grid, 0, wx.EXPAND | wx.ALL, 15)

# Login/Logout buttons

btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.login\_btn = wx.Button(self, label="Login", size=(100, 35))

self.logout\_btn = wx.Button(self, label="Logout", size=(100, 35))

self.clear\_btn = wx.Button(self, label="Clear", size=(80, 35))

self.login\_btn.Bind(wx.EVT\_BUTTON, self.on\_manual\_login)

self.logout\_btn.Bind(wx.EVT\_BUTTON, self.on\_manual\_logout)

self.clear\_btn.Bind(wx.EVT\_BUTTON, self.on\_clear\_manual)

btn\_sizer.Add(self.login\_btn, 0, wx.ALL, 5)

btn\_sizer.Add(self.logout\_btn, 0, wx.ALL, 5)

btn\_sizer.Add(self.clear\_btn, 0, wx.ALL, 5)

manual\_sizer.Add(btn\_sizer, 0, wx.CENTER)

vbox.Add(manual\_sizer, 0, wx.EXPAND | wx.ALL, 5)

# QR Code section

qr\_box = wx.StaticBox(self, label="QR Code Login/Logout")

qr\_sizer = wx.StaticBoxSizer(qr\_box, wx.VERTICAL)

self.camera\_display = wx.StaticBitmap(self, size=(400, 300))

qr\_sizer.Add(self.camera\_display, 0, wx.ALL | wx.CENTER, 10)

# Camera controls

cam\_btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.start\_cam\_btn = wx.Button(self, label="Start Camera", size=(120, 35))

self.stop\_cam\_btn = wx.Button(self, label="Stop Camera", size=(120, 35))

self.stop\_cam\_btn.Disable()

self.start\_cam\_btn.Bind(wx.EVT\_BUTTON, self.on\_start\_camera)

self.stop\_cam\_btn.Bind(wx.EVT\_BUTTON, self.on\_stop\_camera)

cam\_btn\_sizer.Add(self.start\_cam\_btn, 0, wx.ALL, 5)

cam\_btn\_sizer.Add(self.stop\_cam\_btn, 0, wx.ALL, 5)

qr\_sizer.Add(cam\_btn\_sizer, 0, wx.CENTER)

vbox.Add(qr\_sizer, 0, wx.EXPAND | wx.ALL, 5)

self.SetSizer(vbox)

# Camera variables

self.camera = None

self.timer = wx.Timer(self)

self.is\_scanning = False

self.last\_scanned = None

self.Bind(wx.EVT\_TIMER, self.on\_update\_camera, self.timer)

def on\_manual\_login(self, event):

user\_id = self.manual\_id\_input.GetValue().strip()

password = self.manual\_password\_input.GetValue()

if not user\_id or not password:

wx.MessageBox("Please enter both Employee ID and Password", "Error", wx.OK | wx.ICON\_ERROR)

return

success, message = self.login\_manager.handle\_manual\_login(user\_id, password)

if success:

wx.MessageBox(message, "Success", wx.OK | wx.ICON\_INFORMATION)

self.on\_clear\_manual()

else:

wx.MessageBox(message, "Login Failed", wx.OK | wx.ICON\_ERROR)

def on\_manual\_logout(self, event):

user\_id = self.manual\_id\_input.GetValue().strip()

password = self.manual\_password\_input.GetValue()

if not user\_id or not password:

wx.MessageBox("Please enter both Employee ID and Password", "Error", wx.OK | wx.ICON\_ERROR)

return

success, message = self.login\_manager.handle\_manual\_logout(user\_id, password)

if success:

wx.MessageBox(message, "Success", wx.OK | wx.ICON\_INFORMATION)

self.on\_clear\_manual()

else:

wx.MessageBox(message, "Logout Failed", wx.OK | wx.ICON\_ERROR)

def on\_clear\_manual(self, event=None):

self.manual\_id\_input.Clear()

self.manual\_password\_input.Clear()

def on\_start\_camera(self, event):

self.camera = cv2.VideoCapture(0)

if not self.camera.isOpened():

self.camera = cv2.VideoCapture(1)

if not self.camera.isOpened():

wx.MessageBox("Cannot open camera. Please check camera connection.", "Error", wx.OK | wx.ICON\_ERROR)

return

self.is\_scanning = True

self.start\_cam\_btn.Disable()

self.stop\_cam\_btn.Enable()

self.timer.Start(100)

def on\_stop\_camera(self, event):

self.is\_scanning = False

self.timer.Stop()

if self.camera:

self.camera.release()

self.start\_cam\_btn.Enable()

self.stop\_cam\_btn.Disable()

# Display blank image

blank\_bitmap = wx.Bitmap(400, 300)

self.camera\_display.SetBitmap(blank\_bitmap)

def on\_update\_camera(self, event):

if self.camera and self.is\_scanning:

ret, frame = self.camera.read()

if ret:

# Convert frame for display

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

frame\_resized = cv2.resize(frame\_rgb, (400, 300))

height, width = frame\_resized.shape[:2]

image = wx.Bitmap.FromBuffer(width, height, frame\_resized)

self.camera\_display.SetBitmap(image)

# Scan for QR codes

qr\_data = QRHandler.scan\_qr\_code(frame)

if qr\_data and qr\_data != self.last\_scanned:

self.last\_scanned = qr\_data

success, message = self.login\_manager.handle\_qr\_login(qr\_data)

if success:

wx.MessageBox(message, "QR Scan Result", wx.OK | wx.ICON\_INFORMATION)

else:

wx.MessageBox(message, "QR Scan Error", wx.OK | wx.ICON\_WARNING)

wx.CallLater(2000, self.reset\_scan)

def reset\_scan(self):

self.last\_scanned = None

class HistoryPanel(wx.Panel):

def \_\_init\_\_(self, parent, db\_manager, user\_manager):

super().\_\_init\_\_(parent)

self.db\_manager = db\_manager

self.user\_manager = user\_manager

self.init\_ui()

wx.CallAfter(self.refresh\_history)

def init\_ui(self):

vbox = wx.BoxSizer(wx.VERTICAL)

# Title

title = wx.StaticText(self, label="Login/Logout History", style=wx.ALIGN\_CENTER)

title\_font = wx.Font(16, wx.FONTFAMILY\_DEFAULT, wx.FONTSTYLE\_NORMAL, wx.FONTWEIGHT\_BOLD)

title.SetFont(title\_font)

vbox.Add(title, 0, wx.ALL | wx.CENTER, 10)

# Filters

filter\_box = wx.StaticBox(self, label="Filter History")

filter\_sizer = wx.StaticBoxSizer(filter\_box, wx.VERTICAL)

filter\_grid = wx.FlexGridSizer(2, 4, 10, 10)

filter\_grid.Add(wx.StaticText(self, label="User:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.user\_filter = wx.ComboBox(self, style=wx.CB\_READONLY, size=(200, -1))

self.user\_filter.Append("All Users", None)

# Load users asynchronously to avoid database issues

wx.CallAfter(self.load\_users)

self.user\_filter.SetSelection(0)

self.user\_filter.Bind(wx.EVT\_COMBOBOX, self.on\_filter\_change)

filter\_grid.Add(self.user\_filter, 0, wx.EXPAND)

# Date filters using text inputs instead of DatePickerCtrl

filter\_grid.Add(wx.StaticText(self, label="Start Date:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

date\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.start\_date\_input = wx.TextCtrl(self, size=(100, -1))

self.start\_date\_input.SetToolTip("YYYY-MM-DD format")

date\_sizer.Add(self.start\_date\_input, 0, wx.RIGHT, 5)

date\_sizer.Add(wx.StaticText(self, label="(YYYY-MM-DD)"), 0, wx.ALIGN\_CENTER\_VERTICAL)

filter\_grid.Add(date\_sizer, 0, wx.EXPAND)

filter\_grid.Add(wx.StaticText(self, label="End Date:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

date\_sizer2 = wx.BoxSizer(wx.HORIZONTAL)

self.end\_date\_input = wx.TextCtrl(self, size=(100, -1))

self.end\_date\_input.SetToolTip("YYYY-MM-DD format")

date\_sizer2.Add(self.end\_date\_input, 0, wx.RIGHT, 5)

date\_sizer2.Add(wx.StaticText(self, label="(YYYY-MM-DD)"), 0, wx.ALIGN\_CENTER\_VERTICAL)

filter\_grid.Add(date\_sizer2, 0, wx.EXPAND)

filter\_sizer.Add(filter\_grid, 0, wx.EXPAND | wx.ALL, 10)

# Filter buttons

filter\_btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.apply\_filter\_btn = wx.Button(self, label=" Refresh ", size=(120, 30))

self.clear\_filter\_btn = wx.Button(self, label="Clear Filters", size=(120, 30))

self.export\_btn = wx.Button(self, label="Export to CSV", size=(120, 30))

self.apply\_filter\_btn.Bind(wx.EVT\_BUTTON, self.on\_apply\_filters)

self.clear\_filter\_btn.Bind(wx.EVT\_BUTTON, self.on\_clear\_filters)

self.export\_btn.Bind(wx.EVT\_BUTTON, self.on\_export\_csv)

filter\_btn\_sizer.Add(self.apply\_filter\_btn, 0, wx.ALL, 5)

filter\_btn\_sizer.Add(self.clear\_filter\_btn, 0, wx.ALL, 5)

filter\_btn\_sizer.Add(self.export\_btn, 0, wx.ALL, 5)

filter\_sizer.Add(filter\_btn\_sizer, 0, wx.CENTER)

vbox.Add(filter\_sizer, 0, wx.EXPAND | wx.ALL, 5)

# History list

history\_box = wx.StaticBox(self, label="Login/Logout Records")

history\_sizer = wx.StaticBoxSizer(history\_box, wx.VERTICAL)

self.history\_list = wx.ListCtrl(self, style=wx.LC\_REPORT | wx.BORDER\_SUNKEN)

self.history\_list.InsertColumn(0, "Employee ID", width=100)

self.history\_list.InsertColumn(1, "Name", width=150)

self.history\_list.InsertColumn(2, "Action", width=80)

self.history\_list.InsertColumn(3, "Timestamp", width=150)

self.history\_list.InsertColumn(4, "Method", width=100)

history\_sizer.Add(self.history\_list, 1, wx.EXPAND | wx.ALL, 5)

vbox.Add(history\_sizer, 1, wx.EXPAND | wx.ALL, 5)

self.SetSizer(vbox)

def load\_users(self):

"""Load users for the filter dropdown"""

try:

users = self.user\_manager.get\_all\_users()

self.user\_filter.Clear()

self.user\_filter.Append("All Users", None)

for user in users:

self.user\_filter.Append(f"{user['name']} ({user['user\_id']})", user['user\_id'])

self.user\_filter.SetSelection(0)

except Exception as e:

print(f"Error loading users: {e}")

def on\_apply\_filters(self, event):

self.refresh\_history()

def on\_clear\_filters(self, event):

self.user\_filter.SetSelection(0)

self.start\_date\_input.Clear()

self.end\_date\_input.Clear()

self.refresh\_history()

def refresh\_history(self):

self.history\_list.DeleteAllItems()

# Get filter values

selected\_user = self.user\_filter.GetClientData(self.user\_filter.GetSelection())

start\_date = self.start\_date\_input.GetValue().strip()

end\_date = self.end\_date\_input.GetValue().strip()

# Validate date formats

if start\_date and not self.validate\_date\_format(start\_date):

wx.MessageBox("Start date must be in YYYY-MM-DD format", "Invalid Date", wx.OK | wx.ICON\_WARNING)

return

if end\_date and not self.validate\_date\_format(end\_date):

wx.MessageBox("End date must be in YYYY-MM-DD format", "Invalid Date", wx.OK | wx.ICON\_WARNING)

return

# Get history

try:

history = self.db\_manager.get\_login\_history(selected\_user, start\_date, end\_date)

for record in history:

index = self.history\_list.InsertItem(self.history\_list.GetItemCount(), record[0]) # user\_id

self.history\_list.SetItem(index, 1, record[1]) # name

self.history\_list.SetItem(index, 2, record[2].upper()) # action

self.history\_list.SetItem(index, 3, record[3]) # timestamp

self.history\_list.SetItem(index, 4, record[4].title()) # method

# Color code based on action

if record[2].lower() == "login":

self.history\_list.SetItemTextColour(index, wx.Colour(0, 128, 0)) # Green

else:

self.history\_list.SetItemTextColour(index, wx.Colour(128, 0, 0)) # Red

except Exception as e:

wx.MessageBox(f"Error loading history: {str(e)}", "Error", wx.OK | wx.ICON\_ERROR)

def validate\_date\_format(self, date\_str):

"""Validate YYYY-MM-DD date format"""

try:

datetime.datetime.strptime(date\_str, '%Y-%m-%d')

return True

except ValueError:

return False

def on\_filter\_change(self, event):

self.refresh\_history()

def on\_export\_csv(self, event):

with wx.FileDialog(self, "Save CSV file", wildcard="CSV files (\*.csv)|\*.csv",

style=wx.FD\_SAVE | wx.FD\_OVERWRITE\_PROMPT) as dialog:

if dialog.ShowModal() == wx.ID\_CANCEL:

return

filename = dialog.GetPath()

try:

# Get current filter values

selected\_user = self.user\_filter.GetClientData(self.user\_filter.GetSelection())

start\_date = self.start\_date\_input.GetValue().strip()

end\_date = self.end\_date\_input.GetValue().strip()

history = self.db\_manager.get\_login\_history(selected\_user, start\_date, end\_date)

with open(filename, 'w', newline='', encoding='utf-8') as csvfile:

writer = csv.writer(csvfile)

writer.writerow(['Employee ID', 'Name', 'Action', 'Timestamp', 'Method'])

for record in history:

writer.writerow([

record[0], # user\_id

record[1], # name

record[2].upper(), # action

record[3], # timestamp

record[4].title() # method

])

wx.MessageBox(f"History exported to {filename}", "Success", wx.OK | wx.ICON\_INFORMATION)

except Exception as e:

wx.MessageBox(f"Error exporting CSV: {str(e)}", "Error", wx.OK | wx.ICON\_ERROR)

class LeaveManagementPanel(wx.Panel):

def \_\_init\_\_(self, parent, db\_manager, user\_manager):

super().\_\_init\_\_(parent)

self.db\_manager = db\_manager

self.user\_manager = user\_manager

self.init\_ui()

wx.CallAfter(self.refresh\_data)

def init\_ui(self):

vbox = wx.BoxSizer(wx.VERTICAL)

# Title

title = wx.StaticText(self, label="Leave and Attendance Management", style=wx.ALIGN\_CENTER)

title\_font = wx.Font(16, wx.FONTFAMILY\_DEFAULT, wx.FONTSTYLE\_NORMAL, wx.FONTWEIGHT\_BOLD)

title.SetFont(title\_font)

vbox.Add(title, 0, wx.ALL | wx.CENTER, 10)

# Create notebook for tabs

notebook = wx.Notebook(self)

# Mark Leave tab

self.mark\_leave\_tab = wx.Panel(notebook)

self.init\_mark\_leave\_tab()

notebook.AddPage(self.mark\_leave\_tab, "Mark Leave")

# View Attendance tab

self.view\_attendance\_tab = wx.Panel(notebook)

self.init\_view\_attendance\_tab()

notebook.AddPage(self.view\_attendance\_tab, "View Attendance")

# Calendar tab

self.calendar\_tab = CalendarPanel(notebook, self.db\_manager)

notebook.AddPage(self.calendar\_tab, "Company Calendar")

# Attendance Summary tab

self.summary\_tab = wx.Panel(notebook)

self.init\_summary\_tab()

notebook.AddPage(self.summary\_tab, "Attendance Summary")

vbox.Add(notebook, 1, wx.EXPAND | wx.ALL, 5)

self.SetSizer(vbox)

def init\_mark\_leave\_tab(self):

vbox = wx.BoxSizer(wx.VERTICAL)

# User selection

user\_sizer = wx.BoxSizer(wx.HORIZONTAL)

user\_sizer.Add(wx.StaticText(self.mark\_leave\_tab, label="Select User:"), 0, wx.ALIGN\_CENTER\_VERTICAL | wx.RIGHT, 5)

self.user\_combo = wx.ComboBox(self.mark\_leave\_tab, style=wx.CB\_READONLY, size=(200, -1))

user\_sizer.Add(self.user\_combo, 0, wx.RIGHT, 10)

vbox.Add(user\_sizer, 0, wx.ALL, 10)

# Date range

date\_box = wx.StaticBox(self.mark\_leave\_tab, label="Date Range")

date\_sizer = wx.StaticBoxSizer(date\_box, wx.VERTICAL)

date\_grid = wx.FlexGridSizer(2, 2, 10, 10)

date\_grid.AddGrowableCol(1, 1)

date\_grid.Add(wx.StaticText(self.mark\_leave\_tab, label="Start Date:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

start\_date\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.start\_date = wx.TextCtrl(self.mark\_leave\_tab, size=(120, -1))

self.start\_date.SetValue(datetime.datetime.now().strftime('%Y-%m-%d'))

start\_date\_sizer.Add(self.start\_date, 0, wx.RIGHT, 5)

start\_date\_sizer.Add(wx.StaticText(self.mark\_leave\_tab, label="(YYYY-MM-DD)"), 0, wx.ALIGN\_CENTER\_VERTICAL)

date\_grid.Add(start\_date\_sizer, 0, wx.EXPAND)

date\_grid.Add(wx.StaticText(self.mark\_leave\_tab, label="End Date:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

end\_date\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.end\_date = wx.TextCtrl(self.mark\_leave\_tab, size=(120, -1))

self.end\_date.SetValue(datetime.datetime.now().strftime('%Y-%m-%d'))

end\_date\_sizer.Add(self.end\_date, 0, wx.RIGHT, 5)

end\_date\_sizer.Add(wx.StaticText(self.mark\_leave\_tab, label="(YYYY-MM-DD)"), 0, wx.ALIGN\_CENTER\_VERTICAL)

date\_grid.Add(end\_date\_sizer, 0, wx.EXPAND)

date\_sizer.Add(date\_grid, 0, wx.EXPAND | wx.ALL, 10)

vbox.Add(date\_sizer, 0, wx.EXPAND | wx.ALL, 5)

# Leave type

type\_box = wx.StaticBox(self.mark\_leave\_tab, label="Leave Type")

type\_sizer = wx.StaticBoxSizer(type\_box, wx.VERTICAL)

self.leave\_type = wx.RadioBox(self.mark\_leave\_tab, choices=["Leave", "Sick Leave", "Personal Leave", "Absent"], majorDimension=2, style=wx.RA\_SPECIFY\_ROWS)

type\_sizer.Add(self.leave\_type, 0, wx.EXPAND | wx.ALL, 10)

vbox.Add(type\_sizer, 0, wx.EXPAND | wx.ALL, 5)

# Notes

notes\_box = wx.StaticBox(self.mark\_leave\_tab, label="Notes (Optional)")

notes\_sizer = wx.StaticBoxSizer(notes\_box, wx.VERTICAL)

self.notes\_input = wx.TextCtrl(self.mark\_leave\_tab, style=wx.TE\_MULTILINE, size=(-1, 60))

notes\_sizer.Add(self.notes\_input, 1, wx.EXPAND | wx.ALL, 10)

vbox.Add(notes\_sizer, 1, wx.EXPAND | wx.ALL, 5)

# Buttons

btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.mark\_leave\_btn = wx.Button(self.mark\_leave\_tab, label="Mark Leave", size=(120, 35))

self.mark\_unmarked\_btn = wx.Button(self.mark\_leave\_tab, label="Mark Unmarked as Leave", size=(180, 35))

self.mark\_leave\_btn.Bind(wx.EVT\_BUTTON, self.on\_mark\_leave)

self.mark\_unmarked\_btn.Bind(wx.EVT\_BUTTON, self.on\_mark\_unmarked\_as\_leave)

btn\_sizer.Add(self.mark\_leave\_btn, 0, wx.ALL, 5)

btn\_sizer.Add(self.mark\_unmarked\_btn, 0, wx.ALL, 5)

vbox.Add(btn\_sizer, 0, wx.CENTER | wx.ALL, 10)

self.mark\_leave\_tab.SetSizer(vbox)

def init\_view\_attendance\_tab(self):

vbox = wx.BoxSizer(wx.VERTICAL)

# Filters

filter\_box = wx.StaticBox(self.view\_attendance\_tab, label="Filters")

filter\_sizer = wx.StaticBoxSizer(filter\_box, wx.VERTICAL)

filter\_grid = wx.FlexGridSizer(2, 4, 10, 10)

filter\_grid.AddGrowableCol(1, 1)

filter\_grid.Add(wx.StaticText(self.view\_attendance\_tab, label="User:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.attendance\_user\_filter = wx.ComboBox(self.view\_attendance\_tab, style=wx.CB\_READONLY, size=(200, -1))

self.attendance\_user\_filter.Append("All Users", None)

filter\_grid.Add(self.attendance\_user\_filter, 0, wx.EXPAND)

filter\_grid.Add(wx.StaticText(self.view\_attendance\_tab, label="Start Date:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

start\_date\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.attendance\_start\_date = wx.TextCtrl(self.view\_attendance\_tab, size=(120, -1))

# Default to first day of current month

first\_day = datetime.datetime.now().replace(day=1).strftime('%Y-%m-%d')

self.attendance\_start\_date.SetValue(first\_day)

start\_date\_sizer.Add(self.attendance\_start\_date, 0, wx.RIGHT, 5)

start\_date\_sizer.Add(wx.StaticText(self.view\_attendance\_tab, label="(YYYY-MM-DD)"), 0, wx.ALIGN\_CENTER\_VERTICAL)

filter\_grid.Add(start\_date\_sizer, 0, wx.EXPAND)

filter\_grid.Add(wx.StaticText(self.view\_attendance\_tab, label="End Date:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

end\_date\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.attendance\_end\_date = wx.TextCtrl(self.view\_attendance\_tab, size=(120, -1))

self.attendance\_end\_date.SetValue(datetime.datetime.now().strftime('%Y-%m-%d'))

end\_date\_sizer.Add(self.attendance\_end\_date, 0, wx.RIGHT, 5)

end\_date\_sizer.Add(wx.StaticText(self.view\_attendance\_tab, label="(YYYY-MM-DD)"), 0, wx.ALIGN\_CENTER\_VERTICAL)

filter\_grid.Add(end\_date\_sizer, 0, wx.EXPAND)

filter\_sizer.Add(filter\_grid, 0, wx.EXPAND | wx.ALL, 10)

# Filter buttons

filter\_btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.refresh\_attendance\_btn = wx.Button(self.view\_attendance\_tab, label="Refresh", size=(100, 30))

self.export\_attendance\_btn = wx.Button(self.view\_attendance\_tab, label="Export to CSV", size=(120, 30))

self.refresh\_attendance\_btn.Bind(wx.EVT\_BUTTON, self.on\_refresh\_attendance)

self.export\_attendance\_btn.Bind(wx.EVT\_BUTTON, self.on\_export\_attendance)

filter\_btn\_sizer.Add(self.refresh\_attendance\_btn, 0, wx.ALL, 5)

filter\_btn\_sizer.Add(self.export\_attendance\_btn, 0, wx.ALL, 5)

filter\_sizer.Add(filter\_btn\_sizer, 0, wx.CENTER)

vbox.Add(filter\_sizer, 0, wx.EXPAND | wx.ALL, 5)

# Attendance list

attendance\_box = wx.StaticBox(self.view\_attendance\_tab, label="Attendance Records")

attendance\_sizer = wx.StaticBoxSizer(attendance\_box, wx.VERTICAL)

self.attendance\_list = wx.ListCtrl(self.view\_attendance\_tab, style=wx.LC\_REPORT | wx.BORDER\_SUNKEN)

self.attendance\_list.InsertColumn(0, "Employee ID", width=100)

self.attendance\_list.InsertColumn(1, "Name", width=150)

self.attendance\_list.InsertColumn(2, "Date", width=100)

self.attendance\_list.InsertColumn(3, "Status", width=100)

self.attendance\_list.InsertColumn(4, "Login Time", width=120)

self.attendance\_list.InsertColumn(5, "Logout Time", width=120)

self.attendance\_list.InsertColumn(6, "Hours Worked", width=100)

self.attendance\_list.InsertColumn(7, "Notes", width=200)

attendance\_sizer.Add(self.attendance\_list, 1, wx.EXPAND | wx.ALL, 5)

vbox.Add(attendance\_sizer, 1, wx.EXPAND | wx.ALL, 5)

self.view\_attendance\_tab.SetSizer(vbox)

def init\_summary\_tab(self):

vbox = wx.BoxSizer(wx.VERTICAL)

# Date range for summary

summary\_filter\_box = wx.StaticBox(self.summary\_tab, label="Summary Period")

summary\_filter\_sizer = wx.StaticBoxSizer(summary\_filter\_box, wx.VERTICAL)

summary\_grid = wx.FlexGridSizer(2, 2, 10, 10)

summary\_grid.AddGrowableCol(1, 1)

summary\_grid.Add(wx.StaticText(self.summary\_tab, label="Start Date:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

start\_date\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.summary\_start\_date = wx.TextCtrl(self.summary\_tab, size=(120, -1))

# Default to first day of current month

first\_day = datetime.datetime.now().replace(day=1).strftime('%Y-%m-%d')

self.summary\_start\_date.SetValue(first\_day)

start\_date\_sizer.Add(self.summary\_start\_date, 0, wx.RIGHT, 5)

start\_date\_sizer.Add(wx.StaticText(self.summary\_tab, label="(YYYY-MM-DD)"), 0, wx.ALIGN\_CENTER\_VERTICAL)

summary\_grid.Add(start\_date\_sizer, 0, wx.EXPAND)

summary\_grid.Add(wx.StaticText(self.summary\_tab, label="End Date:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

end\_date\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.summary\_end\_date = wx.TextCtrl(self.summary\_tab, size=(120, -1))

self.summary\_end\_date.SetValue(datetime.datetime.now().strftime('%Y-%m-%d'))

end\_date\_sizer.Add(self.summary\_end\_date, 0, wx.RIGHT, 5)

end\_date\_sizer.Add(wx.StaticText(self.summary\_tab, label="(YYYY-MM-DD)"), 0, wx.ALIGN\_CENTER\_VERTICAL)

summary\_grid.Add(end\_date\_sizer, 0, wx.EXPAND)

summary\_filter\_sizer.Add(summary\_grid, 0, wx.EXPAND | wx.ALL, 10)

# Generate button

self.generate\_summary\_btn = wx.Button(self.summary\_tab, label="Generate Summary", size=(140, 35))

self.generate\_summary\_btn.Bind(wx.EVT\_BUTTON, self.on\_generate\_summary)

summary\_filter\_sizer.Add(self.generate\_summary\_btn, 0, wx.ALL | wx.CENTER, 10)

vbox.Add(summary\_filter\_sizer, 0, wx.EXPAND | wx.ALL, 5)

# Summary list

summary\_box = wx.StaticBox(self.summary\_tab, label="Attendance Summary")

summary\_sizer = wx.StaticBoxSizer(summary\_box, wx.VERTICAL)

self.summary\_list = wx.ListCtrl(self.summary\_tab, style=wx.LC\_REPORT | wx.BORDER\_SUNKEN)

self.summary\_list.InsertColumn(0, "Employee ID", width=100)

self.summary\_list.InsertColumn(1, "Name", width=150)

self.summary\_list.InsertColumn(2, "Total Days", width=80)

self.summary\_list.InsertColumn(3, "Present", width=80)

self.summary\_list.InsertColumn(4, "Leave", width=80)

self.summary\_list.InsertColumn(5, "Holiday", width=80)

self.summary\_list.InsertColumn(6, "Absent", width=80)

self.summary\_list.InsertColumn(7, "Attendance %", width=100)

summary\_sizer.Add(self.summary\_list, 1, wx.EXPAND | wx.ALL, 5)

vbox.Add(summary\_sizer, 1, wx.EXPAND | wx.ALL, 5)

self.summary\_tab.SetSizer(vbox)

def refresh\_data(self):

"""Refresh all data in the panel"""

self.load\_users()

self.refresh\_attendance()

def load\_users(self):

"""Load users for dropdowns"""

try:

users = self.user\_manager.get\_all\_users()

# Clear and populate user combo in mark leave tab

self.user\_combo.Clear()

for user in users:

self.user\_combo.Append(f"{user['name']} ({user['user\_id']})", user['user\_id'])

# Clear and populate user filter in view attendance tab

self.attendance\_user\_filter.Clear()

self.attendance\_user\_filter.Append("All Users", None)

for user in users:

self.attendance\_user\_filter.Append(f"{user['name']} ({user['user\_id']})", user['user\_id'])

self.attendance\_user\_filter.SetSelection(0)

except Exception as e:

print(f"Error loading users: {e}")

def on\_mark\_leave(self, event):

user\_id = self.user\_combo.GetClientData(self.user\_combo.GetSelection())

start\_date = self.start\_date.GetValue().strip()

end\_date = self.end\_date.GetValue().strip()

leave\_type = self.leave\_type.GetStringSelection().lower().replace(" ", "\_")

notes = self.notes\_input.GetValue().strip()

if not user\_id:

wx.MessageBox("Please select a user", "Error", wx.OK | wx.ICON\_ERROR)

return

if not self.validate\_date\_format(start\_date) or not self.validate\_date\_format(end\_date):

wx.MessageBox("Please enter valid dates in YYYY-MM-DD format", "Error", wx.OK | wx.ICON\_ERROR)

return

# Mark leave for each day in the range

current\_date = datetime.datetime.strptime(start\_date, '%Y-%m-%d').date()

end\_date\_obj = datetime.datetime.strptime(end\_date, '%Y-%m-%d').date()

days\_marked = 0

while current\_date <= end\_date\_obj:

date\_str = current\_date.strftime('%Y-%m-%d')

# Skip if it's a holiday

if not self.db\_manager.is\_holiday(date\_str):

success = self.db\_manager.mark\_attendance(user\_id, date\_str, leave\_type, notes=notes)

if success:

days\_marked += 1

current\_date += timedelta(days=1)

wx.MessageBox(f"Leave marked successfully for {days\_marked} days", "Success", wx.OK | wx.ICON\_INFORMATION)

self.refresh\_attendance()

def on\_mark\_unmarked\_as\_leave(self, event):

user\_id = self.user\_combo.GetClientData(self.user\_combo.GetSelection())

start\_date = self.start\_date.GetValue().strip()

end\_date = self.end\_date.GetValue().strip()

leave\_type = self.leave\_type.GetStringSelection().lower().replace(" ", "\_")

if not user\_id:

wx.MessageBox("Please select a user", "Error", wx.OK | wx.ICON\_ERROR)

return

if not self.validate\_date\_format(start\_date) or not self.validate\_date\_format(end\_date):

wx.MessageBox("Please enter valid dates in YYYY-MM-DD format", "Error", wx.OK | wx.ICON\_ERROR)

return

success = self.db\_manager.mark\_unmarked\_dates\_as\_leave(user\_id, start\_date, end\_date, leave\_type)

if success:

wx.MessageBox("Unmarked dates successfully marked as leave", "Success", wx.OK | wx.ICON\_INFORMATION)

self.refresh\_attendance()

else:

wx.MessageBox("Error marking unmarked dates as leave", "Error", wx.OK | wx.ICON\_ERROR)

def on\_refresh\_attendance(self, event=None):

self.refresh\_attendance()

def refresh\_attendance(self):

"""Refresh attendance list"""

self.attendance\_list.DeleteAllItems()

selected\_user = self.attendance\_user\_filter.GetClientData(self.attendance\_user\_filter.GetSelection())

start\_date = self.attendance\_start\_date.GetValue().strip()

end\_date = self.attendance\_end\_date.GetValue().strip()

# Validate dates

if start\_date and not self.validate\_date\_format(start\_date):

wx.MessageBox("Start date must be in YYYY-MM-DD format", "Invalid Date", wx.OK | wx.ICON\_WARNING)

return

if end\_date and not self.validate\_date\_format(end\_date):

wx.MessageBox("End date must be in YYYY-MM-DD format", "Invalid Date", wx.OK | wx.ICON\_WARNING)

return

try:

attendance = self.db\_manager.get\_attendance(selected\_user, start\_date, end\_date)

for record in attendance:

index = self.attendance\_list.InsertItem(self.attendance\_list.GetItemCount(), record[0]) # user\_id

self.attendance\_list.SetItem(index, 1, record[1]) # name

self.attendance\_list.SetItem(index, 2, record[2]) # date

self.attendance\_list.SetItem(index, 3, record[3].title()) # status

self.attendance\_list.SetItem(index, 4, record[4] or "") # login\_time

self.attendance\_list.SetItem(index, 5, record[5] or "") # logout\_time

self.attendance\_list.SetItem(index, 6, str(record[6]) if record[6] else "") # hours\_worked

self.attendance\_list.SetItem(index, 7, record[7] or "") # notes

# Color code based on status

if record[3] == "present":

self.attendance\_list.SetItemTextColour(index, wx.Colour(0, 128, 0)) # Green

elif record[3] == "leave":

self.attendance\_list.SetItemTextColour(index, wx.Colour(255, 165, 0)) # Orange

elif record[3] == "holiday":

self.attendance\_list.SetItemTextColour(index, wx.Colour(0, 0, 255)) # Blue

elif record[3] == "absent":

self.attendance\_list.SetItemTextColour(index, wx.Colour(255, 0, 0)) # Red

except Exception as e:

wx.MessageBox(f"Error loading attendance: {str(e)}", "Error", wx.OK | wx.ICON\_ERROR)

def on\_export\_attendance(self, event):

with wx.FileDialog(self, "Save CSV file", wildcard="CSV files (\*.csv)|\*.csv",

style=wx.FD\_SAVE | wx.FD\_OVERWRITE\_PROMPT) as dialog:

if dialog.ShowModal() == wx.ID\_CANCEL:

return

filename = dialog.GetPath()

try:

selected\_user = self.attendance\_user\_filter.GetClientData(self.attendance\_user\_filter.GetSelection())

start\_date = self.attendance\_start\_date.GetValue().strip()

end\_date = self.attendance\_end\_date.GetValue().strip()

attendance = self.db\_manager.get\_attendance(selected\_user, start\_date, end\_date)

with open(filename, 'w', newline='', encoding='utf-8') as csvfile:

writer = csv.writer(csvfile)

writer.writerow(['Employee ID', 'Name', 'Date', 'Status', 'Login Time', 'Logout Time', 'Hours Worked', 'Notes', 'Holidays/Events'])

# Get holidays and events for the date range

holidays = self.db\_manager.get\_holidays(start\_date, end\_date)

events = self.db\_manager.get\_events(start\_date, end\_date)

holiday\_dict = {holiday[0]: holiday[1] for holiday in holidays}

event\_dict = {}

for event in events:

if event[3] not in event\_dict:

event\_dict[event[3]] = []

event\_dict[event[3]].append(f"{event[4]}: {event[1]}")

for record in attendance:

date\_str = record[2]

holiday\_event\_info = ""

if date\_str in holiday\_dict:

holiday\_event\_info = f"Holiday: {holiday\_dict[date\_str]}"

elif date\_str in event\_dict:

holiday\_event\_info = "; ".join(event\_dict[date\_str])

writer.writerow([

record[0], # user\_id

record[1], # name

record[2], # date

record[3].title(), # status

record[4] or "", # login\_time

record[5] or "", # logout\_time

record[6] or "", # hours\_worked

record[7] or "", # notes

holiday\_event\_info # holidays and events

])

wx.MessageBox(f"Attendance exported to {filename}\n\nHolidays and events are included in the export.", "Success", wx.OK | wx.ICON\_INFORMATION)

except Exception as e:

wx.MessageBox(f"Error exporting CSV: {str(e)}", "Error", wx.OK | wx.ICON\_ERROR)

def on\_generate\_summary(self, event):

"""Generate attendance summary"""

self.summary\_list.DeleteAllItems()

start\_date = self.summary\_start\_date.GetValue().strip()

end\_date = self.summary\_end\_date.GetValue().strip()

if not self.validate\_date\_format(start\_date) or not self.validate\_date\_format(end\_date):

wx.MessageBox("Please enter valid dates in YYYY-MM-DD format", "Error", wx.OK | wx.ICON\_ERROR)

return

try:

summary = self.db\_manager.get\_attendance\_summary(start\_date, end\_date)

for record in summary:

user\_id, name, total\_days, present, leave, holiday, absent = record

# Calculate attendance percentage (excluding holidays)

working\_days = total\_days - holiday

if working\_days > 0:

attendance\_pct = (present / working\_days) \* 100

else:

attendance\_pct = 0

index = self.summary\_list.InsertItem(self.summary\_list.GetItemCount(), user\_id)

self.summary\_list.SetItem(index, 1, name)

self.summary\_list.SetItem(index, 2, str(total\_days))

self.summary\_list.SetItem(index, 3, str(present))

self.summary\_list.SetItem(index, 4, str(leave))

self.summary\_list.SetItem(index, 5, str(holiday))

self.summary\_list.SetItem(index, 6, str(absent))

self.summary\_list.SetItem(index, 7, f"{attendance\_pct:.1f}%")

# Color code based on attendance percentage

if attendance\_pct >= 90:

self.summary\_list.SetItemTextColour(index, wx.Colour(0, 128, 0)) # Green

elif attendance\_pct >= 75:

self.summary\_list.SetItemTextColour(index, wx.Colour(255, 165, 0)) # Orange

else:

self.summary\_list.SetItemTextColour(index, wx.Colour(255, 0, 0)) # Red

except Exception as e:

wx.MessageBox(f"Error generating summary: {str(e)}", "Error", wx.OK | wx.ICON\_ERROR)

def validate\_date\_format(self, date\_str):

"""Validate YYYY-MM-DD date format"""

try:

datetime.datetime.strptime(date\_str, '%Y-%m-%d')

return True

except ValueError:

return False

class AdminPanel(wx.Panel):

def \_\_init\_\_(self, parent, user\_manager, login\_manager, db\_manager):

super().\_\_init\_\_(parent)

self.user\_manager = user\_manager

self.login\_manager = login\_manager

self.db\_manager = db\_manager

self.init\_ui()

wx.CallAfter(self.refresh\_user\_list)

def init\_ui(self):

vbox = wx.BoxSizer(wx.VERTICAL)

# Create notebook for tabs

notebook = wx.Notebook(self)

# User Management tab

self.user\_management\_tab = wx.Panel(notebook)

self.init\_user\_management\_tab()

notebook.AddPage(self.user\_management\_tab, "User Management")

# Leave Management tab

self.leave\_management\_tab = LeaveManagementPanel(notebook, self.db\_manager, self.user\_manager)

notebook.AddPage(self.leave\_management\_tab, "Leave Management")

vbox.Add(notebook, 1, wx.EXPAND | wx.ALL, 5)

self.SetSizer(vbox)

def init\_user\_management\_tab(self):

vbox = wx.BoxSizer(wx.VERTICAL)

# Title

title = wx.StaticText(self.user\_management\_tab, label="Admin Panel - User Management", style=wx.ALIGN\_CENTER)

title\_font = wx.Font(16, wx.FONTFAMILY\_DEFAULT, wx.FONTSTYLE\_NORMAL, wx.FONTWEIGHT\_BOLD)

title.SetFont(title\_font)

vbox.Add(title, 0, wx.ALL | wx.CENTER, 10)

# Add user section

add\_user\_box = wx.StaticBox(self.user\_management\_tab, label="Add New User")

add\_user\_sizer = wx.StaticBoxSizer(add\_user\_box, wx.VERTICAL)

grid = wx.FlexGridSizer(6, 2, 10, 10)

grid.AddGrowableCol(1, 1)

# Name field

grid.Add(wx.StaticText(self.user\_management\_tab, label="Full Name:\*"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.name\_input = wx.TextCtrl(self.user\_management\_tab, size=(250, -1))

grid.Add(self.name\_input, 0, wx.EXPAND)

# Address field

grid.Add(wx.StaticText(self.user\_management\_tab, label="Address:\*"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.address\_input = wx.TextCtrl(self.user\_management\_tab, size=(250, -1))

grid.Add(self.address\_input, 0, wx.EXPAND)

# Email field

grid.Add(wx.StaticText(self.user\_management\_tab, label="Email:\*"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.email\_input = wx.TextCtrl(self.user\_management\_tab, size=(250, -1))

grid.Add(self.email\_input, 0, wx.EXPAND)

# Phone field

grid.Add(wx.StaticText(self.user\_management\_tab, label="Phone:\*"), 0, wx.ALIGN\_CENTER\_VERTICAL)

phone\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.phone\_input = wx.TextCtrl(self.user\_management\_tab, size=(150, -1))

phone\_sizer.Add(self.phone\_input, 0, wx.RIGHT, 5)

phone\_sizer.Add(wx.StaticText(self.user\_management\_tab, label="(10 digits)"), 0, wx.ALIGN\_CENTER\_VERTICAL)

grid.Add(phone\_sizer, 0, wx.EXPAND)

# Password field

grid.Add(wx.StaticText(self.user\_management\_tab, label="Password:\*"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.password\_input = wx.TextCtrl(self.user\_management\_tab, style=wx.TE\_PASSWORD, size=(200, -1))

grid.Add(self.password\_input, 0, wx.EXPAND)

# ID Prefix

grid.Add(wx.StaticText(self.user\_management\_tab, label="ID Prefix:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

prefix\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.prefix\_input = wx.TextCtrl(self.user\_management\_tab, value="ALLY", size=(60, -1))

prefix\_sizer.Add(self.prefix\_input, 0, wx.RIGHT, 5)

prefix\_sizer.Add(wx.StaticText(self.user\_management\_tab, label="(e.g.,ALLY, EMP, USR, ADM)"), 0, wx.ALIGN\_CENTER\_VERTICAL)

grid.Add(prefix\_sizer, 0, wx.EXPAND)

add\_user\_sizer.Add(grid, 0, wx.EXPAND | wx.ALL, 15)

# Add user button

self.add\_user\_btn = wx.Button(self.user\_management\_tab, label="Add User & Generate QR Code", size=(220, 40))

self.add\_user\_btn.Bind(wx.EVT\_BUTTON, self.on\_add\_user)

add\_user\_sizer.Add(self.add\_user\_btn, 0, wx.ALL | wx.CENTER, 10)

vbox.Add(add\_user\_sizer, 0, wx.EXPAND | wx.ALL, 5)

# User management section

user\_mgmt\_box = wx.StaticBox(self.user\_management\_tab, label="User Management")

user\_mgmt\_sizer = wx.StaticBoxSizer(user\_mgmt\_box, wx.VERTICAL)

# User list

self.user\_list = wx.ListCtrl(self.user\_management\_tab, style=wx.LC\_REPORT | wx.BORDER\_SUNKEN | wx.LC\_SINGLE\_SEL)

self.user\_list.InsertColumn(0, "Employee ID", width=100)

self.user\_list.InsertColumn(1, "Name", width=150)

self.user\_list.InsertColumn(2, "Address", width=150)

self.user\_list.InsertColumn(3, "Email", width=200)

self.user\_list.InsertColumn(4, "Phone", width=120)

self.user\_list.InsertColumn(5, "Status", width=100)

self.user\_list.InsertColumn(6, "Created", width=120)

user\_mgmt\_sizer.Add(self.user\_list, 1, wx.EXPAND | wx.ALL, 5)

# Management buttons

mgmt\_btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

self.refresh\_btn = wx.Button(self.user\_management\_tab, label="Refresh List", size=(100, 35))

self.reset\_password\_btn = wx.Button(self.user\_management\_tab, label="Reset Password", size=(120, 35))

self.force\_logout\_btn = wx.Button(self.user\_management\_tab, label="Force Logout", size=(120, 35))

self.view\_qr\_btn = wx.Button(self.user\_management\_tab, label="View QR Code", size=(120, 35))

self.refresh\_btn.Bind(wx.EVT\_BUTTON, self.on\_refresh)

self.reset\_password\_btn.Bind(wx.EVT\_BUTTON, self.on\_reset\_password)

self.force\_logout\_btn.Bind(wx.EVT\_BUTTON, self.on\_force\_logout)

self.view\_qr\_btn.Bind(wx.EVT\_BUTTON, self.on\_view\_qr)

mgmt\_btn\_sizer.Add(self.refresh\_btn, 0, wx.ALL, 5)

mgmt\_btn\_sizer.Add(self.reset\_password\_btn, 0, wx.ALL, 5)

mgmt\_btn\_sizer.Add(self.force\_logout\_btn, 0, wx.ALL, 5)

mgmt\_btn\_sizer.Add(self.view\_qr\_btn, 0, wx.ALL, 5)

user\_mgmt\_sizer.Add(mgmt\_btn\_sizer, 0, wx.CENTER)

vbox.Add(user\_mgmt\_sizer, 1, wx.EXPAND | wx.ALL, 5)

self.user\_management\_tab.SetSizer(vbox)

def get\_selected\_user(self):

"""Get selected user ID from list"""

selected\_index = self.user\_list.GetFirstSelected()

if selected\_index == -1:

return None

return self.user\_list.GetItemText(selected\_index)

def on\_add\_user(self, event):

name = self.name\_input.GetValue().strip()

address = self.address\_input.GetValue().strip()

email = self.email\_input.GetValue().strip()

phone = self.phone\_input.GetValue().strip()

password = self.password\_input.GetValue()

prefix = self.prefix\_input.GetValue().strip().upper()

if not prefix:

prefix = "ALLY"

# Validate required fields

if not all([name, address, email, phone, password]):

wx.MessageBox("Please fill in all required fields (\*)", "Error", wx.OK | wx.ICON\_ERROR)

return

# Validate phone number

if not SecurityManager.validate\_phone(phone):

wx.MessageBox("Phone number must be exactly 10 digits", "Validation Error", wx.OK | wx.ICON\_WARNING)

return

# Add user

success, user\_id, message = self.user\_manager.add\_user(name, address, email, phone, password, prefix)

if success:

wx.MessageBox(f"{message}\n\nEmployee ID: {user\_id}", "Success", wx.OK | wx.ICON\_INFORMATION)

self.clear\_form()

self.refresh\_user\_list()

else:

wx.MessageBox(message, "Error", wx.OK | wx.ICON\_ERROR)

def on\_reset\_password(self, event):

user\_id = self.get\_selected\_user()

if not user\_id:

wx.MessageBox("Please select a user from the list", "Error", wx.OK | wx.ICON\_ERROR)

return

user = self.user\_manager.get\_user\_by\_id(user\_id)

if not user:

wx.MessageBox("Selected user not found", "Error", wx.OK | wx.ICON\_ERROR)

return

# Passes both the name (for the dialog title/instruction) and the ID

dialog = PasswordResetDialog(self, user['name'], user\_id)

if dialog.ShowModal() == wx.ID\_OK:

new\_password = dialog.get\_password()

# This relies on self.user\_manager having a method that handles hashing and DB update.

success, message = self.user\_manager.reset\_user\_password(user\_id, new\_password)

if success:

wx.MessageBox(message, "Success", wx.OK | wx.ICON\_INFORMATION)

else:

wx.MessageBox(message, "Error", wx.OK | wx.ICON\_ERROR)

dialog.Destroy() # Crucial: always destroy wx.Dialogs

def on\_force\_logout(self, event):

user\_id = self.get\_selected\_user()

if not user\_id:

wx.MessageBox("Please select a user from the list", "Error", wx.OK | wx.ICON\_ERROR)

return

user = self.user\_manager.get\_user\_by\_id(user\_id)

if not user:

wx.MessageBox("Selected user not found", "Error", wx.OK | wx.ICON\_ERROR)

return

# Confirm force logout

confirm = wx.MessageBox(

f"Are you sure you want to force logout {user['name']} ({user\_id})?",

"Confirm Force Logout",

wx.YES\_NO | wx.ICON\_QUESTION

)

if confirm == wx.YES:

success, message = self.login\_manager.admin\_logout\_user(user\_id)

if success:

wx.MessageBox(message, "Success", wx.OK | wx.ICON\_INFORMATION)

self.refresh\_user\_list()

else:

wx.MessageBox(message, "Info", wx.OK | wx.ICON\_INFORMATION)

def on\_view\_qr(self, event):

user\_id = self.get\_selected\_user()

if not user\_id:

wx.MessageBox("Please select a user from the list", "Error", wx.OK | wx.ICON\_ERROR)

return

user = self.user\_manager.get\_user\_by\_id(user\_id)

if not user:

wx.MessageBox("Selected user not found", "Error", wx.OK | wx.ICON\_ERROR)

return

qr\_path = Path(get\_qr\_codes\_path()) / f"{user\_id}.png"

if not qr\_path.exists():

wx.MessageBox(f"QR code not found for {user\_id}", "Error", wx.OK | wx.ICON\_ERROR)

return

# Display QR code in a dialog

dialog = wx.Dialog(self, title=f"QR Code - {user['name']} ({user\_id})", size=(400, 450))

vbox = wx.BoxSizer(wx.VERTICAL)

# User info

info\_text = wx.StaticText(dialog, label=f"Name: {user['name']}\nID: {user\_id}\nEmail: {user['email']}\nAddress: {user['address']}")

vbox.Add(info\_text, 0, wx.ALL | wx.CENTER, 10)

# QR code image

image = wx.Image(str(qr\_path), wx.BITMAP\_TYPE\_PNG)

image = image.Scale(300, 300, wx.IMAGE\_QUALITY\_HIGH)

bitmap = wx.Bitmap(image)

qr\_display = wx.StaticBitmap(dialog, bitmap=bitmap)

vbox.Add(qr\_display, 0, wx.ALL | wx.CENTER, 10)

# Close button

close\_btn = wx.Button(dialog, label="Close")

close\_btn.Bind(wx.EVT\_BUTTON, lambda e: dialog.EndModal(wx.ID\_OK))

vbox.Add(close\_btn, 0, wx.ALL | wx.CENTER, 10)

dialog.SetSizer(vbox)

dialog.Centre()

dialog.ShowModal()

def clear\_form(self):

"""Clear the add user form"""

self.name\_input.Clear()

self.address\_input.Clear()

self.email\_input.Clear()

self.phone\_input.Clear()

self.password\_input.Clear()

self.prefix\_input.SetValue("ALLY")

def on\_refresh(self, event):

"""Handle refresh button click by refreshing the user list."""

self.refresh\_user\_list()

# Optionally, refresh other AdminPanel tabs if they exist and have a refresh method

if hasattr(self, 'leave\_management\_tab'):

self.leave\_management\_tab.refresh\_data()

wx.MessageBox("User data refreshed.", "Success", wx.OK | wx.ICON\_INFORMATION)

def refresh\_user\_list(self):

"""Refreshes the list of users displayed in the User Management list control."""

try:

# Clear the current list control content

self.user\_list.DeleteAllItems()

# Fetch the latest user data

users = self.user\_manager.get\_all\_users()

# Populate the list control

for i, user in enumerate(users):

index = self.user\_list.InsertItem(i, user['user\_id'])

self.user\_list.SetItem(index, 1, user['name'])

self.user\_list.SetItem(index, 2, user['role'])

self.user\_list.SetItem(index, 3, user['status'])

self.user\_list.SetItem(index, 4, str(user['created\_at']))

except Exception as e:

wx.MessageBox(f"Failed to refresh user list: {str(e)}", "Error", wx.OK | wx.ICON\_ERROR)

class LoginDialog(wx.Dialog):

def \_\_init\_\_(self, parent):

super().\_\_init\_\_(parent, title="Admin Login", size=(300, 200))

self.init\_ui()

def init\_ui(self):

vbox = wx.BoxSizer(wx.VERTICAL)

vbox.Add(wx.StaticText(self, label="Admin Authentication Required"), 0, wx.ALL | wx.CENTER, 10)

grid = wx.FlexGridSizer(2, 2, 10, 10)

grid.AddGrowableCol(1, 1)

grid.Add(wx.StaticText(self, label="Password:"), 0, wx.ALIGN\_CENTER\_VERTICAL)

self.password\_input = wx.TextCtrl(self, style=wx.TE\_PASSWORD)

grid.Add(self.password\_input, 0, wx.EXPAND)

vbox.Add(grid, 0, wx.EXPAND | wx.ALL, 15)

# Buttons

btn\_sizer = wx.BoxSizer(wx.HORIZONTAL)

login\_btn = wx.Button(self, label="Login")

cancel\_btn = wx.Button(self, label="Cancel")

login\_btn.Bind(wx.EVT\_BUTTON, self.on\_login)

cancel\_btn.Bind(wx.EVT\_BUTTON, self.on\_cancel)

btn\_sizer.Add(login\_btn, 0, wx.ALL, 5)

btn\_sizer.Add(cancel\_btn, 0, wx.ALL, 5)

vbox.Add(btn\_sizer, 0, wx.CENTER)

self.SetSizer(vbox)

self.Centre()

def on\_login(self, event):

# For demo purposes, using a simple password check

# In production, this should be properly secured

if self.password\_input.GetValue() == "admin123":

self.EndModal(wx.ID\_OK)

else:

wx.MessageBox("Invalid admin password", "Error", wx.OK | wx.ICON\_ERROR)

def on\_cancel(self, event):

self.EndModal(wx.ID\_CANCEL)

class MainFrame(wx.Frame):

def \_\_init\_\_(self):

super().\_\_init\_\_(None, title="Secure QR Login System", size=(1200, 800))

# Initialize managers

self.db\_manager = DatabaseManager()

self.user\_manager = UserManager(self.db\_manager)

self.login\_manager = LoginManager(self.db\_manager, self.user\_manager)

self.init\_ui()

self.Centre()

def init\_ui(self):

# Create notebook for tabs

notebook = wx.Notebook(self)

# Add tabs

self.user\_tab = UserPanel(notebook, self.login\_manager)

self.history\_tab = HistoryPanel(notebook, self.db\_manager, self.user\_manager)

notebook.AddPage(self.user\_tab, "User Panel")

notebook.AddPage(self.history\_tab, "Login History")

# Admin panel is protected and now includes leave management

self.admin\_tab = AdminPanel(notebook, self.user\_manager, self.login\_manager, self.db\_manager)

notebook.AddPage(self.admin\_tab, "Admin Panel")

# Create sizer

sizer = wx.BoxSizer(wx.VERTICAL)

sizer.Add(notebook, 1, wx.EXPAND)

self.SetSizer(sizer)

# Create status bar

self.CreateStatusBar()

self.update\_status\_bar()

# Bind events

self.Bind(wx.EVT\_CLOSE, self.on\_close)

notebook.Bind(wx.EVT\_NOTEBOOK\_PAGE\_CHANGED, self.on\_tab\_change)

def on\_tab\_change(self, event):

if event.GetSelection() == 2: # Admin panel tab

if not self.authenticate\_admin():

# Switch back to first tab if authentication fails

event.GetEventObject().SetSelection(0)

else:

# Refresh data in admin panel

if hasattr(self.admin\_tab, 'refresh\_user\_list'):

self.admin\_tab.refresh\_user\_list()

if hasattr(self.admin\_tab, 'leave\_management\_tab'):

self.admin\_tab.leave\_management\_tab.refresh\_data()

else:

self.update\_status\_bar()

def authenticate\_admin(self):

"""Authenticate admin access"""

dialog = LoginDialog(self)

result = dialog.ShowModal()

dialog.Destroy()

return result == wx.ID\_OK

def update\_status\_bar(self):

"""Update status bar with current statistics"""

try:

users = self.user\_manager.get\_all\_users()

total\_users = len(users)

logged\_in = sum(1 for user in users if user['status'] == "Logged In")

self.SetStatusText(f"Total Users: {total\_users} | Logged In: {logged\_in} | Logged Out: {total\_users - logged\_in}")

except Exception as e:

self.SetStatusText(f"Error loading user statistics: {str(e)}")

def on\_close(self, event):

# Stop camera if running

if hasattr(self.user\_tab, 'camera') and self.user\_tab.camera:

self.user\_tab.camera.release()

self.Destroy()

def main():

app = wx.App()

frame = MainFrame()

frame.Show()

app.MainLoop()

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

main()