Voy a realizar la Fase 1 te enviare todo lo que tengo:

main.py:

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from config_email import EMAIL_HOST, EMAIL_HOST_PASSWORD, EMAIL_PORT,
EMAIL_HOST_USER
import os
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
from werkzeug.security import generate_password_hash, check_password_hash
from werkzeug.utils import secure_filename
from flask import send_from_directory
from Config.database_connection import create_connection
from Controllers.admin_controller import AdminController
from Controllers.profesor_controller import ProfesorController
from Controllers.estudiante_controller import EstudianteController
from Controllers.horario_controller import HorarioController
from Controllers.indices_controller import IndicesController
from Controllers.evento_controller import EventoController
from Controllers.autenticacion import autenticar_usuario
import sqlite3  # O el conector que uses
import traceback
import io
import csv
import time

app = Flask(__name__, template_folder='Views', static_folder='Static')
app.secret_key = 'clave_secreta_gestion_estudiantil_2023'

# OAuth (Google) - intenta habilitar sólo si `authlib` está instalada y las credenciales están
configuradas.
# OAuth (Google) - intenta habilitar sólo si `authlib` está instalada y las credenciales están
configuradas.
oauth = None
try:
    # Importa authlib sólo si está disponible en el entorno
    from authlib.integrations.flask_client import OAuth  # type: ignore
except ImportError:
    app.logger.info('authlib no está instalada; Google OAuth deshabilitado.')
    oauth = None
else:
    # Comprueba que las credenciales estén en las variables de entorno
    google_client_id = os.environ.get('GOOGLE_CLIENT_ID')
    google_client_secret = os.environ.get('GOOGLE_CLIENT_SECRET')
    if not google_client_id or not google_client_secret:
        app.logger.warning('Google OAuth deshabilitado: faltan
GOOGLE_CLIENT_ID/GOOGLE_CLIENT_SECRET en variables de entorno.')
        oauth = None
```

```
    else:
        oauth = OAuth(app)
        oauth.register(
            name='google',
            client_id=google_client_id,
            client_secret=google_client_secret,
            access_token_url='https://oauth2.googleapis.com/token',
            authorize_url='https://accounts.google.com/o/oauth2/v2/auth',
            api_base_url='https://www.googleapis.com/oauth2/v2/',
            userinfo_endpoint='https://openidconnect.googleapis.com/v1/userinfo',
            client_kwargs={'scope': 'openid email profile'}
        )

        # Configuración de subida de archivos (hojas de vida)
        ALLOWED_EXTENSIONS = {'pdf'}
        UPLOAD_FOLDER = os.path.join(app.static_folder, 'uploads', 'hojas')
        os.makedirs(UPLOAD_FOLDER, exist_ok=True)

        def allowed_file(filename):
            return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def get_db_connection():
    conn = sqlite3.connect('tu_base_de_datos.db')
    conn.row_factory = sqlite3.Row
    return conn

# --------------------------
# Manejadores de errores
# --------------------------
@app.errorhandler(404)
def page_not_found(e):
    return render_template('error/404.html'), 404

@app.errorhandler(500)
def internal_server_error(e):
    return render_template('error/500.html'), 500

# --------------------------
# Rutas de Autenticación
# --------------------------
@app.route('/')
def home():
    if 'usuario' in session:
        return redirect(url_for('dashboard'))
    return render_template('auth/Home.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
```

```
        correo = request.form.get('correo')
        contraseña = request.form.get('contraseña')

        if not correo or not contraseña:
            flash("Por favor, completa todos los campos.", "error")
            return redirect(url_for('login'))

        usuario = autenticar_usuario(correo, contraseña)
        if usuario:
            session['usuario'] = usuario
            flash("Inicio de sesión exitoso.", "success")
            # Redirige según el tipo de usuario
            if usuario['tipo'] == 'admin':
                return redirect(url_for('admin_dashboard'))
            elif usuario['tipo'] == 'profesor':
                return redirect(url_for('profesor_dashboard'))
            elif usuario['tipo'] == 'estudiante':
                return redirect(url_for('estudiante_dashboard'))
            elif usuario['tipo'] == 'padre':
                return redirect(url_for('padre_dashboard'))
            else:
                flash("Rol desconocido.", "error")
                return redirect(url_for('login'))
        else:
            flash("Correo o contraseña incorrectos.", "error")
    return render_template('auth/login.html')

@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        nombre = request.form.get('nombre')
        correo = request.form.get('correo')
        contraseña = request.form.get('contraseña')
        rol = request.form.get('rol')

        if not nombre or not correo or not contraseña or not rol:
            flash("Por favor, completa todos los campos.", "error")
            return redirect(url_for('register'))

        conexion = create_connection()
        if conexion:
            try:
                cursor = conexion.cursor()
                contraseña_cifrada = generate_password_hash(contraseña)
                cursor.execute(
                    "INSERT INTO usuarios (nombre, correo, contraseña, rol) VALUES (%s, %s, %s, %s)",
                    (nombre, correo, contraseña_cifrada, rol)
                )
                tabla = {
```

```
                'admin': 'administradores',
                'profesor': 'profesores',
                'estudiante': 'estudiantes',
                'padre': 'padres'
            }.get(rol)
            if not tabla:
                flash("Rol no válido.", "error")
                return redirect(url_for('register'))
            cursor.execute(
                f"INSERT INTO {tabla} (nombre, correo, contraseña) VALUES (%s, %s, %s)",
                (nombre, correo, contraseña_cifrada)
            )
            conexion.commit()
            flash("Registro exitoso. Ahora puedes iniciar sesión.", "success")
            return redirect(url_for('login'))
        except Exception as e:
            flash(f"Error al registrar usuario: {e}", "error")
        finally:
            cursor.close()
            conexion.close()
    return render_template('auth/register.html')

@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('home'))


@app.route('/login/google')
def login_google():
    if oauth is None:
        flash('Inicio de sesión con Google no está disponible (dependencia faltante).', 'error')
        return redirect(url_for('login'))
    redirect_uri = url_for('authorize_google', _external=True)
    return oauth.google.authorize_redirect(redirect_uri)


@app.route('/authorize/google')
def authorize_google():
    if oauth is None:
        flash('Inicio de sesión con Google no está disponible (dependencia faltante).', 'error')
        return redirect(url_for('login'))
    try:
        token = oauth.google.authorize_access_token()
        # Obtener información del usuario
        resp = oauth.google.get('userinfo')
        user_info = resp.json()
        email = user_info.get('email')
        nombre = user_info.get('name') or user_info.get('given_name') or email
```

```
        if not email:
            flash('No se pudo obtener el correo desde Google.', 'error')
            return redirect(url_for('login'))

        # Buscar usuario en la tabla estudiantes
        conexion = create_connection()
        if conexion:
            try:
                cursor = conexion.cursor(dictionary=True)
                cursor.execute("SELECT id, nombre, correo FROM estudiantes WHERE correo=%s",
(email,))
                usuario = cursor.fetchone()
                if usuario:
                    session['usuario'] = {'id': usuario['id'], 'nombre': usuario['nombre'], 'correo':
usuario['correo'], 'tipo': 'estudiante'}
                    flash('Inicio de sesión con Google exitoso.', 'success')
                    return redirect(url_for('dashboard'))
                else:
                    # Crear estudiante nuevo (sin contraseña) y marcarlo para revisión si es necesario
                    cursor.execute("INSERT INTO estudiantes (nombre, correo) VALUES (%s, %s)",
(nombre, email))
                    conexion.commit()
                    new_id = cursor.lastrowid
                    session['usuario'] = {'id': new_id, 'nombre': nombre, 'correo': email, 'tipo': 'estudiante'}
                    flash('Cuenta creada e iniciada con Google. Si requiere aprobación, el administrador podrá
gestionarla.', 'success')
                    return redirect(url_for('dashboard'))
            except Exception as e:
                app.logger.error(f"Error en login Google: {e}")
                flash('Ocurrió un error al iniciar sesión con Google.', 'error')
            finally:
                try:
                    cursor.close()
                except Exception:
                    pass
                try:
                    conexion.close()
                except Exception:
                    pass
    except Exception as e:
        app.logger.error(f"Error autorizando con Google: {e}")
        flash('Error durante la autorización con Google.', 'error')
    return redirect(url_for('login'))

# -------------------------
# Dashboard y rutas por rol
# -------------------------
@app.route('/dashboard')
def dashboard():
```

```
    if 'usuario' not in session:
        return redirect(url_for('home'))
    if 'tipo' not in session['usuario']:
        flash("Error en la sesión. Por favor, inicia sesión nuevamente.", "error")
        return redirect(url_for('logout'))
    tipo_usuario = session['usuario']['tipo']
    dashboards = {
        'admin': 'admin_dashboard',
        'profesor': 'profesor_dashboard',
        'estudiante': 'estudiante_dashboard',
        'padre': 'padre_dashboard'
    }
    if tipo_usuario not in dashboards:
        flash("Rol desconocido. Contacta al administrador.", "error")
        return redirect(url_for('home'))
    return redirect(url_for(dashboards[tipo_usuario]))

# -------------------------
# Rutas ADMINISTRADOR
# -------------------------
@app.route('/admin/dashboard')
def admin_dashboard():
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('admin_login'))
    conexion = create_connection()
    try:
        with conexion.cursor(dictionary=True) as cursor:
            cursor.execute("SELECT COUNT(*) AS total FROM usuarios")
            total_usuarios = cursor.fetchone()['total']
            cursor.execute("SELECT COUNT(*) AS total FROM materias")
            total_materias = cursor.fetchone()['total']
            cursor.execute("SELECT COUNT(*) AS total FROM horarios")
            total_horarios = cursor.fetchone()['total']
            cursor.execute("SELECT COUNT(*) AS total FROM inscripciones")
            total_inscripciones = cursor.fetchone()['total']
        stats = {
            'total_usuarios': total_usuarios,
            'total_materias': total_materias,
            'total_horarios': total_horarios,
            'total_inscripciones': total_inscripciones
        }
        return render_template('admin/dashboard.html', usuario=session['usuario'], stats=stats)
    except Exception as e:
        app.logger.error(f"Error al cargar el dashboard del administrador: {str(e)}")
        flash("Ocurrió un error al cargar el dashboard.", "error")
        return redirect(url_for('home'))
    finally:
        if conexion:
```

```
        conexion.close()

@app.route('/admin/usuarios', methods=['GET'])
def gestionar_usuarios():
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    usuarios = AdminController.obtener_usuarios()
    return render_template('admin/usuarios.html', usuarios=usuarios)

@app.route('/admin/agregar_usuario', methods=['GET', 'POST'])
def agregar_usuario():
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    if request.method == 'POST':
        nombre = request.form.get('nombre')
        correo = request.form.get('correo')
        contraseña = request.form.get('contraseña')
        rol = request.form.get('rol')
        if nombre and correo and contraseña and rol:
            success = AdminController.agregar_usuario(nombre, correo, contraseña, rol)
            if success:
                flash("Usuario agregado correctamente.", "success")
                return redirect(url_for('gestionar_usuarios'))
            else:
                flash("Error al agregar el usuario. Verifica los datos.", "error")
        else:
            flash("Por favor, completa todos los campos.", "error")
    return render_template('admin/agregarUsuario.html')

@app.route('/admin/eliminar_usuario/<int:id>', methods=['POST'])
def eliminar_usuario(id):
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    exito = AdminController.eliminar_usuario(id)
    if exito:
        flash("Usuario eliminado correctamente.", "success")
    else:
        flash("Error al eliminar el usuario.", "error")
    return redirect(url_for('gestionar_usuarios'))

@app.route('/admin/editar_usuario/<int:id>', methods=['GET', 'POST'])
def editar_usuario(id):
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    usuario = AdminController.obtener_usuario_por_id(id)
```

```
    if not usuario:
        flash("Usuario no encontrado.", "error")
        return redirect(url_for('gestionar_usuarios'))
    if request.method == 'POST':
        nombre = request.form.get('nombre')
        correo = request.form.get('correo')
        rol = request.form.get('rol')
        if nombre and correo and rol:
            exito = AdminController.modificar_usuario(id, nombre, correo, rol)
            if exito:
                flash("Usuario actualizado correctamente.", "success")
                return redirect(url_for('gestionar_usuarios'))
            else:
                flash("Error al actualizar el usuario. Verifica los datos.", "error")
        else:
            flash("Por favor, completa todos los campos.", "error")
    return render_template('admin/editarUsuario.html', usuario=usuario)

# Materias
@app.route('/admin/materias', methods=['GET', 'POST'])
def gestionar_materias():
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    conexion = create_connection()
    materias = []
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            if request.method == 'POST':
                nombre_materia = request.form.get('nombre_materia')
                descripcion = request.form.get('descripcion')
                if nombre_materia and descripcion:
                    cursor.execute("INSERT INTO materias (nombre, descripcion) VALUES (%s, %s)",
(nombre_materia, descripcion))
                    conexion.commit()
                    flash("Materia agregada correctamente.", "success")
                else:
                    flash("Por favor, completa todos los campos.", "error")
            cursor.execute("SELECT * FROM materias")
            materias = cursor.fetchall()
        except Exception as e:
            flash(f"Error al gestionar materias: {e}", "error")
        finally:
            cursor.close()
            conexion.close()
    return render_template('admin/materias.html', materias=materias)

# Ruta de Horarios para el dashboard del administrador
```

```
@app.route('/admin/horarios', methods=['GET', 'POST'])
def gestionar_horarios_admin():
    conexion = create_connection()
    if not conexion:
        return "Error de conexión a la base de datos", 500

    cursor = conexion.cursor(dictionary=True)
    # Obtener datos para los selects
    cursor.execute("SELECT id, nombre FROM estudiantes")
    estudiantes = cursor.fetchall()
    cursor.execute("SELECT id, nombre FROM profesores")
    profesores = cursor.fetchall()
    cursor.execute("SELECT id, nombre FROM materias")
    materias = cursor.fetchall()

    if request.method == 'POST':
        id_estudiante = request.form['id_estudiante']
        id_profesor = request.form['id_profesor']
        id_materia = request.form['id_materia']
        dia_semana = request.form['dia_semana']
        hora_inicio = request.form['hora_inicio']
        hora_fin = request.form['hora_fin']
        try:
            cursor.execute("""
                INSERT INTO horarios (id_estudiante, id_profesor, dia_semana, hora_inicio, hora_fin,
id_materia)
                VALUES (%s, %s, %s, %s, %s, %s)
            """, (id_estudiante, id_profesor, dia_semana, hora_inicio, hora_fin, id_materia))
            conexion.commit()
        except Exception as e:
            conexion.rollback()
            return f"Error al guardar el horario: {e}", 500

    # Obtener todos los horarios para mostrar en la tabla
    cursor.execute("""
        SELECT h.id, h.id_estudiante, m.nombre AS materia, h.dia_semana, h.hora_inicio, h.hora_fin
        FROM horarios h
        JOIN materias m ON h.id_materia = m.id
        ORDER BY h.id DESC
    """)
    horarios = cursor.fetchall()
    cursor.close()
    conexion.close()
    return render_template('admin/horarios.html', estudiantes=estudiantes, profesores=profesores,
materias=materias, horarios=horarios)
#Ruta para editar horarios al dasboard del administrador
@app.route('/admin/horarios/editar/<int:id>', methods=['GET', 'POST'])
def editar_horario(id):
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
```

```
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    conexion = create_connection()
    horario = None
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            if request.method == 'POST':
                id_estudiante = request.form.get('id_estudiante')
                dia_semana = request.form.get('dia_semana')
                hora_inicio = request.form.get('hora_inicio')
                hora_fin = request.form.get('hora_fin')
                id_materia = request.form.get('id_materia')
                print("DEBUG FORM:", id_estudiante, dia_semana, hora_inicio, hora_fin, id_materia)
                if id_estudiante and dia_semana and hora_inicio and hora_fin and id_materia:
                    cursor.execute("""
                        UPDATE horarios
                        SET id_estudiante=%s, dia_semana=%s, hora_inicio=%s, hora_fin=%s, id_materia=%s
                        WHERE id=%s
                    """, (id_estudiante, dia_semana, hora_inicio, hora_fin, id_materia, id))
                    conexion.commit()
                    flash("Horario actualizado correctamente.", "success")
                    return redirect(url_for('gestionar_horarios_admin'))
                else:
                    flash("Por favor, completa todos los campos.", "error")
            else:
                cursor.execute("SELECT * FROM horarios WHERE id=%s", (id,))
                horario = cursor.fetchone()
        except Exception as e:
            flash(f"Error al editar horario: {e}", "error")
        finally:
            cursor.close()
            conexion.close()
    return render_template('admin/editar_horario.html', horario=horario)

@app.route('/admin/horarios/eliminar/<int:id>', methods=['POST'])
def eliminar_horario(id):
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    conexion = create_connection()
    if conexion:
        try:
            cursor = conexion.cursor()
            cursor.execute("DELETE FROM horarios WHERE id=%s", (id,))
            conexion.commit()
            flash("Horario eliminado correctamente.", "success")
        except Exception as e:
            flash(f"Error al eliminar horario: {e}", "error")
```

```
        finally:
            cursor.close()
            conexion.close()
    return redirect(url_for('gestionar_horarios_admin'))

# Inscripciones
@app.route('/admin/inscripciones', methods=['GET', 'POST'])
def gestionar_inscripciones():
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    conexion = create_connection()
    inscripciones = []
    estudiantes = []
    materias = []
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            cursor.execute("SELECT id, nombre, puede_inscribirse FROM estudiantes")
            estudiantes = cursor.fetchall()
            cursor.execute("SELECT id, nombre FROM materias")
            materias = cursor.fetchall()
            cursor.execute("""
                SELECT i.id_estudiante, i.id_materia, e.nombre AS estudiante, m.nombre AS materia
                FROM inscripciones i
                JOIN estudiantes e ON i.id_estudiante = e.id
                JOIN materias m ON i.id_materia = m.id
            """)
            inscripciones = cursor.fetchall()
            if request.method == 'POST':
                id_estudiante = request.form.get('id_estudiante')
                id_materia = request.form.get('id_materia')
                if id_estudiante and id_materia:
                    cursor.execute("INSERT IGNORE INTO inscripciones (id_estudiante, id_materia)
VALUES (%s, %s)", (id_estudiante, id_materia))
                    conexion.commit()
                    flash("Inscripción agregada correctamente.", "success")
                    return redirect(url_for('gestionar_inscripciones'))
                else:
                    flash("Selecciona estudiante y materia.", "error")
        except Exception as e:
            flash(f"Error al gestionar inscripciones: {e}", "error")
        finally:
            cursor.close()
            conexion.close()
    # CREA LOS DICCIONARIOS Y PASALOS AL TEMPLATE
    estudiantes_dict = {e['id']: e for e in estudiantes}
    materias_dict = {m['id']: m for m in materias}
    return render_template(
```

```
        'admin/inscripciones.html',
        inscripciones=inscripciones,
        estudiantes=estudiantes,
        materias=materias,
        estudiantes_dict=estudiantes_dict,
        materias_dict=materias_dict
    )

@app.route('/admin/inscripciones/eliminar', methods=['POST'])
def eliminar_inscripcion():
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    id_estudiante = request.form.get('id_estudiante')
    id_materia = request.form.get('id_materia')
    conexion = create_connection()
    if conexion and id_estudiante and id_materia:
        try:
            cursor = conexion.cursor()
            cursor.execute("DELETE FROM inscripciones WHERE id_estudiante=%s AND
id_materia=%s", (id_estudiante, id_materia))
            conexion.commit()
            flash("Inscripción eliminada correctamente.", "success")
        except Exception as e:
            flash(f"Error al eliminar inscripción: {e}", "error")
        finally:
            cursor.close()
            conexion.close()
    return redirect(url_for('gestionar_inscripciones'))

@app.route('/admin/inscripciones/habilitar_estudiante', methods=['POST'])
def habilitar_estudiante_inscripcion():
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    id_estudiante = request.form.get('id_estudiante')
    puede = request.form.get('puede_inscribirse') == '1'
    conexion = create_connection()
    if conexion and id_estudiante:
        try:
            cursor = conexion.cursor()
            cursor.execute("UPDATE estudiantes SET puede_inscribirse=%s WHERE id=%s", (puede,
id_estudiante))
            conexion.commit()
            flash("Permiso de inscripción actualizado.", "success")
        except Exception as e:
            flash(f"Error al actualizar permiso: {e}", "error")
        finally:
            cursor.close()
```

```
            conexion.close()
    return redirect(url_for('gestionar_inscripciones'))

@app.route('/admin/inscripciones/modificar', methods=['POST'])
def modificar_inscripcion():
    if 'usuario' not in session or session['usuario']['tipo'] != 'admin':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))

    id_estudiante = request.form.get('id_estudiante')
    id_materia = request.form.get('id_materia')
    nuevo_id_estudiante = request.form.get('nuevo_id_estudiante')
    nuevo_id_materia = request.form.get('nuevo_id_materia')

    conexion = create_connection()
    if conexion and id_estudiante and id_materia and nuevo_id_estudiante and nuevo_id_materia:
        try:
            cursor = conexion.cursor()
            # Eliminar la inscripción anterior
            cursor.execute("DELETE FROM inscripciones WHERE id_estudiante=%s AND
id_materia=%s", (id_estudiante, id_materia))
            # Insertar la nueva inscripción
            cursor.execute("INSERT INTO inscripciones (id_estudiante, id_materia) VALUES (%s, %s)",
(nuevo_id_estudiante, nuevo_id_materia))
            conexion.commit()
            flash("Inscripción modificada correctamente.", "success")
        except Exception as e:
            flash(f"Error al modificar inscripción: {e}", "error")
        finally:
            cursor.close()
            conexion.close()
    else:
        flash("Datos incompletos para modificar inscripción.", "error")
    return redirect(url_for('gestionar_inscripciones'))

# Asignación de padres a estudiantes
@app.route('/admin/asignar_padre', methods=['GET', 'POST'])
def asignar_padre():
    conexion = create_connection()
    estudiantes = []
    padres = []
    mensaje_exito = None
    mensaje_error = None

    if conexion:
        cursor = conexion.cursor(dictionary=True)
        cursor.execute('SELECT id, nombre FROM estudiantes')
        estudiantes = cursor.fetchall()
        cursor.execute('SELECT id, nombre FROM padres')
```

```
        padres = cursor.fetchall()

        if request.method == 'POST':
            id_estudiante = request.form['id_estudiante']
            id_padre = request.form['id_padre']
            cursor.execute(
                'SELECT 1 FROM padres_estudiantes WHERE id_estudiante = %s AND id_padre = %s',
                (id_estudiante, id_padre)
            )
            existe = cursor.fetchone()
            if existe:
                mensaje_error = '¡Ya existe esta asignación!'
            else:
                cursor.execute(
                    'INSERT INTO padres_estudiantes (id_padre, id_estudiante) VALUES (%s, %s)',
                    (id_padre, id_estudiante)
                )
                conexion.commit()
                mensaje_exito = '¡Padre asignado correctamente!'
        cursor.close()
        conexion.close()
    return render_template(
        'admin/asignar_padre.html',
        estudiantes=estudiantes,
        padres=padres,
        mensaje_exito=mensaje_exito,
        mensaje_error=mensaje_error
    )

# -------------------------
# Rutas PROFESOR
# -------------------------
@app.route('/profesor/dashboard')
def profesor_dashboard():
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))

    id_profesor = session['usuario']['id']

    conexion = create_connection()
    estudiantes_por_materia = []
    promedios_por_materia = []
    if conexion:
        cursor = conexion.cursor()
        # Cantidad de estudiantes por materia (corrigiendo el WHERE)
        cursor.execute("""
            SELECT m.nombre, COUNT(DISTINCT n.id_estudiante)
            FROM materias m
```

```
            JOIN notas n ON n.id_materia = m.id
            WHERE n.id_profesor = %s
            GROUP BY m.nombre
        """, (id_profesor,))
        estudiantes_por_materia = cursor.fetchall()

        # Promedio de notas por materia (corrigiendo el WHERE)
        cursor.execute("""
            SELECT m.nombre, AVG(n.nota)
            FROM materias m
            JOIN notas n ON n.id_materia = m.id
            WHERE n.id_profesor = %s
            GROUP BY m.nombre
        """, (id_profesor,))
        promedios_por_materia = cursor.fetchall()
        cursor.close()
        conexion.close()

    # Prepara los datos para las gráficas
    labels = [row[0] for row in estudiantes_por_materia]
    data_estudiantes = [row[1] for row in estudiantes_por_materia]
    data_promedios = [float(row[1]) if row[1] is not None else 0 for row in promedios_por_materia]

    return render_template(
        'profesor/dashboard.html',
        labels=labels,
        data_estudiantes=data_estudiantes,
        data_promedios=data_promedios,
        usuario=session['usuario']
    )

@app.route('/profesor/obtener_estudiantes', methods=['GET', 'POST'])
def obtener_estudiantes():
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))

    materias = ProfesorController.obtener_materias()
    estudiantes = []
    id_materia = None

    if request.method == 'POST':
        id_materia = request.form.get('id_materia')
        if id_materia:
            estudiantes = ProfesorController.obtener_estudiantes_por_materia(id_materia)
        else:
            flash("Por favor, selecciona una materia.", "error")

    return render_template(
```

```
        'profesor/obtener_estudiantes.html',
        materias=materias,
        estudiantes=estudiantes,
        id_materia=id_materia,
        usuario=session['usuario']
    )


@app.route('/profesor/hoja_vida')
def profesor_hoja_vida():
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))

    id_profesor = session['usuario']['id']
    hoja = None
    conexion = create_connection()
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            cursor.execute("SELECT hoja_vida FROM profesores WHERE id = %s", (id_profesor,))
            fila = cursor.fetchone()
            if fila:
                hoja = fila.get('hoja_vida')
        except Exception as e:
            app.logger.error(f"Error obteniendo hoja de vida: {e}")
        finally:
            try:
                cursor.close()
            except Exception:
                pass
            try:
                conexion.close()
            except Exception:
                pass

    return render_template('profesor/HojadeVida.html', hoja=hoja, usuario=session['usuario'])


@app.route('/profesor/subir_hoja', methods=['POST'])
def subir_hoja():
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para realizar esta acción.", "error")
        return redirect(url_for('home'))

    id_profesor = session['usuario']['id']
    if 'hoja' not in request.files:
        flash('No se encontró ningún archivo.', 'error')
        return redirect(url_for('profesor_hoja_vida'))
```

```
    archivo = request.files['hoja']
    if archivo.filename == '':
        flash('No se seleccionó ningún archivo.', 'error')
        return redirect(url_for('profesor_hoja_vida'))

    if archivo and allowed_file(archivo.filename):
        filename = secure_filename(f"{id_profesor}_{int(time.time())}_{archivo.filename}")
        ruta = os.path.join(UPLOAD_FOLDER, filename)
        try:
            archivo.save(ruta)
            # Actualizar registro en BD
            conexion = create_connection()
            if conexion:
                cursor = conexion.cursor()
                try:
                    cursor.execute("UPDATE profesores SET hoja_vida = %s WHERE id = %s", (filename,
id_profesor))
                    conexion.commit()
                except Exception as e:
                    conexion.rollback()
                    app.logger.error(f"Error guardando nombre de archivo en BD: {e}")
                finally:
                    try:
                        cursor.close()
                    except Exception:
                        pass
                    try:
                        conexion.close()
                    except Exception:
                        pass

            flash('Hoja de vida subida correctamente.', 'success')
        except Exception as e:
            app.logger.error(f"Error al guardar archivo: {e}")
            flash('Ocurrió un error al subir el archivo.', 'error')
    else:
        flash('Tipo de archivo no permitido. Solo PDF.', 'error')

    return redirect(url_for('profesor_hoja_vida'))


@app.route('/profesor/actualizar_hoja', methods=['POST'])
def actualizar_hoja():
    # Reutiliza la lógica de subir_hoja (sobrescribe registro y archivo)
    return subir_hoja()


@app.route('/profesor/ver_hoja/<int:id_profesor>')
```

```
def ver_hoja(id_profesor):
    conexion = create_connection()
    filename = None
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            cursor.execute("SELECT hoja_vida FROM profesores WHERE id = %s", (id_profesor,))
            fila = cursor.fetchone()
            if fila:
                filename = fila.get('hoja_vida')
        except Exception as e:
            app.logger.error(f"Error obteniendo hoja de vida: {e}")
        finally:
            try:
                cursor.close()
            except Exception:
                pass
            try:
                conexion.close()
            except Exception:
                pass

    if not filename:
        flash('No existe hoja de vida para este profesor.', 'error')
        return redirect(url_for('profesor_hoja_vida'))

    # Enviar archivo desde carpeta de uploads
    return send_from_directory(UPLOAD_FOLDER, filename)

@app.route('/profesor/asignar_nota', methods=['GET', 'POST'])
def asignar_nota():
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))

    id_profesor = session['usuario']['id']
    materias = ProfesorController.obtener_materias_asignadas(id_profesor)
    estudiantes = []
    id_materia = None

    if request.method == 'POST':
        id_materia = request.form.get('id_materia')
        tipo_evaluacion = request.form.get('tipo_evaluacion')
        notas = request.form.getlist('nota[]')
        comentarios = request.form.getlist('comentario[]')
        ids_estudiantes = request.form.getlist('id_estudiante[]')

        # Verifica que la materia seleccionada esté entre las asignadas
        materia_ids = [str(m['id']) for m in materias]
```

```
        if id_materia not in materia_ids:
            flash("No tienes permisos para asignar notas en esta materia.", "error")
            return redirect(url_for('asignar_nota'))

        # Si se seleccionó materia pero no hay notas, mostrar estudiantes
        if id_materia and not notas:
            estudiantes = ProfesorController.obtener_estudiantes_por_materia(id_materia)
        # Si hay notas, procesar asignación
        elif notas and comentarios and ids_estudiantes and id_materia and tipo_evaluacion:
            from Controllers.nota_controller import NotaController
            for i in range(len(ids_estudiantes)):
                ProfesorController.asignar_nota(
                    id_estudiante=ids_estudiantes[i],
                    id_profesor=id_profesor,
                    id_materia=id_materia,
                    nota=notas[i],
                    tipo_evaluacion=tipo_evaluacion,
                    comentario=comentarios[i]
                )
            flash("Notas asignadas correctamente.", "success")
            return redirect(url_for('asignar_nota'))
        else:
            if not tipo_evaluacion:
                flash("Por favor, selecciona el tipo de evaluación.", "error")
            else:
                flash("Por favor, completa todos los campos.", "error")

    return render_template(
        'profesor/asignarNota.html',
        usuario=session['usuario'],
        materias=materias,
        estudiantes=estudiantes,
        id_materia=id_materia
    )

@app.route('/profesor/cambiar_nota', methods=['GET', 'POST'])
def cambiar_nota():
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    materias = ProfesorController.obtener_materias()
    estudiantes = []
    if request.method == 'POST':
        id_materia = request.form.get('id_materia')
        ids_estudiantes = request.form.getlist('id_estudiante[]')
        nuevas_notas = request.form.getlist('nueva_nota[]')
        comentarios = request.form.getlist('comentario[]')
        id_profesor = session['usuario']['id']
```

```
        if ids_estudiantes and nuevas_notas and comentarios:
            for i in range(len(ids_estudiantes)):
                ProfesorController.cambiar_nota(
                    ids_estudiantes[i],
                    id_materia,
                    nuevas_notas[i],
                    comentarios[i],
                    id_profesor
                )
            flash("Notas y comentarios actualizados correctamente.", "success")
            return redirect(url_for('cambiar_nota'))
        elif id_materia:
            estudiantes = ProfesorController.obtener_estudiantes_por_materia(id_materia)
        else:
            flash("Por favor, selecciona una materia.", "error")

    return render_template('profesor/cambiarNota.html', materias=materias, estudiantes=estudiantes)


@app.route('/profesor/enviar_notificacion', methods=['GET', 'POST'])
def enviar_notificacion():
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))

    id_profesor = session['usuario']['id']
    materias = ProfesorController.obtener_materias_asignadas(id_profesor)

    if request.method == 'POST':
        tipo_envio = request.form.get('tipo_envio', 'individual')
        id_materia = request.form.get('id_materia')
        titulo = request.form.get('titulo')
        mensaje = request.form.get('mensaje')

        if not id_materia or not titulo or not mensaje:
            flash("Por favor completa todos los campos.", "error")
            return redirect(url_for('enviar_notificacion'))

        if tipo_envio == 'individual':
            id_estudiante = request.form.get('id_estudiante')
            if not id_estudiante:
                flash("Selecciona un estudiante.", "error")
                return redirect(url_for('enviar_notificacion'))

            from Controllers.notificacion_controller import NotificacionController
            resultado = NotificacionController.enviar_notificacion_a_estudiante(
                int(id_estudiante), id_profesor, titulo, mensaje
            )
            if resultado['success']:
```

```
            flash(resultado['message'], "success")
        else:
            flash(resultado['message'], "error")

    return redirect(url_for('enviar_notificacion'))

  return render_template('profesor/EnviarNotificacion.html', materias=materias)

@app.route('/profesor/enviar_notificacion_grupo', methods=['POST'])
def enviar_notificacion_grupo():
  if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
    flash("No tienes permisos para acceder a esta sección.", "error")
    return redirect(url_for('home'))

  id_profesor = session['usuario']['id']
  id_materia = request.form.get('id_materia')
  titulo = request.form.get('titulo')
  mensaje = request.form.get('mensaje')

  if not id_materia or not titulo or not mensaje:
    flash("Por favor completa todos los campos.", "error")
    return redirect(url_for('enviar_notificacion'))

  from Controllers.notificacion_controller import NotificacionController
  resultado = NotificacionController.enviar_notificacion_a_clase(
    int(id_materia), id_profesor, titulo, mensaje
  )

  if resultado['success']:
    flash(resultado['message'], "success")
  else:
    flash(resultado['message'], "error")

  return redirect(url_for('enviar_notificacion'))

@app.route('/api/estudiantes_por_materia/<int:id_materia>', methods=['GET'])
def estudiantes_por_materia(id_materia):
  """Retorna estudiantes inscritos en una materia en formato JSON."""
  estudiantes = ProfesorController.obtener_estudiantes_por_materia(id_materia)
  # Convertir a formato JSON con id y nombre
  result = [{'id': e['id'], 'nombre': e['estudiante']} for e in estudiantes]
  return jsonify(result)

# -------------------------
# API RUTAS PARA NOTIFICACIONES
# -------------------------
@app.route('/api/notificacion/marcar_leida/<int:id_notificacion>', methods=['POST'])
def api_marcar_notificacion_leida(id_notificacion):
  """API para marcar una notificación como leída."""
```

```
    if 'usuario' not in session:
        return jsonify({'success': False, 'message': 'No autorizado'}), 401

    from Controllers.notificacion_controller import NotificacionController
    resultado = NotificacionController.marcar_como_leida(id_notificacion)
    return jsonify(resultado)

@app.route('/api/notificacion/marcar_todas_leidas', methods=['POST'])
def api_marcar_todas_notificaciones_leidas():
    """API para marcar todas las notificaciones como leídas."""
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({'success': False, 'message': 'No autorizado'}), 401

    id_estudiante = session['usuario']['id']
    from Controllers.notificacion_controller import NotificacionController
    resultado = NotificacionController.marcar_todas_como_leidas(id_estudiante)
    return jsonify(resultado)

@app.route('/api/notificacion/eliminar/<int:id_notificacion>', methods=['DELETE'])
def api_eliminar_notificacion(id_notificacion):
    """API para eliminar una notificación."""
    if 'usuario' not in session:
        return jsonify({'success': False, 'message': 'No autorizado'}), 401

    from Controllers.notificacion_controller import NotificacionController
    resultado = NotificacionController.eliminar_notificacion(id_notificacion)
    return jsonify(resultado)

@app.route('/api/notificacion/sin_leer', methods=['GET'])
def api_notificaciones_sin_leer():
    """API para obtener el conteo de notificaciones sin leer."""
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({'count': 0}), 401

    id_estudiante = session['usuario']['id']
    from Controllers.notificacion_controller import NotificacionController
    count = NotificacionController.obtener_conteo_no_leidas(id_estudiante)
    return jsonify({'count': count})

# -------------------------
# Rutas ÍNDICES DE APRENDIZAJE
# -------------------------
@app.route('/profesor/indices', methods=['GET', 'POST'])
def indices():
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))

    id_profesor = session['usuario']['id']
```

```
    if request.method == 'POST':
        id_materia = request.form.get('id_materia')
        if not id_materia:
            flash("Selecciona una materia.", "error")
            return redirect(url_for('indices'))
        session['id_materia_indices'] = int(id_materia)

    id_materia = session.get('id_materia_indices')
    materias = ProfesorController.obtener_materias_asignadas(id_profesor)
    indices_list = []

    if id_materia:
        try:
            indices_list = IndicesController.obtener_indices_con_evaluaciones(id_materia)
        except Exception as e:
            flash(f"Error al cargar índices: {str(e)}", "error")

    return render_template('profesor/indices.html', materias=materias, indices=indices_list,
id_materia_actual=id_materia)

@app.route('/profesor/crear_indice', methods=['GET', 'POST'])
def crear_indice():
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))

    id_profesor = session['usuario']['id']
    id_materia = session.get('id_materia_indices')

    if not id_materia:
        flash("Selecciona una materia primero.", "error")
        return redirect(url_for('indices'))

    if request.method == 'POST':
        nombre = request.form.get('nombre')
        descripcion = request.form.get('descripcion')
        parcial = request.form.get('parcial')
        porcentaje = request.form.get('porcentaje')

        try:
            porcentaje = float(porcentaje) if porcentaje else 0
        except ValueError:
            flash("El porcentaje debe ser un número válido.", "error")
            return redirect(url_for('crear_indice'))

        resultado = IndicesController.crear_indice(
            id_materia=int(id_materia),
            id_profesor=id_profesor,
```

```
            nombre=nombre,
            descripcion=descripcion,
            parcial=parcial,
            porcentaje=porcentaje
        )

        if resultado['success']:
            flash(resultado['message'], "success")
            return redirect(url_for('indices'))
        else:
            flash(resultado['message'], "error")

    return render_template('profesor/crear_indice.html', id_materia=id_materia)

@app.route('/profesor/evaluar_indice/<int:id_indice>', methods=['GET', 'POST'])
def evaluar_indice(id_indice):
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))

    id_profesor = session['usuario']['id']

    try:
        indice = IndicesController.obtener_indice_por_id(id_indice)
        if not indice or indice['id_profesor'] != id_profesor:
            flash("Índice no encontrado o no tienes permisos.", "error")
            return redirect(url_for('indices'))

        if request.method == 'POST':
            porcentaje_dominio = request.form.get('porcentaje_dominio')
            comentario = request.form.get('comentario', '').strip()

            try:
                porcentaje_dominio = float(porcentaje_dominio)
                if not (0 <= porcentaje_dominio <= 100):
                    flash("El porcentaje de dominio debe estar entre 0 y 100.", "error")
                    return redirect(url_for('evaluar_indice', id_indice=id_indice))
            except ValueError:
                flash("El porcentaje debe ser un número válido.", "error")
                return redirect(url_for('evaluar_indice', id_indice=id_indice))

            resultado = IndicesController.guardar_evaluacion_indice(
                id_indice=id_indice,
                id_profesor=id_profesor,
                porcentaje_dominio=porcentaje_dominio,
                comentario=comentario
            )

            if resultado['success']:
```

```
            flash(resultado['message'], "success")
            return redirect(url_for('indices'))
        else:
            flash(resultado['message'], "error")

    evaluaciones = IndicesController.obtener_evaluaciones_indice(id_indice)
    ultima_evaluacion = evaluaciones[0] if evaluaciones else None

    return render_template('profesor/evaluar_indice.html', indice=indice, evaluaciones=evaluaciones,
ultima_evaluacion=ultima_evaluacion)

  except Exception as e:
    flash(f"Error al cargar índice: {str(e)}", "error")
    return redirect(url_for('indices'))

@app.route('/profesor/editar_indice/<int:id_indice>', methods=['GET', 'POST'])
def editar_indice(id_indice):
  if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
    flash("No tienes permisos para acceder a esta sección.", "error")
    return redirect(url_for('home'))

  id_profesor = session['usuario']['id']

  try:
    indice = IndicesController.obtener_indice_por_id(id_indice)
    if not indice or indice['id_profesor'] != id_profesor:
      flash("Índice no encontrado o no tienes permisos.", "error")
      return redirect(url_for('indices'))

    if request.method == 'POST':
      nombre = request.form.get('nombre')
      descripcion = request.form.get('descripcion')
      parcial = request.form.get('parcial')
      porcentaje = request.form.get('porcentaje')

      try:
        porcentaje = float(porcentaje) if porcentaje else 0
      except ValueError:
        flash("El porcentaje debe ser un número válido.", "error")
        return redirect(url_for('editar_indice', id_indice=id_indice))

      resultado = IndicesController.actualizar_indice(
        id_indice=id_indice,
        nombre=nombre,
        descripcion=descripcion,
        parcial=parcial,
        porcentaje=porcentaje
      )
```

```
            if resultado['success']:
                flash(resultado['message'], "success")
                return redirect(url_for('indices'))
            else:
                flash(resultado['message'], "error")

        return render_template('profesor/crear_indice.html', indice=indice,
id_materia=indice['id_materia'])

    except Exception as e:
        flash(f"Error al cargar índice: {str(e)}", "error")
        return redirect(url_for('indices'))

@app.route('/profesor/eliminar_indice/<int:id_indice>')
def eliminar_indice(id_indice):
    if 'usuario' not in session or session['usuario']['tipo'] != 'profesor':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))

    id_profesor = session['usuario']['id']

    try:
        indice = IndicesController.obtener_indice_por_id(id_indice)
        if not indice or indice['id_profesor'] != id_profesor:
            flash("Índice no encontrado o no tienes permisos.", "error")
            return redirect(url_for('indices'))

        resultado = IndicesController.eliminar_indice(id_indice)

        if resultado['success']:
            flash(resultado['message'], "success")
        else:
            flash(resultado['message'], "error")

        return redirect(url_for('indices'))

    except Exception as e:
        flash(f"Error al eliminar índice: {str(e)}", "error")
        return redirect(url_for('indices'))

# -------------------------
# Rutas ESTUDIANTE
# -------------------------
@app.route('/estudiante/dashboard')
def estudiante_dashboard():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return redirect(url_for('home'))
    try:
        id_estudiante = session['usuario']['id']
```

```
        # Obtener todos los datos del dashboard usando el controlador
        datos_dashboard = EstudianteController.obtener_datos_dashboard(id_estudiante)

        return render_template(
            'estudiante/dashboard.html',
            usuario=session['usuario'],
            tareas_pendientes=datos_dashboard['tareas_pendientes'],
            asistencia=datos_dashboard['asistencia'],
            materias=datos_dashboard['estadisticas'],
            promedio_general=datos_dashboard['promedio_general'],
            estadisticas=datos_dashboard['estadisticas'],
            notificaciones=datos_dashboard['notificaciones']
        )
    except Exception as e:
        app.logger.error(f"Error en estudiante dashboard: {str(e)}")
        flash("Ocurrió un error al cargar el dashboard", "error")
        return redirect(url_for('home'))
    finally:
        pass


# Ruta: Mis Clases (nueva - renderiza la plantilla personalizada)
@app.route('/estudiante/clases')
def clases_estudiante():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    id_estudiante = session['usuario']['id']
    try:
        horario = EstudianteController.obtener_horario(id_estudiante)
        return render_template('estudiante/mis_clases.html', usuario=session['usuario'], horario=horario)
    except Exception as e:
        app.logger.error(f"Error al cargar Mis Clases: {e}")
        flash("Ocurrió un error al cargar tus clases.", "error")
        return redirect(url_for('estudiante_dashboard'))


# Ruta: Asignaturas (lista de materias disponibles)
@app.route('/estudiante/asignaturas')
def asignaturas_estudiante():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    usuario = session['usuario']
    conexion = create_connection()
    materias = []
    try:
        if conexion:
            cursor = conexion.cursor(dictionary=True)
```

```
            cursor.execute("SELECT id, nombre, descripcion FROM materias ORDER BY nombre")
            materias = cursor.fetchall()
            # Obtener permiso de inscripción del estudiante para mostrar botones
            cursor.execute("SELECT puede_inscribirse FROM estudiantes WHERE id=%s",
(usuario['id'],))
            permiso = cursor.fetchone()
            usuario['puede_inscribirse'] = permiso['puede_inscribirse'] if permiso and 'puede_inscribirse' in
permiso else False
            session['usuario'] = usuario
    except Exception as e:
        app.logger.error(f"Error al cargar Asignaturas: {e}")
        flash("Ocurrió un error al cargar las asignaturas.", "error")
    finally:
        try:
            cursor.close()
        except Exception:
            pass
        try:
            if conexion:
                conexion.close()
        except Exception:
            pass
    return render_template('estudiante/mis_asignaturas.html', usuario=usuario, materias=materias)


# Ruta: Mis Calificaciones (nueva)
@app.route('/estudiante/calificaciones')
def calificaciones_estudiante():
    app.logger.debug(f"Entering calificaciones_estudiante; session usuario: {session.get('usuario')}")
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    id_estudiante = session['usuario']['id']
    try:
        from Controllers.nota_controller import NotaController
        # Obtener notas agrupadas por materia
        materias_agrupadas = NotaController.obtener_notas_por_materia_agrupadas(id_estudiante)

        app.logger.debug(f"Calificaciones obtained: materias={len(materias_agrupadas)}")
        return render_template(
            'estudiante/mis_calificaciones.html',
            usuario=session['usuario'],
            materias_agrupadas=materias_agrupadas
        )
    except Exception as e:
        tb = traceback.format_exc()
        app.logger.error(f"Error al cargar Calificaciones: {e}\n{tb}")
        flash("Ocurrió un error al cargar tus calificaciones.", "error")
        return redirect(url_for('estudiante_dashboard'))
```

```
@app.route('/estudiante/calificaciones/descargar')
def descargar_calificaciones():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    id_estudiante = session['usuario']['id']
    formato = request.args.get('formato', 'csv')
    try:
        datos = EstudianteController.obtener_calificaciones_detalle(id_estudiante)
        notas = datos.get('notas', [])
        # Build CSV
        if formato == 'csv':
            # Use semicolon delimiter and UTF-8 BOM so Excel (locale-dependent) opens columns
correctly
            si = io.StringIO()
            writer = csv.writer(si, delimiter=';')
            writer.writerow(['Materia', 'Nota', 'Comentario'])
            for n in notas:
                writer.writerow([n.get('materia'), n.get('nota'), n.get('comentario', '')])
            output = si.getvalue()
            si.close()
            # Prepend BOM so Excel recognizes UTF-8 and splits columns correctly in many locales
            output = '\ufeff' + output
            response = app.make_response(output)
            response.headers['Content-Disposition'] = f'attachment;
filename=calificaciones_{id_estudiante}.csv'
            response.headers['Content-Type'] = 'text/csv; charset=utf-8'
            return response
        else:
            flash('Formato no soportado para descarga.', 'error')
            return redirect(url_for('calificaciones_estudiante'))
    except Exception as e:
        app.logger.error(f"Error al generar descarga de calificaciones: {e}")
        flash('Ocurrió un error al generar la descarga.', 'error')
        return redirect(url_for('calificaciones_estudiante'))


# Ruta: Mis Notificaciones (nueva)
# Ruta: Mis Notificaciones (nueva)
@app.route('/estudiante/notificaciones')
def notificaciones_estudiante():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    id_estudiante = session['usuario']['id']
    try:
        from Controllers.notificacion_controller import NotificacionController
```

```
        notificaciones = NotificacionController.obtener_notificaciones_estudiante(id_estudiante)
        return render_template('estudiante/mis_notificaciones.html', usuario=session['usuario'],
notificaciones=notificaciones)
    except Exception as e:
        app.logger.error(f"Error al cargar Notificaciones: {e}")
        flash("Ocurrió un error al cargar tus notificaciones.", "error")
        return redirect(url_for('estudiante_dashboard'))


# API: Obtener detalles de notificación con mensajes
@app.route('/api/notificacion/<int:id_notificacion>')
def api_obtener_notificacion(id_notificacion):
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({"success": False, "message": "No autorizado"}), 403

    id_estudiante = session['usuario']['id']
    try:
        from Controllers.mensaje_controller import MensajeController
        resultado = MensajeController.obtener_notificacion_con_detalles(id_notificacion, id_estudiante)

        if resultado:
            return jsonify({
                "success": True,
                "notificacion": resultado['notificacion'],
                "mensajes": resultado['mensajes']
            })
        else:
            return jsonify({"success": False, "message": "Notificación no encontrada"}), 404
    except Exception as e:
        app.logger.error(f"Error al obtener notificación: {e}")
        return jsonify({"success": False, "message": str(e)}), 500


# API: Enviar respuesta a notificación
@app.route('/api/notificacion/<int:id_notificacion>/responder', methods=['POST'])
def api_responder_notificacion(id_notificacion):
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({"success": False, "message": "No autorizado"}), 403

    id_estudiante = session['usuario']['id']
    data = request.get_json()

    try:
        if not data or 'contenido' not in data or 'id_profesor' not in data or 'id_materia' not in data:
            return jsonify({"success": False, "message": "Datos incompletos"}), 400

        from Controllers.mensaje_controller import MensajeController
        resultado = MensajeController.enviar_respuesta(
            id_notificacion,
```

```
            id_estudiante,
            data['id_profesor'],
            data['id_materia'],
            data['contenido']
        )

        return jsonify(resultado)
    except Exception as e:
        app.logger.error(f"Error al responder notificación: {e}")
        return jsonify({"success": False, "message": str(e)}), 500


# API: Obtener materias del estudiante
@app.route('/api/estudiante/materias')
def api_materias_estudiante():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({"success": False, "message": "No autorizado"}), 403

    id_estudiante = session['usuario']['id']
    try:
        from Controllers.estudiante_controller import EstudianteController
        materias = EstudianteController.obtener_materias_asignadas(id_estudiante)

        return jsonify({
            "success": True,
            "materias": materias
        })
    except Exception as e:
        app.logger.error(f"Error al obtener materias: {e}")
        return jsonify({"success": False, "message": str(e)}), 500


# API: Enviar mensaje inicial a profesor
@app.route('/api/mensaje/enviar', methods=['POST'])
def api_enviar_mensaje():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({"success": False, "message": "No autorizado"}), 403

    id_estudiante = session['usuario']['id']
    data = request.get_json()

    try:
        if not data or 'id_profesor' not in data or 'id_materia' not in data or \
           'titulo' not in data or 'contenido' not in data:
            return jsonify({"success": False, "message": "Datos incompletos"}), 400

        from Controllers.mensaje_controller import MensajeController
        resultado = MensajeController.enviar_mensaje_inicial(
            id_estudiante,
```

```
            data['id_profesor'],
            data['id_materia'],
            data['titulo'],
            data['contenido']
        )

        return jsonify(resultado)
    except Exception as e:
        app.logger.error(f"Error al enviar mensaje: {e}")
        return jsonify({"success": False, "message": str(e)}), 500


# Ruta: Mis Tareas (nueva)
@app.route('/estudiante/tareas')
def tareas_estudiante():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    id_estudiante = session['usuario']['id']
    try:
        tareas = EstudianteController.obtener_tareas_detalle(id_estudiante)
        return render_template('estudiante/mis_tareas.html', usuario=session['usuario'], tareas=tareas)
    except Exception as e:
        app.logger.error(f"Error al cargar Tareas: {e}")
        flash("Ocurrió un error al cargar tus tareas.", "error")
        return redirect(url_for('estudiante_dashboard'))


# Ajustar ruta existente de horario para usar la plantilla nueva `mi_horario.html`

@app.route('/estudiante/ver_notas')
def ver_notas():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    usuario = session['usuario']
    conexion = create_connection()
    notas = []
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            query = """
                SELECT m.nombre AS materia,
                       p.nombre AS profesor,
                       n.nota
                FROM notas n
                JOIN materias m ON n.id_materia = m.id
                LEFT JOIN profesores p ON n.id_profesor = p.id
                WHERE n.id_estudiante = %s
```

```
            """
            cursor.execute(query, (usuario['id'],))
            notas = cursor.fetchall()
        except Exception as e:
            flash(f"Error al obtener las notas: {e}", "error")
        finally:
            cursor.close()
            conexion.close()
    return render_template('estudiante/ver_notas.html', usuario=usuario, notas=notas)

@app.route('/estudiante/ver_clases')
def ver_clases():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    usuario = session['usuario']
    conexion = create_connection()
    horario = []
    materias = []
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            query = """
                SELECT h.dia_semana, h.hora_inicio, h.hora_fin, m.nombre AS materia, p.nombre AS
profesor
                FROM horarios h
                JOIN materias m ON h.id_materia = m.id
                LEFT JOIN profesores p ON h.id_profesor = p.id
                WHERE h.id_estudiante = %s
                ORDER BY FIELD(h.dia_semana,
'Lunes','Martes','Miércoles','Jueves','Viernes','Sábado','Domingo'), h.hora_inicio
            """
            cursor.execute(query, (usuario['id'],))
            horario = cursor.fetchall()
        except Exception as e:
            flash(f"Error al obtener el horario: {e}", "error")
        finally:
            cursor.close()
            conexion.close()
    return render_template('estudiante/ver_clases.html', usuario=usuario, horario=horario)

@app.route('/estudiante/horario')
def horario_estudiante():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        flash("No tienes permisos para acceder a esta sección.", "error")
        return redirect(url_for('home'))
    usuario = session['usuario']
    conexion = create_connection()
    horario = []
```

```
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            query = """
                SELECT h.dia_semana, h.hora_inicio, h.hora_fin, m.nombre AS materia, p.nombre AS
profesor
                FROM horarios h
                JOIN materias m ON h.id_materia = m.id
                LEFT JOIN profesores p ON h.id_profesor = p.id
                WHERE h.id_estudiante = %s
                ORDER BY FIELD(h.dia_semana,
'Lunes','Martes','Miércoles','Jueves','Viernes','Sábado','Domingo'), h.hora_inicio
            """
            cursor.execute(query, (usuario['id'],))
            horario = cursor.fetchall()
            # Obtener materias disponibles para inscribirse
            cursor.execute("SELECT id, nombre FROM materias ORDER BY nombre")
            materias = cursor.fetchall()
            # Obtener permiso de inscripción del estudiante
            cursor.execute("SELECT puede_inscribirse FROM estudiantes WHERE id=%s",
(usuario['id'],))
            permiso = cursor.fetchone()
            usuario['puede_inscribirse'] = permiso['puede_inscribirse'] if permiso and 'puede_inscribirse' in
permiso else False
            # Asegura que la sesión refleje el permiso (para la plantilla)
            session['usuario'] = usuario
        except Exception as e:
            flash(f"Error al obtener el horario: {e}", "error")
        finally:
            cursor.close()
            conexion.close()
    return render_template('estudiante/mi_horario.html', usuario=usuario, horario=horario,
materias=materias)


@app.route('/estudiante/inscribir', methods=['POST'])
def inscribir_materia():
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        flash("No tienes permisos para realizar esta acción.", "error")
        return redirect(url_for('home'))
    usuario = session['usuario']
    id_materia = request.form.get('id_materia')
    if not id_materia:
        flash("Materia no especificada.", "error")
        return redirect(url_for('horario_estudiante'))

    conexion = create_connection()
    if not conexion:
        flash("Error de conexión con la base de datos.", "error")
```

```
        return redirect(url_for('horario_estudiante'))

    try:
        cursor = conexion.cursor(dictionary=True)
        # Verificar si el estudiante tiene permiso para inscribirse
        cursor.execute("SELECT puede_inscribirse FROM estudiantes WHERE id=%s", (usuario['id'],))
        fila = cursor.fetchone()
        if not fila or not fila.get('puede_inscribirse'):
            flash("No estás autorizado para inscribirte. Contacta al administrador.", "error")
            return redirect(url_for('horario_estudiante'))

        # Insertar la solicitud/inscripción (evita duplicados)
        cursor.execute("INSERT IGNORE INTO inscripciones (id_estudiante, id_materia) VALUES (%s,
%s)", (usuario['id'], id_materia))
        conexion.commit()
        flash("Solicitud de inscripción enviada. Espera la aprobación del administrador.", "success")
    except Exception as e:
        app.logger.error(f"Error al inscribirse en la materia: {e}")
        flash(f"Ocurrió un error al inscribirte: {e}", "error")
    finally:
        try:
            cursor.close()
        except Exception:
            pass
        try:
            conexion.close()
        except Exception:
            pass

    return redirect(url_for('horario_estudiante'))
# --------------------------
# Rutas PADRE
# --------------------------
@app.route('/padre/dashboard')
def padre_dashboard():
    if 'usuario' not in session or session['usuario']['tipo'] != 'padre':
        return redirect(url_for('home'))
    try:
        conexion = create_connection()
        if not conexion:
            flash("Error de conexión con la base de datos", "error")
            return render_template('padre/dashboard.html', usuario=session['usuario'])
        with conexion.cursor(dictionary=True) as cursor:
            estudiantes = []
            cursor.execute("SHOW TABLES LIKE 'padres_estudiantes'")
            if cursor.fetchone():
                cursor.execute("""
                    SELECT e.id, e.nombre
                    FROM padres_estudiantes pe
```

```
                JOIN estudiantes e ON pe.id_estudiante = e.id
                WHERE pe.id_padre = %s
            """, (session['usuario']['id'],))
            estudiantes = cursor.fetchall()
        return render_template('padre/dashboard.html', usuario=session['usuario'],
estudiantes=estudiantes)
    except Exception as e:
        app.logger.error(f"Error en padre dashboard: {str(e)}")
        flash("Ocurrió un error al cargar el dashboard", "error")
        return render_template('padre/dashboard.html', usuario=session['usuario'])
    finally:
        if 'conexion' in locals() and conexion:
            conexion.close()

@app.route('/padre/hijos')
def ver_hijos():
    if 'usuario' not in session or session['usuario']['tipo'] != 'padre':
        return redirect(url_for('home'))
    padre_id = session['usuario']['id']
    conexion = create_connection()
    hijos = []
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            cursor.execute("""
                SELECT e.nombre
                FROM padres_estudiantes pe
                JOIN estudiantes e ON pe.id_estudiante = e.id
                WHERE pe.id_padre = %s
            """, (padre_id,))
            hijos = cursor.fetchall()
        except Exception as e:
            hijos = []
        finally:
            cursor.close()
            conexion.close()
    return render_template('padre/ver_hijos.html', hijos=hijos)

@app.route('/ver_horarios')
def ver_horarios():
    if 'usuario' not in session or session['usuario']['tipo'] != 'padre':
        return redirect(url_for('home'))
    padre_id = session['usuario']['id']
    conexion = create_connection()
    materias_por_estudiante = {}
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            # 1. Obtener los hijos del padre
```

```
            cursor.execute("""
                SELECT e.id, e.nombre
                FROM padres_estudiantes pe
                JOIN estudiantes e ON pe.id_estudiante = e.id
                WHERE pe.id_padre = %s
            """, (padre_id,))
            estudiantes = cursor.fetchall()
            if estudiantes:
                for estudiante in estudiantes:
                    # 2. Obtener materias inscritas de cada hijo
                    cursor.execute("""
                        SELECT m.id, m.nombre, m.descripcion
                        FROM inscripciones i
                        JOIN materias m ON i.id_materia = m.id
                        WHERE i.id_estudiante = %s
                    """, (estudiante['id'],))
                    materias = cursor.fetchall()
                    materias_por_estudiante[estudiante['nombre']] = materias
        except Exception as e:
            materias_por_estudiante = {}
        finally:
            cursor.close()
            conexion.close()
    return render_template('padre/ver_horarios.html', materias_por_estudiante=materias_por_estudiante)


@app.route('/padre/ver_descripcion')
def ver_descripcion():
    if 'usuario' not in session or session['usuario']['tipo'] != 'padre':
        return redirect(url_for('home'))
    padre_id = session['usuario']['id']
    conexion = create_connection()
    notas = []
    if conexion:
        try:
            cursor = conexion.cursor(dictionary=True)
            # Obtener los hijos del padre
            cursor.execute("""
                SELECT e.id, e.nombre
                FROM padres_estudiantes pe
                JOIN estudiantes e ON pe.id_estudiante = e.id
                WHERE pe.id_padre = %s
            """, (padre_id,))
            estudiantes = cursor.fetchall()
            if estudiantes:
                for estudiante in estudiantes:
                    # Obtener notas y comentarios de cada hijo
                    cursor.execute("""
```

```
                    SELECT m.nombre AS nombre_materia, n.nota, n.comentario, %s AS
nombre_estudiante
                    FROM notas n
                    JOIN materias m ON n.id_materia = m.id
                    WHERE n.id_estudiante = %s
                """, (estudiante['nombre'], estudiante['id']))
                notas += cursor.fetchall()
        finally:
            cursor.close()
            conexion.close()
    return render_template('padre/ver_descripcion.html', notas=notas)

# -------------------------
# APIs de Eventos (Calendario)
# -------------------------

@app.route('/api/eventos/crear', methods=['POST'])
def crear_evento():
    """Crear un nuevo evento en el calendario del estudiante"""
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({'success': False, 'message': 'No autorizado'}), 401

    try:
        datos = request.get_json()
        id_estudiante = session['usuario']['id']
        resultado = EventoController.crear_evento(id_estudiante, datos)
        return jsonify(resultado)
    except Exception as e:
        return jsonify({'success': False, 'message': str(e)}), 500

@app.route('/api/eventos/obtener', methods=['GET'])
def obtener_eventos():
    """Obtener todos los eventos del estudiante"""
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({'success': False, 'message': 'No autorizado'}), 401

    try:
        id_estudiante = session['usuario']['id']
        resultado = EventoController.obtener_eventos(id_estudiante)
        return jsonify(resultado)
    except Exception as e:
        return jsonify({'success': False, 'message': str(e)}), 500

@app.route('/api/eventos/obtener/<int:evento_id>', methods=['GET'])
def obtener_evento(evento_id):
    """Obtener un evento específico"""
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({'success': False, 'message': 'No autorizado'}), 401
```

```
    try:
        id_estudiante = session['usuario']['id']
        resultado = EventoController.obtener_evento(evento_id, id_estudiante)
        return jsonify(resultado)
    except Exception as e:
        return jsonify({'success': False, 'message': str(e)}), 500

@app.route('/api/eventos/actualizar/<int:evento_id>', methods=['PUT'])
def actualizar_evento(evento_id):
    """Actualizar un evento existente"""
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({'success': False, 'message': 'No autorizado'}), 401

    try:
        datos = request.get_json()
        id_estudiante = session['usuario']['id']
        resultado = EventoController.actualizar_evento(evento_id, id_estudiante, datos)
        return jsonify(resultado)
    except Exception as e:
        return jsonify({'success': False, 'message': str(e)}), 500

@app.route('/api/eventos/eliminar/<int:evento_id>', methods=['DELETE'])
def eliminar_evento(evento_id):
    """Eliminar un evento"""
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({'success': False, 'message': 'No autorizado'}), 401

    try:
        id_estudiante = session['usuario']['id']
        resultado = EventoController.eliminar_evento(evento_id, id_estudiante)
        return jsonify(resultado)
    except Exception as e:
        return jsonify({'success': False, 'message': str(e)}), 500

@app.route('/api/eventos/fecha/<fecha>', methods=['GET'])
def obtener_eventos_fecha(fecha):
    """Obtener eventos de una fecha específica"""
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({'success': False, 'message': 'No autorizado'}), 401

    try:
        id_estudiante = session['usuario']['id']
        resultado = EventoController.obtener_eventos_por_fecha(id_estudiante, fecha)
        return jsonify(resultado)
    except Exception as e:
        return jsonify({'success': False, 'message': str(e)}), 500

@app.route('/api/eventos/mes/<int:anio>/<int:mes>', methods=['GET'])
def obtener_eventos_mes(anio, mes):
```

```
    """Obtener eventos del mes"""
    if 'usuario' not in session or session['usuario']['tipo'] != 'estudiante':
        return jsonify({'success': False, 'message': 'No autorizado'}), 401

    try:
        id_estudiante = session['usuario']['id']
        resultado = EventoController.obtener_eventos_mes(id_estudiante, anio, mes)
        return jsonify(resultado)
    except Exception as e:
        return jsonify({'success': False, 'message': str(e)}), 500

# -------------------------
# Main
# -------------------------
if __name__ == '__main__':
    app.run(debug=True)
```

Routes/auth_routes.py → se borro el codigo

Controllers/autenticacion.py:

```
"""
   Controlador de autenticación para la aplicación.
   Provee métodos estáticos para validar credenciales y devolver instancias
   de usuario apropiadas según el tipo o el administrador autenticado.
   Métodos:
   - login(correo, contraseña):
     Valida el inicio de sesión de usuarios no administrativos.
     - Parámetros:
       - correo (str): correo electrónico del usuario.
       - contraseña (str): contraseña en texto plano a verificar contra el hash
almacenado.
     - Comportamiento:
       - Busca un objeto Usuario mediante Usuario.obtener_por_correo(correo).
       - Verifica la contraseña usando check_password_hash(contrasena_almacenada,
contraseña).
       - Si la verificación es correcta, devuelve una instancia correspondiente
         a la subclase del usuario (Profesor, Estudiante o Padre) construida
         con los datos del usuario.
     - Retorno:
       - Instancia de Profesor/Estudiante/Padre si la autenticación es exitosa.
       - None si no existe el usuario, la contraseña es incorrecta o el tipo
         de usuario no coincide con los esperados.
     - Notas:
       - No crea sesiones ni tokens; solo devuelve un objeto representativo.
       - Puede propagar excepciones si Usuario.obtener_por_correo u otras
         dependencias fallan.
   - login_por_correo(correo, contraseña):
     Valida el inicio de sesión de administradores por correo y contraseña.
     - Parámetros:
       - correo (str): correo electrónico del administrador.
```

```
        - contraseña (str): contraseña en texto plano a verificar contra el hash
almacenado.
    - Comportamiento:
        - Intenta obtener un administrador mediante Admin.obtener_por_correo(correo).
        - Verifica la contraseña usando check_password_hash(contraseña_almacenada,
contraseña).
        - Incluye impresiones de depuración para indicar si se encontró el administrador
         y si la contraseña es válida o no.
        - Atrapa excepciones internas y las registra (impresión), devolviendo None en
caso de error.
    - Retorno:
        - Objeto administrador si la autenticación es exitosa.
        - None si no se encuentra el administrador, la contraseña es incorrecta o ocurre
un error.
    - Notas:
        - Igual que login, no gestiona sesiones ni tokens; se limita a validar
credenciales.
        - Contiene logs/prints de depuración que deberían retirarse o reemplazarse
         por un sistema de logging en producción.
    """
from Models.admin import Admin
from Models.profesor import Profesor
from Models.estudiante import Estudiante
from Models.padre import Padre
from werkzeug.security import check_password_hash

class AutenticacionController:

    @staticmethod
    def login(correo, contraseña):
        """Valida el inicio de sesión basado en correo y contraseña."""
        # Verifica si el correo pertenece a un usuario (no administradores)
        usuario = Usuario.obtener_por_correo(correo)
        if usuario and check_password_hash(usuario.contraseña, contraseña):
            if usuario.tipo == 'profesor':
                return Profesor(usuario.id, usuario.nombre, usuario.correo,
usuario.contraseña)
            elif usuario.tipo == 'estudiante':
                return Estudiante(usuario.id, usuario.nombre, usuario.correo,
usuario.contraseña)
            elif usuario.tipo == 'padre':
                return Padre(usuario.id, usuario.nombre, usuario.correo, usuario.contraseña)
        return None

    @staticmethod
    def login_por_correo(correo, contraseña):
        """Valida el inicio de sesión de un administrador por correo y contraseña."""
```

```
    try:
        administrador = Admin.obtener_por_correo(correo)
        if administrador:
            print(f"Administrador encontrado: {administrador.nombre}")  # Depuración
            if check_password_hash(administrador.contraseña, contraseña):
                print("Contraseña válida")  # Depuración
                return administrador
            else:
                print("Contraseña inválida")  # Depuración
        else:
            print(f"No se encontró un administrador con el correo: {correo}")  #
Depuración
    except Exception as e:
        print(f"Error en login_por_correo: {str(e)}")  # Depuración
    return None

def autenticar_usuario(correo, contraseña):
    # Buscar en administradores
    admin = Admin.obtener_por_correo(correo)
    if admin and check_password_hash(admin.contraseña, contraseña):
        return {'id': admin.id, 'nombre': admin.nombre, 'correo': admin.correo, 'tipo':
'admin'}

    # Buscar en profesores
    profesor = Profesor.obtener_por_correo(correo)
    if profesor and check_password_hash(profesor.contraseña, contraseña):
        return {'id': profesor.id, 'nombre': profesor.nombre, 'correo': profesor.correo,
'tipo': 'profesor'}

    # Buscar en estudiantes
    estudiante = Estudiante.obtener_por_correo(correo)
    if estudiante and check_password_hash(estudiante.contraseña, contraseña):
        return {'id': estudiante.id, 'nombre': estudiante.nombre, 'correo':
estudiante.correo, 'tipo': 'estudiante'}

    # Buscar en padres
    padre = Padre.obtener_por_correo(correo)
    if padre and check_password_hash(padre.contraseña, contraseña):
        return {'id': padre.id, 'nombre': padre.nombre, 'correo': padre.correo, 'tipo':
'padre'}

    return None
```

Models/usuario.py:

```
from pymysql import Error
from Config.database_connection import create_connection
from Models.admin import Admin
```

```python
from Models.profesor import Profesor
from Models.estudiante import Estudiante
from Models.padre import Padre
from werkzeug.security import check_password_hash

class Usuario:
  def __init__(self, id=None, nombre=None, correo=None, contraseña=None, tipo=None):
    self.id = id
    self.nombre = nombre
    self.correo = correo
    self.contraseña = contraseña
    self.tipo = tipo

  @classmethod
  def obtener_por_correo(cls, correo):
    conexion = create_connection()
    if conexion:
      try:
        cursor = conexion.cursor(dictionary=True)
        query = """
        SELECT 'admin' as tipo, id, nombre, correo, contraseña FROM administradores
WHERE correo = %s
        UNION SELECT 'profesor' as tipo, id, nombre, correo, contraseña FROM
profesores WHERE correo = %s
        UNION SELECT 'estudiante' as tipo, id, nombre, correo, contraseña FROM
estudiantes WHERE correo = %s
        UNION SELECT 'padre' as tipo, id, nombre, correo, contraseña FROM padres WHERE
correo = %s
        """
        cursor.execute(query, (correo, correo, correo, correo))
        usuario = cursor.fetchone()
        if usuario:
          return cls(
            id=usuario['id'],
            nombre=usuario['nombre'],
            correo=usuario['correo'],
            contraseña=usuario['contraseña'],
            tipo=usuario['tipo']
          )
        return None
      except Error as e:
        print(f"Error al obtener usuario: {e}")
        return None
      finally:
        if conexion.is_connected():
          cursor.close()
          conexion.close()
```

```python
  @staticmethod
  def autenticar(correo, contraseña):
      # Primero busca en administradores
      admin = Admin.obtener_por_correo(correo)
      if admin and check_password_hash(admin.contraseña, contraseña):
          admin.rol = 'admin'
          return admin
      # Luego busca en usuarios generales
      usuario = Usuario.obtener_por_correo(correo)
      if usuario and check_password_hash(usuario.contraseña, contraseña):
          usuario.rol = usuario.tipo  # tipo: 'profesor', 'estudiante', 'padre'
          return usuario
      return None
```