Super user

Usuario (Email): admin@tudominio.com

Clave Inicial: admin\_password\_inicial\_segura

La casona de Oscar

Usuario 'walterarielpelourson@gmail.com' agregado con éxito. Contraseña inicial: Mjp1234$

Luis Cocina

Usuario 'ljn@gmail.com' agregado con éxito. Contraseña inicial: Lui1234$

def conectar\_db():

    conn = sqlite3.connect(DB\_NAME)

    conn.row\_factory = sqlite3.Row # Permite acceder a las columnas por nombre

    return conn

def crear\_tablas():

    """Crea las tablas de la base de datos si no existen y añade columnas si faltan."""

    conn = sqlite3.connect(DB\_NAME)

    cursor = conn.cursor()

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS platos (

            id\_plato INTEGER PRIMARY KEY AUTOINCREMENT,

            nombre TEXT NOT NULL,

            descripcion TEXT,

            precio REAL NOT NULL,

            activo INTEGER DEFAULT 1 -- 1 para activo, 0 para inactivo (eliminación lógica)

        )

    """)

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS repartidores (

            id\_repartidor INTEGER PRIMARY KEY AUTOINCREMENT,

            nombre TEXT NOT NULL,

            apellido TEXT NOT NULL,

            telefono TEXT,

            activo INTEGER DEFAULT 1 -- 1 para activo, 0 para inactivo

        )

    """)

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS pedidos (

            id\_pedido INTEGER PRIMARY KEY AUTOINCREMENT,

            cliente\_nombre TEXT NOT NULL,

            cliente\_apellido TEXT NOT NULL,

            direccion\_entrega TEXT NOT NULL,

            es\_envio INTEGER NOT NULL, -- 0 para retiro, 1 para envío

            horario\_entrega TEXT NOT NULL, -- Formato ISO: YYYY-MM-DD HH:MM:SS

            costo\_envio REAL NOT NULL,

            costo\_total REAL NOT NULL,

            forma\_pago TEXT NOT NULL,

            estado\_pago TEXT NOT NULL DEFAULT 'Pendiente', -- 'Pendiente', 'Pagado'

            fecha\_creacion TEXT NOT NULL, -- Formato ISO: YYYY-MM-%d %H:%M:%S

            fecha\_pago TEXT, -- Formato ISO: YYYY-MM-%d %H:%M:%S (puede ser NULL)

            lat\_cliente REAL,

            lon\_cliente REAL,

            id\_repartidor INTEGER, -- Repartidor asignado a este pedido

            FOREIGN KEY(id\_repartidor) REFERENCES repartidores(id\_repartidor)

        )

    """)

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS items\_pedido (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            id\_pedido INTEGER NOT NULL,

            id\_plato INTEGER NOT NULL,

            cantidad INTEGER NOT NULL,

            precio\_unitario REAL NOT NULL, -- Precio del plato al momento de hacer el pedido

            FOREIGN KEY(id\_pedido) REFERENCES pedidos(id\_pedido),

            FOREIGN KEY(id\_plato) REFERENCES platos(id\_plato)

        )

    """)

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS ingresos\_egresos (

            id INTEGER PRIMARY KEY AUTOINCREMENT,

            tipo TEXT NOT NULL, -- 'Ingreso', 'Egreso', 'Pago a Repartidor'

            monto REAL NOT NULL,

            descripcion TEXT,

            fecha\_hora TEXT NOT NULL, -- Formato ISO: YYYY-MM-%d %H:%M:%S

            id\_pedido\_origen INTEGER, -- Para vincular a pedidos, NULL para egresos generales

            id\_repartidor\_origen INTEGER, -- Para vincular a repartidor en caso de pago

            FOREIGN KEY(id\_pedido\_origen) REFERENCES pedidos(id\_pedido),

            FOREIGN KEY(id\_repartidor\_origen) REFERENCES repartidores(id\_repartidor)

        )

    """)

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS configuracion (

            clave TEXT PRIMARY KEY,

            valor TEXT

        )

    """)

    # --- Nuevas tablas para la gestión de usuarios y empresas ---

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS empresas (

            id\_empresa INTEGER PRIMARY KEY AUTOINCREMENT,

            nombre TEXT NOT NULL UNIQUE,

            activo INTEGER DEFAULT 1 -- 1 para activo, 0 para inactivo

        )

    """)

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS roles (

            id\_rol INTEGER PRIMARY KEY AUTOINCREMENT,

            nombre\_rol TEXT NOT NULL UNIQUE -- Ej: 'super\_admin', 'admin\_empresa', 'empleado'

        )

    """)

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS usuarios (

            id\_usuario INTEGER PRIMARY KEY AUTOINCREMENT,

            email TEXT NOT NULL UNIQUE,

            password TEXT NOT NULL,

            nombre TEXT NOT NULL,

            apellido TEXT NOT NULL,

            id\_rol INTEGER NOT NULL,

            id\_empresa INTEGER, -- NULL para el super\_admin, o para usuarios no vinculados a una empresa específica

            activo INTEGER DEFAULT 1, -- 1 para activo, 0 para inactivo

            primer\_login\_requerido INTEGER DEFAULT 1, -- 1 si necesita cambiar la clave al primer login

            FOREIGN KEY(id\_rol) REFERENCES roles(id\_rol),

            FOREIGN KEY(id\_empresa) REFERENCES empresas(id\_empresa)

        )

    """)

    # --- FIN Nuevas tablas ---

    # Comprobar y añadir columna id\_repartidor a 'pedidos' si no existe

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

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

    if 'id\_repartidor' not in columns:

        cursor.execute("ALTER TABLE pedidos ADD COLUMN id\_repartidor INTEGER REFERENCES repartidores(id\_repartidor)")

        print("Columna 'id\_repartidor' añadida a la tabla 'pedidos'.")

    # Comprobar y añadir columna id\_repartidor\_origen a 'ingresos\_egresos' si no existe

    cursor.execute("PRAGMA table\_info(ingresos\_egresos)")

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

    if 'id\_repartidor\_origen' not in columns:

        cursor.execute("ALTER TABLE ingresos\_egresos ADD COLUMN id\_repartidor\_origen INTEGER REFERENCES repartidores(id\_repartidor)")

        print("Columna 'id\_repartidor\_origen' añadida a la tabla 'ingresos\_egresos'.")

    # Insertar roles iniciales si no existen (usamos INSERT OR IGNORE)

    cursor.execute("INSERT OR IGNORE INTO roles (id\_rol, nombre\_rol) VALUES (1, 'super\_admin')")

    cursor.execute("INSERT OR IGNORE INTO roles (id\_rol, nombre\_rol) VALUES (2, 'admin\_empresa')")

    cursor.execute("INSERT OR IGNORE INTO roles (id\_rol, nombre\_rol) VALUES (3, 'empleado')")

    conn.commit()

    conn.close()

@app.route('/gestion/pedidos')

@login\_required # Proteger esta ruta

def gestion\_pedidos():

    """Muestra una lista de todos los pedidos para gestión, filtrados por empresa."""

    if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa') or current\_user.has\_role('empleado')):

        flash("No tienes permiso para acceder a esta página.", "danger")

        return redirect(url\_for('index'))

    conn = conectar\_db()

    cursor = conn.cursor()

    query = """

        SELECT p.id\_pedido, p.cliente\_nombre, p.cliente\_apellido, p.direccion\_entrega, p.horario\_entrega,

               p.forma\_pago, p.costo\_total, p.estado\_pago, p.es\_envio,

               r.nombre AS repartidor\_nombre, r.apellido AS repartidor\_apellido,

               e.nombre AS nombre\_empresa

        FROM pedidos p

        LEFT JOIN repartidores r ON p.id\_repartidor = r.id\_repartidor

        LEFT JOIN empresas e ON p.id\_empresa = e.id\_empresa

    """

    params = []

    # <--- MODIFICADO: Aplicar filtro de empresa a los pedidos

    company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params(table\_alias='p')

    if company\_filter\_clause:

        query += f" WHERE {company\_filter\_clause}"

        params.extend(company\_filter\_params)

    query += " ORDER BY p.horario\_entrega DESC"

    cursor.execute(query, params)

    pedidos = cursor.fetchall()

    conn.close()

    pedidos\_procesados = []

    for p in pedidos:

        p\_dict = dict(p)

        p\_dict['horario\_entrega\_dt'] = datetime.strptime(p\_dict['horario\_entrega'], '%Y-%m-%d %H:%M:%S')

        pedidos\_procesados.append(p\_dict)

    # <--- MODIFICADO: Repartidores también filtrados por empresa

    conn = conectar\_db()

    cursor = conn.cursor()

    query\_repartidores = "SELECT id\_repartidor, nombre, apellido, activo FROM repartidores WHERE activo = 1"

    params\_repartidores = []

    company\_filter\_clause\_rep, company\_filter\_params\_rep = get\_company\_filter\_clause\_and\_params()

    if company\_filter\_clause\_rep:

        # Si ya hay un WHERE, se usa AND. En este caso no suele haberlo.

        query\_repartidores += f" WHERE {company\_filter\_clause\_rep}"

        params\_repartidores.extend(company\_filter\_params\_rep)

    query\_repartidores += " ORDER BY nombre, apellido"

    cursor.execute(query\_repartidores, params\_repartidores)

    repartidores = cursor.fetchall()

    conn.close()

    print(f"Repartidores cargados para gestión: {repartidores}")

    return render\_template('gestion\_pedidos.html',

                           pedidos=pedidos\_procesados,

                           repartidores=repartidores)

App.py

from flask import Flask, render\_template, request, redirect, url\_for, flash, session, jsonify

import sqlite3

from datetime import datetime, timedelta

import math

import requests

import json

import os

import sys # <--- AÑADIDO: Importar sys para acceder a módulos

# Importar Flask-Login y Werkzeug para autenticación

from flask\_login import LoginManager, UserMixin, login\_user, logout\_user, current\_user, login\_required

from werkzeug.security import generate\_password\_hash, check\_password\_hash

# Importar configuración

from config import (

GOOGLE\_MAPS\_API\_KEY, MAX\_PEDIDOS\_POR\_FRANJA\_HORARIA,

RADIO\_ENVIO\_CUADRAS, CUADRA\_METROS, DB\_NAME,

SUCURSAL\_LAT, SUCURSAL\_LON, HORA\_APERTURA, HORA\_CIERRE, INTERVALO\_FRANJAS\_MINUTOS,

DEFAULT\_COMPANY\_FOR\_ORDERS # <--- AÑADIDO: Para asignar pedidos de clientes

)

app = Flask(\_\_name\_\_)

app.secret\_key = 'super\_secreto\_de\_casa\_comida\_web\_202024' # CAMBIA ESTO POR UNA CLAVE MÁS SEGURA EN PRODUCCIÓN

# Inicializar Flask-Login

login\_manager = LoginManager()

login\_manager.init\_app(app)

login\_manager.login\_view = 'login' # Define la vista a la que redirigir si se requiere login

login\_manager.login\_message = "Por favor, inicie sesión para acceder a esta página."

login\_manager.login\_message\_category = "warning"

# --- Constante para el costo de envío por defecto si no está en DB ---

DEFAULT\_ENVIO\_COSTO = 500.00

# --- Costo por envío al repartidor (valor por defecto, también configurable en DB) ---

DEFAULT\_PAGO\_REPARTIDOR\_POR\_ENVIO = 300.00

# --- Funciones de Base de Datos ---

def conectar\_db():

conn = sqlite3.connect(DB\_NAME)

conn.row\_factory = sqlite3.Row # Permite acceder a las columnas por nombre

return conn

def crear\_tablas():

"""Crea las tablas de la base de datos si no existen y añade columnas si faltan."""

conn = sqlite3.connect(DB\_NAME)

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS platos (

id\_plato INTEGER PRIMARY KEY AUTOINCREMENT,

nombre TEXT NOT NULL,

descripcion TEXT,

precio REAL NOT NULL,

activo INTEGER DEFAULT 1, -- 1 para activo, 0 para inactivo (eliminación lógica)

id\_empresa INTEGER, -- <--- MODIFICADO: Añadida columna id\_empresa

FOREIGN KEY(id\_empresa) REFERENCES empresas(id\_empresa)

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS repartidores (

id\_repartidor INTEGER PRIMARY KEY AUTOINCREMENT,

nombre TEXT NOT NULL,

apellido TEXT NOT NULL,

telefono TEXT,

activo INTEGER DEFAULT 1, -- 1 para activo, 0 para inactivo

id\_empresa INTEGER, -- <--- MODIFICADO: Añadida columna id\_empresa

FOREIGN KEY(id\_empresa) REFERENCES empresas(id\_empresa)

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS pedidos (

id\_pedido INTEGER PRIMARY KEY AUTOINCREMENT,

cliente\_nombre TEXT NOT NULL,

cliente\_apellido TEXT NOT NULL,

direccion\_entrega TEXT NOT NULL,

es\_envio INTEGER NOT NULL, -- 0 para retiro, 1 para envío

horario\_entrega TEXT NOT NULL, -- Formato ISO: YYYY-MM-DD HH:MM:SS

costo\_envio REAL NOT NULL,

costo\_total REAL NOT NULL,

forma\_pago TEXT NOT NULL,

estado\_pago TEXT NOT NULL DEFAULT 'Pendiente', -- 'Pendiente', 'Pagado'

fecha\_creacion TEXT NOT NULL, -- Formato ISO: YYYY-MM-%d %H:%M:%S

fecha\_pago TEXT, -- Formato ISO: YYYY-MM-%d %H:%M:%S (puede ser NULL)

lat\_cliente REAL,

lon\_cliente REAL,

id\_repartidor INTEGER, -- Repartidor asignado a este pedido

id\_empresa INTEGER, -- <--- MODIFICADO: Añadida columna id\_empresa

FOREIGN KEY(id\_repartidor) REFERENCES repartidores(id\_repartidor),

FOREIGN KEY(id\_empresa) REFERENCES empresas(id\_empresa)

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS items\_pedido (

id INTEGER PRIMARY KEY AUTOINCREMENT,

id\_pedido INTEGER NOT NULL,

id\_plato INTEGER NOT NULL,

cantidad INTEGER NOT NULL,

precio\_unitario REAL NOT NULL, -- Precio del plato al momento de hacer el pedido

FOREIGN KEY(id\_pedido) REFERENCES pedidos(id\_pedido),

FOREIGN KEY(id\_plato) REFERENCES platos(id\_plato)

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS ingresos\_egresos (

id INTEGER PRIMARY KEY AUTOINCREMENT,

tipo TEXT NOT NULL, -- 'Ingreso', 'Egreso', 'Pago a Repartidor'

monto REAL NOT NULL,

descripcion TEXT,

fecha\_hora TEXT NOT NULL, -- Formato ISO: YYYY-MM-%d %H:%M:%S

id\_pedido\_origen INTEGER, -- Para vincular a pedidos, NULL para egresos generales

id\_repartidor\_origen INTEGER, -- Para vincular a repartidor en caso de pago

id\_empresa INTEGER, -- <--- MODIFICADO: Añadida columna id\_empresa

FOREIGN KEY(id\_pedido\_origen) REFERENCES pedidos(id\_pedido),

FOREIGN KEY(id\_repartidor\_origen) REFERENCES repartidores(id\_repartidor),

FOREIGN KEY(id\_empresa) REFERENCES empresas(id\_empresa)

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS configuracion (

clave TEXT PRIMARY KEY,

valor TEXT,

id\_empresa INTEGER, -- <--- MODIFICADO: Añadida columna id\_empresa (para config por empresa)

FOREIGN KEY(id\_empresa) REFERENCES empresas(id\_empresa)

)

""")

# --- Nuevas tablas para la gestión de usuarios y empresas (ya existían, no se modifican aquí) ---

cursor.execute("""

CREATE TABLE IF NOT EXISTS empresas (

id\_empresa INTEGER PRIMARY KEY AUTOINCREMENT,

nombre TEXT NOT NULL UNIQUE,

activo INTEGER DEFAULT 1 -- 1 para activo, 0 para inactivo

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS roles (

id\_rol INTEGER PRIMARY KEY AUTOINCREMENT,

nombre\_rol TEXT NOT NULL UNIQUE -- Ej: 'super\_admin', 'admin\_empresa', 'empleado'

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS usuarios (

id\_usuario INTEGER PRIMARY KEY AUTOINCREMENT,

email TEXT NOT NULL UNIQUE,

password TEXT NOT NULL,

nombre TEXT NOT NULL,

apellido TEXT NOT NULL,

id\_rol INTEGER NOT NULL,

id\_empresa INTEGER, -- NULL para el super\_admin, o para usuarios no vinculados a una empresa específica

activo INTEGER DEFAULT 1, -- 1 para activo, 0 para inactivo

primer\_login\_requerido INTEGER DEFAULT 1, -- 1 si necesita cambiar la clave al primer login

FOREIGN KEY(id\_rol) REFERENCES roles(id\_rol),

FOREIGN KEY(id\_empresa) REFERENCES empresas(id\_empresa)

)

""")

# --- FIN Nuevas tablas ---

# --- Comprobar y añadir columnas si faltan (para migraciones sin borrar DB) ---

# Columna id\_repartidor a 'pedidos' (ya la tenías)

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

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

if 'id\_repartidor' not in columns:

cursor.execute("ALTER TABLE pedidos ADD COLUMN id\_repartidor INTEGER REFERENCES repartidores(id\_repartidor)")

print("Columna 'id\_repartidor' añadida a la tabla 'pedidos'.")

if 'id\_empresa' not in columns: # <--- MODIFICADO: Nueva comprobación para pedidos

cursor.execute("ALTER TABLE pedidos ADD COLUMN id\_empresa INTEGER REFERENCES empresas(id\_empresa)")

print("Columna 'id\_empresa' añadida a la tabla 'pedidos'.")

# Columna id\_repartidor\_origen a 'ingresos\_egresos' (ya la tenías)

cursor.execute("PRAGMA table\_info(ingresos\_egresos)")

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

if 'id\_repartidor\_origen' not in columns:

cursor.execute("ALTER TABLE ingresos\_egresos ADD COLUMN id\_repartidor\_origen INTEGER REFERENCES repartidores(id\_repartidor)")

print("Columna 'id\_repartidor\_origen' añadida a la tabla 'ingresos\_egresos'.")

if 'id\_empresa' not in columns: # <--- MODIFICADO: Nueva comprobación para ingresos\_egresos

cursor.execute("ALTER TABLE ingresos\_egresos ADD COLUMN id\_empresa INTEGER REFERENCES empresas(id\_empresa)")

print("Columna 'id\_empresa' añadida a la tabla 'ingresos\_egresos'.")

# Columna id\_empresa a 'platos'

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

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

if 'id\_empresa' not in columns:

cursor.execute("ALTER TABLE platos ADD COLUMN id\_empresa INTEGER REFERENCES empresas(id\_empresa)")

print("Columna 'id\_empresa' añadida a la tabla 'platos'.")

# Columna id\_empresa a 'repartidores'

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

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

if 'id\_empresa' not in columns:

cursor.execute("ALTER TABLE repartidores ADD COLUMN id\_empresa INTEGER REFERENCES empresas(id\_empresa)")

print("Columna 'id\_empresa' añadida a la tabla 'repartidores'.")

# Columna id\_empresa a 'configuracion'

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

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

if 'id\_empresa' not in columns:

cursor.execute("ALTER TABLE configuracion ADD COLUMN id\_empresa INTEGER REFERENCES empresas(id\_empresa)")

print("Columna 'id\_empresa' añadida a la tabla 'configuracion'.")

# Insertar roles iniciales si no existen (usamos INSERT OR IGNORE)

cursor.execute("INSERT OR IGNORE INTO roles (id\_rol, nombre\_rol) VALUES (1, 'super\_admin')")

cursor.execute("INSERT OR IGNORE INTO roles (id\_rol, nombre\_rol) VALUES (2, 'admin\_empresa')")

cursor.execute("INSERT OR IGNORE INTO roles (id\_rol, nombre\_rol) VALUES (3, 'empleado')")

conn.commit()

conn.close()

# <--- MODIFICADO: Funciones de configuración para soportar id\_empresa

def guardar\_configuracion(clave, valor, id\_empresa=None):

"""Guarda un par clave-valor en la tabla de configuración, opcionalmente por empresa."""

conn = conectar\_db()

cursor = conn.cursor()

if id\_empresa:

# Si se especifica id\_empresa, se guarda para esa empresa

cursor.execute("REPLACE INTO configuracion (clave, valor, id\_empresa) VALUES (?, ?, ?)", (clave, str(valor), id\_empresa))

else:

# Si no se da id\_empresa, se guarda como configuración global (id\_empresa IS NULL)

cursor.execute("REPLACE INTO configuracion (clave, valor, id\_empresa) VALUES (?, ?, NULL)", (clave, str(valor)))

conn.commit()

conn.close()

def cargar\_configuracion(clave, valor\_defecto=None, id\_empresa=None):

"""Carga un valor de la tabla de configuración por su clave, opcionalmente por empresa."""

conn = conectar\_db()

cursor = conn.cursor()

if id\_empresa:

# Busca la configuración específica para la empresa

cursor.execute("SELECT valor FROM configuracion WHERE clave = ? AND id\_empresa = ?", (clave, id\_empresa))

else:

# Busca la configuración global (sin empresa asociada)

cursor.execute("SELECT valor FROM configuracion WHERE clave = ? AND id\_empresa IS NULL", (clave,))

resultado = cursor.fetchone()

conn.close()

if resultado:

return resultado['valor']

return valor\_defecto

def get\_costo\_envio():

"""Obtiene el costo de envío desde la base de datos o usa un valor por defecto."""

# Prioriza la configuración de la empresa del usuario actual

if current\_user.is\_authenticated and current\_user.id\_empresa:

costo = cargar\_configuracion('ENVIO\_COSTO', str(DEFAULT\_ENVIO\_COSTO), current\_user.id\_empresa)

else:

# Si no hay usuario logueado o es super\_admin (sin id\_empresa), usa la configuración global

costo = cargar\_configuracion('ENVIO\_COSTO', str(DEFAULT\_ENVIO\_COSTO), None)

try:

return float(costo)

except ValueError:

return DEFAULT\_ENVIO\_COSTO

def get\_pago\_repartidor\_por\_envio():

"""Obtiene el pago por envío al repartidor desde la base de datos o usa un valor por defecto."""

# Prioriza la configuración de la empresa del usuario actual

if current\_user.is\_authenticated and current\_user.id\_empresa:

pago = cargar\_configuracion('PAGO\_REPARTIDOR\_POR\_ENVIO', str(DEFAULT\_PAGO\_REPARTIDOR\_POR\_ENVIO), current\_user.id\_empresa)

else:

# Si no hay usuario logueado o es super\_admin (sin id\_empresa), usa la configuración global

pago = cargar\_configuracion('PAGO\_REPARTIDOR\_POR\_ENVIO', str(DEFAULT\_PAGO\_REPARTIDOR\_POR\_ENVIO), None)

try:

return float(pago)

except ValueError:

return DEFAULT\_PAGO\_REPARTIDOR\_POR\_ENVIO

# <--- MODIFICADO: Agregada lógica para crear empresa por defecto y admin\_empresa

def \_agregar\_super\_admin\_inicial():

"""Agrega un usuario super\_admin inicial si no existe ninguno,

y una empresa por defecto con un admin\_empresa si no existen."""

conn = conectar\_db()

cursor = conn.cursor()

# Verificar si ya existe un super\_admin

cursor.execute("SELECT COUNT(\*) FROM usuarios WHERE id\_rol = (SELECT id\_rol FROM roles WHERE nombre\_rol = 'super\_admin')")

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

hashed\_password = generate\_password\_hash("admin\_password\_inicial\_segura", method='pbkdf2:sha256')

cursor.execute("""

INSERT INTO usuarios (email, password, nombre, apellido, id\_rol, id\_empresa, activo, primer\_login\_requerido)

VALUES (?, ?, ?, ?, (SELECT id\_rol FROM roles WHERE nombre\_rol = 'super\_admin'), NULL, 1, 0)

""", ("admin@tudominio.com", hashed\_password, "Super", "Admin",))

conn.commit()

print("Usuario 'super\_admin' inicial creado: admin@tudominio.com con la contraseña 'admin\_password\_inicial\_segura' (¡CÁMBIALA!)")

# Asegurarse de que exista una empresa por defecto (por ejemplo, con ID 1)

# Usamos DEFAULT\_COMPANY\_FOR\_ORDERS de config.py

default\_company\_id = DEFAULT\_COMPANY\_FOR\_ORDERS

cursor.execute("INSERT OR IGNORE INTO empresas (id\_empresa, nombre, activo) VALUES (?, ?, 1)",

(default\_company\_id, "Empresa Principal por Defecto"))

conn.commit() # Commit para asegurar que la empresa exista antes de referenciarla

print(f"Empresa por defecto (ID: {default\_company\_id}, Nombre: Empresa Principal por Defecto) asegurada.")

# Añadir un admin\_empresa para la empresa por defecto si no existe

cursor.execute("SELECT COUNT(\*) FROM usuarios WHERE id\_rol = (SELECT id\_rol FROM roles WHERE nombre\_rol = 'admin\_empresa') AND id\_empresa = ?", (default\_company\_id,))

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

hashed\_password = generate\_password\_hash("empresa\_password\_segura", method='pbkdf2:sha256')

cursor.execute("""

INSERT INTO usuarios (email, password, nombre, apellido, id\_rol, id\_empresa, activo, primer\_login\_requerido)

VALUES (?, ?, ?, ?, (SELECT id\_rol FROM roles WHERE nombre\_rol = 'admin\_empresa'), ?, 1, 1)

""", ("admin\_empresa@empresa.com", hashed\_password, "Admin", "Empresa", default\_company\_id))

conn.commit()

print(f"Usuario 'admin\_empresa' inicial creado para Empresa Principal (ID: {default\_company\_id}): admin\_empresa@empresa.com con la contraseña 'empresa\_password\_segura' (¡CÁMBIALA!)")

conn.close()

# <--- MODIFICADO: Platos de ejemplo asignados a la empresa por defecto

def \_agregar\_platos\_ejemplo\_a\_db():

"""Agrega platos de ejemplo a la DB si no existen, asignándolos a la empresa por defecto."""

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("SELECT COUNT(\*) FROM platos")

if cursor.fetchone()[0] == 0: # Si no hay platos, agregar ejemplos

# Los platos de ejemplo se asignan a la empresa por defecto

default\_company\_id = DEFAULT\_COMPANY\_FOR\_ORDERS

platos\_ejemplo = [

("Hamburguesa Clásica", "Carne 180gr, lechuga, tomate, queso cheddar, cebolla caramelizada, pepinillos.", 3500.00),

("Pizza Muzzarella Grande", "Salsa de tomate casera, abundante muzzarella, aceitunas verdes, orégano.", 4200.00),

("Milanesa Napolitana (con fritas)", "Tierna milanesa de ternera, salsa, jamón, queso gratinado, acompañada de papas fritas.", 5800.00),

("Ensalada César con Pollo", "Lechuga romana, crutones, queso parmesano, aderezo César y tiras de pollo grillado.", 3000.00),

("Empanadas de Carne (unidad)", "Carne picada a cuchillo, huevo, aceitunas. Jugosas y sabrosas.", 600.00),

("Gaseosa Coca-Cola", "Lata de 354ml, bien fría.", 800.00),

("Agua Mineral sin Gas", "Botella de 500ml.", 700.00),

("Flan Casero con Dulce de Leche y Crema", "Receta de la abuela, irresistible.", 2500.00),

("Cerveza Artesanal IPA", "Pinta de cerveza artesanal de lupulado intenso.", 1500.00),

]

for nombre, desc, precio in platos\_ejemplo:

cursor.execute("INSERT INTO platos (nombre, descripcion, precio, activo, id\_empresa) VALUES (?, ?, ?, 1, ?)",

(nombre, desc, precio, default\_company\_id))

conn.commit()

print(f"Platos de ejemplo agregados a la base de datos para la empresa ID {default\_company\_id}.")

conn.close()

# <--- MODIFICADO: Repartidores de ejemplo asignados a la empresa por defecto

def \_agregar\_repartidor\_ejemplo\_a\_db():

"""Agrega un repartidor de ejemplo a la DB si no existen repartidores, asignándolos a la empresa por defecto."""

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("SELECT COUNT(\*) FROM repartidores")

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

default\_company\_id = DEFAULT\_COMPANY\_FOR\_ORDERS

cursor.execute("INSERT INTO repartidores (nombre, apellido, telefono, activo, id\_empresa) VALUES (?, ?, ?, 1, ?)",

("Juan", "Perez", "1123456789", default\_company\_id))

cursor.execute("INSERT INTO repartidores (nombre, apellido, telefono, activo, id\_empresa) VALUES (?, ?, ?, 1, ?)",

("Maria", "Gomez", "1198765432", default\_company\_id))

conn.commit()

print(f"Repartidores de ejemplo agregados a la base de datos para la empresa ID {default\_company\_id}.")

conn.close()

# --- Clases de Modelo (no requieren cambios por id\_empresa aquí, ya que se maneja a nivel de consulta) ---

class Plato:

"""Representa un plato o ítem del menú."""

def \_\_init\_\_(self, id\_plato, nombre, descripcion, precio, activo=1, id\_empresa=None): # id\_empresa añadido al init

self.id\_plato = id\_plato

self.nombre = nombre

self.descripcion = descripcion

self.precio = precio

self.activo = activo

self.id\_empresa = id\_empresa

class Repartidor:

"""Representa un repartidor."""

def \_\_init\_\_(self, id\_repartidor, nombre, apellido, telefono, activo=1, id\_empresa=None): # id\_empresa añadido al init

self.id\_repartidor = id\_repartidor

self.nombre = nombre

self.apellido = apellido

self.telefono = telefono

self.activo = activo

self.id\_empresa = id\_empresa

@property

def nombre\_completo(self):

return f"{self.nombre} {self.apellido}"

class Pedido:

"""Representa un pedido realizado por un cliente, incluyendo su detalle de ítems."""

def \_\_init\_\_(self, id\_pedido, cliente\_nombre, cliente\_apellido, direccion\_entrega, es\_envio,

horario\_entrega, costo\_envio, costo\_total, forma\_pago, estado\_pago, fecha\_creacion,

lat\_cliente, lon\_cliente, fecha\_pago=None, id\_repartidor=None, id\_empresa=None): # id\_empresa añadido al init

self.id\_pedido = id\_pedido

self.cliente\_nombre = cliente\_nombre

self.cliente\_apellido = cliente\_apellido

self.direccion\_entrega = direccion\_entrega

self.es\_envio = bool(es\_envio)

self.horario\_entrega = datetime.strptime(horario\_entrega, '%Y-%m-%d %H:%M:%S') if isinstance(horario\_entrega, str) else horario\_entrega

self.costo\_envio = costo\_envio

self.costo\_total = costo\_total

self.forma\_pago = forma\_pago

self.estado\_pago = estado\_pago

self.fecha\_creacion = datetime.strptime(fecha\_creacion, '%Y-%m-%d %H:%M:%S') if isinstance(fecha\_creacion, str) else fecha\_creacion

self.fecha\_pago = datetime.strptime(fecha\_pago, '%Y-%m-%d %H:%M:%S') if fecha\_pago else None

self.lat\_cliente = lat\_cliente

self.lon\_cliente = lon\_cliente

self.id\_repartidor = id\_repartidor

self.id\_empresa = id\_empresa # <--- id\_empresa añadido

self.repartidor = None # Se cargará lazy si es necesario (ej: en \_obtener\_pedido\_completo\_por\_id)

self.items = [] # Almacenará diccionarios de la forma {"plato": Plato\_objeto, "cantidad": int, "precio\_unitario": float}

def agregar\_item(self, plato, cantidad, precio\_unitario):

"""Agrega un plato (o la información relevante del plato) al detalle del pedido."""

self.items.append({"plato": plato, "cantidad": cantidad, "precio\_unitario": precio\_unitario})

def generar\_ticket(self):

"""Genera un string HTML con el detalle del pedido, formateado para impresión."""

ticket\_html = f"""

<div class="ticket">

<h4 class="text-center">TICKET DE PEDIDO #{self.id\_pedido}</h4>

<hr>

<p><strong>Cliente:</strong> {self.cliente\_nombre} {self.cliente\_apellido}</p>

<p><strong>Dirección:</strong> {self.direccion\_entrega}</p>

<p><strong>Tipo:</strong> {'Envío' if self.es\_envio else 'Retiro en Sucursal'}</p>

<p><strong>Horario:</strong> {self.horario\_entrega.strftime('%H:%M')} ({self.horario\_entrega.strftime('%d/%m')})</p>

"""

if self.es\_envio and self.repartidor:

ticket\_html += f"<p><strong>Repartidor:</strong> {self.repartidor.nombre\_completo}</p>"

ticket\_html += """<hr>

<h6>Detalle del Pedido:</h6>

<ul class="list-unstyled">

"""

for item in self.items:

plato\_nombre = item["plato"].nombre if isinstance(item["plato"], Plato) else item["plato"]

cantidad = item["cantidad"]

precio\_unitario = item["precio\_unitario"]

ticket\_html += f"<li>{cantidad} x {plato\_nombre} @ ${precio\_unitario:,.2f} = ${cantidad \* precio\_unitario:,.2f}</li>"

ticket\_html += "</ul>"

if self.es\_envio:

ticket\_html += f"<p><strong>Costo de Envío:</strong> ${self.costo\_envio:,.2f}</p>"

ticket\_html += f"""

<hr>

<p><strong>TOTAL: ${self.costo\_total:,.2f}</strong></p>

<hr>

<p><strong>Forma de Pago:</strong> {self.forma\_pago}</p>

<p><strong>Estado del Pago:</strong> {self.estado\_pago}</p>

"""

if self.fecha\_pago:

ticket\_html += f"<p><strong>Fecha de Pago:</strong> {self.fecha\_pago.strftime('%d/%m/%Y %H:%M')}</p>"

ticket\_html += f"""

<hr>

<p class="text-center">¡Gracias por su compra!</p>

</div>

"""

return ticket\_html

class Empresa:

def \_\_init\_\_(self, id\_empresa, nombre, activo=1):

self.id\_empresa = id\_empresa

self.nombre = nombre

self.activo = activo

class Rol:

def \_\_init\_\_(self, id\_rol, nombre\_rol):

self.id\_rol = id\_rol

self.nombre\_rol = nombre\_rol

class Usuario(UserMixin):

def \_\_init\_\_(self, id\_usuario, email, password, nombre, apellido, id\_rol, id\_empresa, activo=1, primer\_login\_requerido=1, nombre\_rol=None):

self.id = id\_usuario # Flask-Login espera que el ID se llame 'id'

self.email = email

self.password = password # Esta será la contraseña hasheada

self.nombre = nombre

self.apellido = apellido

self.id\_rol = id\_rol

self.id\_empresa = id\_empresa

self.activo = activo

self.primer\_login\_requerido = primer\_login\_requerido

self.nombre\_rol = nombre\_rol # Para un acceso más fácil al nombre del rol

def get\_id(self):

return str(self.id) # Flask-Login requiere que este método retorne un string

def is\_active(self):

return bool(self.activo)

def get\_full\_name(self):

return f"{self.nombre} {self.apellido}"

def has\_role(self, role\_name):

return self.nombre\_rol == role\_name

@login\_manager.user\_loader

def load\_user(user\_id):

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("""

SELECT u.id\_usuario, u.email, u.password, u.nombre, u.apellido,

u.id\_rol, u.id\_empresa, u.activo, u.primer\_login\_requerido,

r.nombre\_rol

FROM usuarios u

JOIN roles r ON u.id\_rol = r.id\_rol

WHERE u.id\_usuario = ?

""", (user\_id,))

user\_data = cursor.fetchone()

conn.close()

if user\_data:

return Usuario(

user\_data['id\_usuario'],

user\_data['email'],

user\_data['password'],

user\_data['nombre'],

user\_data['apellido'],

user\_data['id\_rol'],

user\_data['id\_empresa'],

user\_data['activo'],

user\_data['primer\_login\_requerido'],

user\_data['nombre\_rol']

)

return None

# --- Funciones de Google Maps y Geocodificación ---

\_info\_restaurante = None

def obtener\_info\_restaurante\_google\_maps\_cached(nombre\_restaurante):

"""

Busca información de un restaurante en Google Maps y la guarda en caché.

Actualiza las coordenadas globales SUCURSAL\_LAT/LON si obtiene datos.

"""

global \_info\_restaurante, SUCURSAL\_LAT, SUCURSAL\_LON

if \_info\_restaurante:

return \_info\_restaurante

if not GOOGLE\_MAPS\_API\_KEY or GOOGLE\_MAPS\_API\_KEY == "YOUR\_GOOGLE\_MAPS\_API\_KEY":

print("Advertencia: API Key de Google Maps no configurada. Usando datos de ejemplo para el restaurante.")

\_info\_restaurante = {

"nombre": nombre\_restaurante,

"direccion": "Dirección de ejemplo, 1234, Ciudad Ficticia",

"lat": SUCURSAL\_LAT,

"lon": SUCURSAL\_LON,

"horario\_atencion": ["Lunes a Viernes: 09:00 - 23:00", "Sábado y Domingo: 10:00 - 00:00"],

"url\_mapa": "https://maps.google.com/?q=Casa+de+Comida+Ejemplo"

}

return \_info\_restaurante

search\_url = "https://maps.googleapis.com/maps/api/place/findplacefromtext/json"

params\_search = { "input": nombre\_restaurante, "inputtype": "textquery", "fields": "place\_id", "key": GOOGLE\_MAPS\_API\_KEY, "language": "es" }

try:

response\_search = requests.get(search\_url, params=params\_search, timeout=5)

response\_search.raise\_for\_status()

data\_search = response\_search.json()

if data\_search["status"] == "OK" and data\_search["candidates"]:

place\_id = data\_search["candidates"][0]["place\_id"]

else:

print(f"Error al buscar Place ID para '{nombre\_restaurante}'. Status: {data\_search.get('status')}. Error: {data\_search.get('error\_message')}")

return None

except requests.exceptions.RequestException as e:

print(f"Error de red con Google Places (Search): {e}")

return None

except json.JSONDecodeError:

print("Error al procesar respuesta JSON de Google Places (Search).")

return None

details\_url = "https://maps.googleapis.com/maps/api/place/details/json"

params\_details = { "place\_id": place\_id, "fields": "name,formatted\_address,geometry,opening\_hours,url", "key": GOOGLE\_MAPS\_API\_KEY, "language": "es" }

try:

response\_details = requests.get(details\_url, params=params\_details, timeout=5)

response\_details.raise\_for\_status()

data\_details = response\_details.json()

if data\_details["status"] == "OK" and data\_details["result"]:

result = data\_details["result"]

\_info\_restaurante = {

"nombre": result.get("name", nombre\_restaurante),

"direccion": result.get("formatted\_address", "Dirección no disponible"),

"lat": result["geometry"]["location"]["lat"],

"lon": result["geometry"]["location"]["lng"],

"horario\_atencion": result.get("opening\_hours", {}).get("weekday\_text", ["Horario no disponible"]),

"url\_mapa": result.get("url", f"https://maps.google.com/?q={nombre\_restaurante.replace(' ', '+')}")

}

SUCURSAL\_LAT = \_info\_restaurante['lat']

SUCURSAL\_LON = \_info\_restaurante['lon']

return \_info\_restaurante

else:

print(f"Error al obtener detalles del restaurante: {data\_details.get('status')}. Error: {data\_details.get('error\_message')}")

return None

except requests.exceptions.RequestException as e:

print(f"Error de red con Google Places (Details): {e}")

return None

except json.JSONDecodeError:

print("Error al procesar respuesta JSON de Google Places (Details).")

return None

def obtener\_coordenadas\_desde\_direccion(direccion):

"""

Convierte una dirección en latitud y longitud usando la API de Geocoding.

Si la API Key no es válida o hay un error, retorna coordenadas de ejemplo.

"""

if not GOOGLE\_MAPS\_API\_KEY or GOOGLE\_MAPS\_API\_KEY == "YOUR\_GOOGLE\_MAPS\_API\_KEY":

print("Advertencia: API Key de Google Maps no configurada. Usando coordenadas de ejemplo para la dirección.")

if "calle falsa 123" in direccion.lower(): return -34.6000, -58.4000

elif "avenida siempreviva 742" in direccion.lower(): return -34.6050, -58.3850

else: return -34.6100, -58.3900

geocoding\_url = "https://maps.googleapis.com/maps/api/geocode/json"

params = { "address": direccion, "key": GOOGLE\_MAPS\_API\_KEY, "language": "es" }

try:

response = requests.get(geocoding\_url, params=params, timeout=5)

response.raise\_for\_status()

data = response.json()

if data["status"] == "OK" and data["results"]:

location = data["results"][0]["geometry"]["location"]

return location["lat"], location["lng"]

else:

print(f"No se pudieron obtener coordenadas para la dirección: {data.get('status')}. Error: {data.get('error\_message')}")

return None

except requests.exceptions.RequestException as e:

print(f"Error de red con Google Geocoding: {e}")

return None

except json.JSONDecodeError:

print("Error al procesar respuesta JSON de Google Geocoding.")

return None

def calcular\_distancia\_cuadras(lat1, lon1, lat2, lon2):

"""

Calcula la distancia Haversine (línea recta) entre dos puntos geográficos

y la convierte aproximadamente a "cuadras".

"""

R = 6371.0

lat1\_rad, lon1\_rad, lat2\_rad, lon2\_rad = map(math.radians, [lat1, lon1, lat2, lon2])

dlon = lon2\_rad - lon1\_rad

dlat = lat2\_rad - lat1\_rad

a = math.sin(dlat / 2)\*\*2 + math.cos(lat1\_rad) \* math.cos(lat2\_rad) \* math.sin(dlon / 2)\*\*2

c = 2 \* math.atan2(math.sqrt(a), math.sqrt(1 - a))

distancia\_km = R \* c

distancia\_metros = distancia\_km \* 1000

distancia\_cuadras = distancia\_metros / CUADRA\_METROS

return distancia\_cuadras

# --- Lógica de Negocio y Utilidades para la App Web ---

# <--- AÑADIDO: Helper para el filtrado por empresa en las consultas

def get\_company\_filter\_clause\_and\_params(table\_alias=''):

"""

Genera la cláusula WHERE y los parámetros para filtrar por empresa para el usuario actual.

Si el usuario es super\_admin, no aplica filtro de empresa.

Retorna (where\_clause\_fragment, params\_list).

"""

where\_clause\_fragment = ""

params = []

# Si el usuario está autenticado y NO es super\_admin, aplica el filtro de empresa

if current\_user.is\_authenticated and not current\_user.has\_role('super\_admin'):

if current\_user.id\_empresa:

if table\_alias:

where\_clause\_fragment = f" {table\_alias}.id\_empresa = ?"

else:

where\_clause\_fragment = " id\_empresa = ?"

params.append(current\_user.id\_empresa)

else:

# Si un admin\_empresa o empleado no tiene id\_empresa asignada, no debería ver nada.

# Esto es un caso de error o configuración incorrecta, se niega el acceso a datos.

where\_clause\_fragment = " 1 = 0" # Cláusula que siempre es falsa

return where\_clause\_fragment, params

def \_generar\_franjas\_horarias\_disponibles():

"""

Genera una lista de franjas horarias futuras disponibles (no completas).

Cada franja es un string 'HH:MM'.

"""

franjas = []

hoy = datetime.now().date()

try:

inicio\_hora, inicio\_min = map(int, HORA\_APERTURA.split(':'))

fin\_hora, fin\_min = map(int, HORA\_CIERRE.split(':'))

except ValueError:

print("Error en formato de HORA\_APERTURA o HORA\_CIERRE en config.py")

return []

inicio\_turno\_hoy = datetime.combine(hoy, datetime.min.time()).replace(hour=inicio\_hora, minute=inicio\_min)

fin\_turno\_hoy = datetime.combine(hoy, datetime.min.time()).replace(hour=fin\_hora, minute=fin\_min)

hora\_actual\_dt = datetime.now()

current\_time = hora\_actual\_dt + timedelta(minutes=(INTERVALO\_FRANJAS\_MINUTOS - hora\_actual\_dt.minute % INTERVALO\_FRANJAS\_MINUTOS) % INTERVALO\_FRANJAS\_MINUTOS)

current\_time = current\_time.replace(second=0, microsecond=0)

if current\_time < inicio\_turno\_hoy:

current\_time = inicio\_turno\_hoy

franjas\_ocupadas = \_cargar\_franjas\_ocupadas\_desde\_db\_interna()

franjas\_para\_mostrar = []

while current\_time <= fin\_turno\_hoy:

if current\_time > hora\_actual\_dt:

pedidos\_en\_franja = franjas\_ocupadas.get(current\_time, 0)

if pedidos\_en\_franja < MAX\_PEDIDOS\_POR\_FRANJA\_HORARIA:

franjas\_para\_mostrar.append(current\_time.strftime('%H:%M'))

current\_time += timedelta(minutes=INTERVALO\_FRANJAS\_MINUTOS)

return franjas\_para\_mostrar

# <--- MODIFICADO: \_cargar\_franjas\_ocupadas\_desde\_db\_interna ahora filtra por DEFAULT\_COMPANY\_FOR\_ORDERS

def \_cargar\_franjas\_ocupadas\_desde\_db\_interna():

"""

Carga el conteo de pedidos por franja horaria para pedidos 'Pendiente'

desde la base de datos, considerando solo franjas futuras para el día actual y

filtrando por la empresa por defecto (para pedidos de clientes).

Retorna un diccionario {datetime\_obj: count}

"""

franjas\_ocupadas = {}

conn = conectar\_db()

cursor = conn.cursor()

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

# Para pedidos del cliente (frontend), siempre se usa la empresa por defecto

cursor.execute("""

SELECT horario\_entrega, COUNT(\*) AS num\_pedidos

FROM pedidos

WHERE estado\_pago = 'Pendiente'

AND horario\_entrega >= ?

AND id\_empresa = ? -- <--- MODIFICADO: Filtrar por empresa por defecto

GROUP BY horario\_entrega

""", (hoy\_str, DEFAULT\_COMPANY\_FOR\_ORDERS))

resultados = cursor.fetchall()

for row in resultados:

dt\_obj = datetime.strptime(row['horario\_entrega'], '%Y-%m-%d %H:%M:%S')

franjas\_ocupadas[dt\_obj] = row['num\_pedidos']

conn.close()

return franjas\_ocupadas

# <--- MODIFICADO: \_obtener\_pedido\_completo\_por\_id ahora filtra por la empresa del usuario logueado

def \_obtener\_pedido\_completo\_por\_id(id\_pedido):

"""

Recupera un objeto Pedido completo (con sus ítems y datos de repartidor si aplica)

de la base de datos, aplicando filtro de empresa para usuarios no super\_admin.

"""

conn = conectar\_db()

cursor = conn.cursor()

query = """

SELECT p.\*, r.nombre AS repartidor\_nombre, r.apellido AS repartidor\_apellido, r.telefono AS repartidor\_telefono

FROM pedidos p

LEFT JOIN repartidores r ON p.id\_repartidor = r.id\_repartidor

"""

params = [id\_pedido]

# Aplica el filtro de empresa solo si el usuario no es super\_admin

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params(table\_alias='p')

if company\_filter\_clause:

query += f" WHERE p.id\_pedido = ? AND {company\_filter\_clause}"

params.extend(company\_filter\_params)

else:

query += " WHERE p.id\_pedido = ?"

cursor.execute(query, params)

pedido\_data = cursor.fetchone()

if not pedido\_data:

conn.close()

return None

pedido = Pedido(

pedido\_data['id\_pedido'], pedido\_data['cliente\_nombre'], pedido\_data['cliente\_apellido'],

pedido\_data['direccion\_entrega'], pedido\_data['es\_envio'], pedido\_data['horario\_entrega'],

pedido\_data['costo\_envio'], pedido\_data['costo\_total'], pedido\_data['forma\_pago'],

pedido\_data['estado\_pago'], pedido\_data['fecha\_creacion'],

pedido\_data['lat\_cliente'], pedido\_data['lon\_cliente'],

pedido\_data['fecha\_pago'], pedido\_data['id\_repartidor'], pedido\_data['id\_empresa'] # <--- id\_empresa añadido

)

if pedido\_data['id\_repartidor']:

pedido.repartidor = Repartidor(

pedido\_data['id\_repartidor'],

pedido\_data['repartidor\_nombre'],

pedido\_data['repartidor\_apellido'],

pedido\_data['repartidor\_telefono']

)

cursor.execute("""

SELECT ip.cantidad, ip.precio\_unitario, p.id\_plato, p.nombre, p.descripcion

FROM items\_pedido ip

JOIN platos p ON ip.id\_plato = p.id\_plato

WHERE ip.id\_pedido = ?

""", (id\_pedido,))

items\_data = cursor.fetchall()

for item\_row in items\_data:

# Asegúrate de que el objeto Plato refleje la empresa si es relevante para futuras extensiones

plato = Plato(item\_row['id\_plato'], item\_row['nombre'], item\_row['descripcion'], item\_row['precio\_unitario'], id\_empresa=pedido.id\_empresa)

pedido.agregar\_item(plato, item\_row['cantidad'], item\_row['precio\_unitario'])

conn.close()

return pedido

# <--- MODIFICADO: init\_app para asegurar config globales

def init\_app():

"""

Función de inicialización que se ejecuta al inicio de la aplicación.

Crea tablas si no existen, añade platos de ejemplo y carga la información del restaurante.

"""

print("Inicializando la aplicación...")

crear\_tablas()

# Asegúrate de que la empresa por defecto y el super\_admin existan antes de añadir platos/repartidores

\_agregar\_super\_admin\_inicial()

\_agregar\_platos\_ejemplo\_a\_db() # Estos se asignarán a la empresa por defecto

\_agregar\_repartidor\_ejemplo\_a\_db() # Estos se asignarán a la empresa por defecto

obtener\_info\_restaurante\_google\_maps\_cached("La Esquina del Sabor")

# Guardar configuración por defecto como GLOBAL (id\_empresa=None) si no existen

if cargar\_configuracion('ENVIO\_COSTO', id\_empresa=None) is None:

guardar\_configuracion('ENVIO\_COSTO', DEFAULT\_ENVIO\_COSTO, id\_empresa=None)

print(f"Costo de envío inicial '{DEFAULT\_ENVIO\_COSTO}' guardado como configuración global.")

if cargar\_configuracion('PAGO\_REPARTIDOR\_POR\_ENVIO', id\_empresa=None) is None:

guardar\_configuracion('PAGO\_REPARTIDOR\_POR\_ENVIO', DEFAULT\_PAGO\_REPARTIDOR\_POR\_ENVIO, id\_empresa=None)

print(f"Costo de pago a repartidor inicial '{DEFAULT\_PAGO\_REPARTIDOR\_POR\_ENVIO}' guardado como configuración global.")

print("Aplicación inicializada.")

# --- Rutas de la Aplicación (Views) ---

@app.route('/')

def index():

"""Página principal, muestra información del restaurante."""

info = obtener\_info\_restaurante\_google\_maps\_cached("La Esquina del Sabor")

return render\_template('index.html', info=info)

# --- Rutas de Autenticación ---

@app.route('/login', methods=['GET', 'POST'])

def login():

if current\_user.is\_authenticated:

return redirect(url\_for('index')) # Si ya está logueado, redirige a la página principal

if request.method == 'POST':

email = request.form['email'].strip()

password = request.form['password'].strip()

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("""

SELECT u.id\_usuario, u.email, u.password, u.nombre, u.apellido,

u.id\_rol, u.id\_empresa, u.activo, u.primer\_login\_requerido,

r.nombre\_rol

FROM usuarios u

JOIN roles r ON u.id\_rol = r.id\_rol

WHERE u.email = ?

""", (email,))

user\_data = cursor.fetchone()

conn.close()

if user\_data:

user = Usuario(

user\_data['id\_usuario'],

user\_data['email'],

user\_data['password'],

user\_data['nombre'],

user\_data['apellido'],

user\_data['id\_rol'],

user\_data['id\_empresa'],

user\_data['activo'],

user\_data['primer\_login\_requerido'],

user\_data['nombre\_rol']

)

# Verificar contraseña hasheada

if check\_password\_hash(user.password, password) and user.is\_active():

login\_user(user) # Inicia la sesión con Flask-Login

flash(f"Bienvenido, {user.nombre}!", "success")

# Redirigir si es el primer login

if user.primer\_login\_requerido:

return redirect(url\_for('cambiar\_clave\_inicial'))

# Redirigir al usuario a la página a la que intentaba acceder, o a index

next\_page = request.args.get('next')

return redirect(next\_page or url\_for('index'))

else:

flash("Credenciales inválidas o usuario inactivo.", "danger")

else:

flash("Credenciales inválidas o usuario no encontrado.", "danger")

return render\_template('login.html') # Necesitarás crear esta plantilla

@app.route('/logout')

@login\_required # Requiere que el usuario esté logueado para cerrar sesión

def logout():

logout\_user() # Cierra la sesión con Flask-Login

flash("Has cerrado sesión.", "info")

return redirect(url\_for('index'))

@app.route('/cambiar\_clave\_inicial', methods=['GET', 'POST'])

@login\_required

def cambiar\_clave\_inicial():

if not current\_user.primer\_login\_requerido:

flash("Tu contraseña ya ha sido cambiada o no se requiere cambio inicial.", "info")

return redirect(url\_for('index')) # O a la página principal del usuario

if request.method == 'POST':

nueva\_clave = request.form['nueva\_clave']

confirmar\_clave = request.form['confirmar\_clave']

if not nueva\_clave or len(nueva\_clave) < 6: # Ejemplo de validación

flash("La nueva contraseña debe tener al menos 6 caracteres.", "danger")

elif nueva\_clave != confirmar\_clave:

flash("Las contraseñas no coinciden.", "danger")

else:

conn = conectar\_db()

cursor = conn.cursor()

try:

hashed\_password = generate\_password\_hash(nueva\_clave, method='pbkdf2:sha256')

cursor.execute("""

UPDATE usuarios SET password = ?, primer\_login\_requerido = 0

WHERE id\_usuario = ?

""", (hashed\_password, current\_user.id))

conn.commit()

# Actualizar el objeto current\_user en la sesión

# Esto es importante para que los cambios se reflejen sin necesidad de volver a loguear

current\_user.password = hashed\_password

current\_user.primer\_login\_requerido = 0

flash("Tu contraseña ha sido actualizada con éxito. Ya puedes acceder a la aplicación.", "success")

return redirect(url\_for('index')) # Redirige a la página principal

except sqlite3.Error as e:

conn.rollback()

flash(f"Error al actualizar la contraseña: {e}", "danger")

finally:

conn.close()

return render\_template('cambiar\_clave\_inicial.html') # Necesitarás crear esta plantilla

# --- MODIFICACIÓN: La ruta /carta ya no existe (la funcionalidad está en /hacer\_pedido) ---

# Se eliminó la función `carta` y su decorador `@app.route('/carta')`

@app.route('/hacer\_pedido', methods=['GET', 'POST'])

def hacer\_pedido():

"""

Ruta para que el cliente realice un nuevo pedido.

GET: Muestra el formulario con la carta y el carrito (unificado).

POST: Procesa el formulario, guarda el pedido en la DB y redirige a la confirmación.

"""

conn = conectar\_db()

cursor = conn.cursor()

# <--- MODIFICADO: Los clientes ven platos de la empresa por defecto

cursor.execute("SELECT id\_plato, nombre, descripcion, precio FROM platos WHERE activo = 1 AND id\_empresa = ? ORDER BY id\_plato ASC",

(DEFAULT\_COMPANY\_FOR\_ORDERS,))

platos\_db = cursor.fetchall()

conn.close()

if request.method == 'POST':

cliente\_nombre = request.form['nombre'].strip()

cliente\_apellido = request.form['apellido'].strip()

direccion\_entrega = request.form['direccion'].strip()

forma\_pago = request.form['forma\_pago']

horario\_str = request.form['horario\_entrega']

es\_envio\_solicitado = 'es\_envio\_solicitado' in request.form

# <--- AÑADIDO: Asignar el pedido a la empresa por defecto

pedido\_id\_empresa = DEFAULT\_COMPANY\_FOR\_ORDERS

form\_data\_on\_error = request.form.to\_dict()

form\_data\_on\_error['es\_envio\_solicitado'] = es\_envio\_solicitado

if not all([cliente\_nombre, cliente\_apellido, forma\_pago, horario\_str]):

flash("Todos los campos obligatorios (nombre, apellido, forma de pago, horario de entrega) deben ser completados.", "danger")

return render\_template('hacer\_pedido.html',

platos=platos\_db,

franjas\_horarias=\_generar\_franjas\_horarias\_disponibles(),

carrito\_detalle=\_get\_carrito\_detalle(platos\_db),

total\_carrito=\_get\_total\_carrito(),

costo\_envio=get\_costo\_envio(),

request\_form=form\_data\_on\_error,

carrito=session.get('carrito', {}))

if es\_envio\_solicitado and not direccion\_entrega:

flash("Si desea envío a domicilio, la dirección de entrega es obligatoria.", "danger")

return render\_template('hacer\_pedido.html',

platos=platos\_db,

franjas\_horarias=\_generar\_franjas\_horarias\_disponibles(),

carrito\_detalle=\_get\_carrito\_detalle(platos\_db),

total\_carrito=\_get\_total\_carrito(),

costo\_envio=get\_costo\_envio(),

request\_form=form\_data\_on\_error,

carrito=session.get('carrito', {}))

try:

horario\_entrega\_dt = datetime.strptime(horario\_str, '%H:%M')

hoy = datetime.now().date()

horario\_entrega\_completo = datetime.combine(hoy, horario\_entrega\_dt.time())

franjas\_ocupadas = \_cargar\_franjas\_ocupadas\_desde\_db\_interna()

if franjas\_ocupadas.get(horario\_entrega\_completo, 0) >= MAX\_PEDIDOS\_POR\_FRANJA\_HORARIA:

flash(f"Lo sentimos, el horario {horario\_str} se ha completado. Por favor, elija otra franja.", "danger")

return render\_template('hacer\_pedido.html',

platos=platos\_db,

franjas\_horarias=\_generar\_franjas\_horarias\_disponibles(),

carrito\_detalle=\_get\_carrito\_detalle(platos\_db),

total\_carrito=\_get\_total\_carrito(),

costo\_envio=get\_costo\_envio(),

request\_form=form\_data\_on\_error,

carrito=session.get('carrito', {}))

except ValueError:

flash("Horario de entrega inválido.", "danger")

return render\_template('hacer\_pedido.html',

platos=platos\_db,

franjas\_horarias=\_generar\_franjas\_horarias\_disponibles(),

carrito\_detalle=\_get\_carrito\_detalle(platos\_db),

total\_carrito=\_get\_total\_carrito(),

costo\_envio=get\_costo\_envio(),

request\_form=form\_data\_on\_error,

carrito=session.get('carrito', {}))

es\_envio = False

costo\_envio\_aplicado = 0.0

lat\_cliente, lon\_cliente = None, None

current\_envio\_costo = get\_costo\_envio() # Esta función ya contempla si el usuario está logueado o no

if es\_envio\_solicitado:

cliente\_lat\_lon = obtener\_coordenadas\_desde\_direccion(direccion\_entrega)

if cliente\_lat\_lon:

lat\_cliente, lon\_cliente = cliente\_lat\_lon

distancia\_cuadras = calcular\_distancia\_cuadras(SUCURSAL\_LAT, SUCURSAL\_LON, lat\_cliente, lon\_cliente)

if distancia\_cuadras <= RADIO\_ENVIO\_CUADRAS:

es\_envio = True

costo\_envio\_aplicado = current\_envio\_costo

flash(f"Su dirección está dentro del rango de envío ({distancia\_cuadras:.2f} cuadras). Costo de envío aplicado.", "info")

else:

flash(f"Su dirección ({distancia\_cuadras:.2f} cuadras) está fuera del rango de envío. El pedido será para retiro en sucursal y no se aplicará costo de envío.", "warning")

else:

flash("No se pudo validar su dirección para el envío. El pedido será para retiro en sucursal y no se aplicará costo de envío.", "warning")

else:

flash("El pedido será para retiro en sucursal.", "info")

if 'carrito' not in session or not session['carrito']:

flash("El carrito está vacío. Agregue productos antes de hacer el pedido.", "danger")

return render\_template('hacer\_pedido.html',

platos=platos\_db,

franjas\_horarias=\_generar\_franjas\_horarias\_disponibles(),

carrito\_detalle=\_get\_carrito\_detalle(platos\_db),

total\_carrito=\_get\_total\_carrito(),

costo\_envio=get\_costo\_envio(),

request\_form=form\_data\_on\_error,

carrito=session.get('carrito', {}))

items\_pedido\_para\_db = []

costo\_total\_pedido = costo\_envio\_aplicado

for item\_id, item\_data in session['carrito'].items():

plato = next((p for p in platos\_db if str(p['id\_plato']) == item\_id), None)

if plato:

cantidad = item\_data['cantidad']

precio\_unitario = plato['precio']

items\_pedido\_para\_db.append({"plato\_id": plato['id\_plato'], "cantidad": cantidad, "precio\_unitario": precio\_unitario})

costo\_total\_pedido += (cantidad \* precio\_unitario)

else:

flash(f"Producto con ID {item\_id} no encontrado en el catálogo. Por favor, revise su carrito.", "danger")

session.pop('carrito', None)

return redirect(url\_for('hacer\_pedido'))

conn = conectar\_db()

cursor = conn.cursor()

try:

fecha\_creacion\_str = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

horario\_entrega\_iso = horario\_entrega\_completo.strftime('%Y-%m-%d %H:%M:%S')

cursor.execute("""

INSERT INTO pedidos (

cliente\_nombre, cliente\_apellido, direccion\_entrega, es\_envio,

horario\_entrega, costo\_envio, costo\_total, forma\_pago, estado\_pago,

fecha\_creacion, lat\_cliente, lon\_cliente, id\_repartidor, id\_empresa

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

""", (

cliente\_nombre, cliente\_apellido, direccion\_entrega, int(es\_envio),

horario\_entrega\_iso, costo\_envio\_aplicado,

costo\_total\_pedido, forma\_pago, "Pendiente",

fecha\_creacion\_str, lat\_cliente, lon\_cliente,

None, pedido\_id\_empresa # <--- MODIFICADO: id\_empresa AÑADIDO

))

id\_nuevo\_pedido = cursor.lastrowid

for item in items\_pedido\_para\_db:

cursor.execute("""

INSERT INTO items\_pedido (id\_pedido, id\_plato, cantidad, precio\_unitario)

VALUES (?, ?, ?, ?)

""", (id\_nuevo\_pedido, item["plato\_id"], item["cantidad"], item["precio\_unitario"]))

conn.commit()

flash(f"Pedido #{id\_nuevo\_pedido} realizado con éxito!", "success")

session.pop('carrito', None)

return redirect(url\_for('pedido\_confirmacion', id\_pedido=id\_nuevo\_pedido))

except sqlite3.Error as e:

conn.rollback()

flash(f"Error al guardar el pedido: {e}", "danger")

return render\_template('hacer\_pedido.html',

platos=platos\_db,

franjas\_horarias=\_generar\_franjas\_horarias\_disponibles(),

carrito\_detalle=\_get\_carrito\_detalle(platos\_db),

total\_carrito=\_get\_total\_carrito(),

costo\_envio=get\_costo\_envio(),

request\_form=form\_data\_on\_error,

carrito=session.get('carrito', {}))

finally:

conn.close()

franjas\_horarias = \_generar\_franjas\_horarias\_disponibles()

return render\_template('hacer\_pedido.html',

platos=platos\_db,

franjas\_horarias=franjas\_horarias,

carrito\_detalle=\_get\_carrito\_detalle(platos\_db),

total\_carrito=\_get\_total\_carrito(),

costo\_envio=get\_costo\_envio(),

request\_form={},

carrito=session.get('carrito', {}))

# --- FUNCIONES AUXILIARES PARA EL CARRO (PARA RECARGAS CON ERRORES) ---

def \_get\_carrito\_detalle(platos\_db):

"""

Genera el detalle del carrito para la plantilla, útil para recargas con errores.

"""

carrito\_detalle = []

if 'carrito' in session and session['carrito']:

for item\_id, item\_data in session['carrito'].items():

plato = next((p for p in platos\_db if str(p['id\_plato']) == item\_id), None)

if plato:

subtotal = plato['precio'] \* item\_data['cantidad']

carrito\_detalle.append({

'id\_plato': plato['id\_plato'],

'nombre': plato['nombre'],

'precio\_unitario': plato['precio'],

'cantidad': item\_data['cantidad'],

'subtotal': subtotal

})

return carrito\_detalle

def \_get\_total\_carrito():

"""

Calcula el total de los productos en el carrito, útil para recargas con errores.

"""

total = 0.0

if 'carrito' in session:

for item\_data in session['carrito'].values():

# Nota: el 'precio' en la sesión se guarda cuando se añade el ítem.

# Idealmente, se debería consultar el precio actual de la DB para evitar desincronización

# si el precio del plato cambia después de añadirlo al carrito.

total += item\_data['precio'] \* item\_data['cantidad']

return total

# ---------------------------------------------------------------------------

@app.route('/pedido\_confirmacion/<int:id\_pedido>')

def pedido\_confirmacion(id\_pedido):

"""Muestra la confirmación del pedido y el ticket para imprimir."""

# <--- MODIFICADO: \_obtener\_pedido\_completo\_por\_id ya filtra por empresa (si el usuario está logueado)

# Sin embargo, el cliente final no está logueado, así que el super\_admin puede ver todos los pedidos.

# Esta ruta es POST-pedido, no tiene un filtro de seguridad fuerte.

pedido = \_obtener\_pedido\_completo\_por\_id(id\_pedido)

if not pedido:

flash("Pedido no encontrado.", "danger")

return redirect(url\_for('index'))

ticket\_html = pedido.generar\_ticket()

return render\_template('pedido\_confirmacion.html', pedido=pedido, ticket\_html=ticket\_html)

# --- Rutas de API para el Carrito (AJAX) ---

@app.route('/api/add\_to\_cart/<int:plato\_id>', methods=['POST'])

def add\_to\_cart(plato\_id):

"""Añade un plato al carrito de la sesión. Los clientes solo pueden añadir platos de la empresa por defecto."""

cantidad = int(request.form.get('cantidad', 1))

if cantidad <= 0:

return jsonify({"success": False, "message": "La cantidad debe ser un número positivo"}), 400

conn = conectar\_db()

cursor = conn.cursor()

# <--- MODIFICADO: Clientes solo pueden añadir platos de la empresa por defecto

cursor.execute("SELECT id\_plato, nombre, precio FROM platos WHERE id\_plato = ? AND activo = 1 AND id\_empresa = ?",

(plato\_id, DEFAULT\_COMPANY\_FOR\_ORDERS))

plato = cursor.fetchone()

conn.close()

if plato:

if 'carrito' not in session:

session['carrito'] = {}

plato\_id\_str = str(plato['id\_plato'])

if plato\_id\_str in session['carrito']:

session['carrito'][plato\_id\_str]['cantidad'] += cantidad

else:

session['carrito'][plato\_id\_str] = {

'nombre': plato['nombre'],

'precio': plato['precio'],

'cantidad': cantidad

}

session.modified = True

total\_items = sum(item['cantidad'] for item in session['carrito'].values())

return jsonify({"success": True, "message": f"{plato['nombre']} agregado al carrito.", "total\_items": total\_items})

return jsonify({"success": False, "message": "Plato no encontrado o inactivo"}), 404

@app.route('/api/remove\_from\_cart/<int:plato\_id>', methods=['POST'])

def remove\_from\_cart(plato\_id):

"""Elimina un plato específico del carrito de la sesión."""

if 'carrito' in session:

plato\_id\_str = str(plato\_id)

if plato\_id\_str in session['carrito']:

del session['carrito'][plato\_id\_str]

session.modified = True

total\_items = sum(item['cantidad'] for item in session['carrito'].values())

return jsonify({"success": True, "message": "Ítem eliminado del carrito.", "total\_items": total\_items})

return jsonify({"success": False, "message": "Ítem no encontrado en el carrito"}), 404

@app.route('/api/update\_cart\_quantity/<int:plato\_id>', methods=['POST'])

def update\_cart\_quantity(plato\_id):

"""

Actualiza la cantidad de un plato en el carrito de la sesión.

Si la cantidad es 0, elimina el plato. Si el plato no existe y la cantidad > 0, lo añade.

"""

cantidad = int(request.form.get('cantidad', 0))

plato\_id\_str = str(plato\_id)

# Asegura que el carrito exista en la sesión.

if 'carrito' not in session:

session['carrito'] = {}

# Si la cantidad deseada es 0 o menos

if cantidad <= 0:

if plato\_id\_str in session['carrito']:

del session['carrito'][plato\_id\_str]

session.modified = True

return jsonify({"success": True, "message": "Ítem eliminado del carrito."})

else:

# Si el ítem no estaba en el carrito y la cantidad es 0, lo consideramos un éxito para la UI.

return jsonify({"success": True, "message": "Ítem no encontrado en el carrito (no se pudo eliminar), pero la cantidad es 0."})

# Si la cantidad deseada es mayor a 0:

if plato\_id\_str in session['carrito']:

# Si ya existe en el carrito, actualizamos la cantidad

session['carrito'][plato\_id\_str]['cantidad'] = cantidad

session.modified = True

return jsonify({"success": True, "message": "Cantidad actualizada."})

else:

# Si NO existe en el carrito, pero la cantidad es > 0, buscamos el plato en la DB y lo añadimos

conn = conectar\_db()

cursor = conn.cursor()

# <--- MODIFICADO: Clientes solo pueden añadir platos de la empresa por defecto

cursor.execute("SELECT id\_plato, nombre, precio FROM platos WHERE id\_plato = ? AND activo = 1 AND id\_empresa = ?",

(plato\_id, DEFAULT\_COMPANY\_FOR\_ORDERS))

plato\_data = cursor.fetchone()

conn.close()

if plato\_data:

session['carrito'][plato\_id\_str] = {

'nombre': plato\_data['nombre'],

'precio': plato\_data['precio'],

'cantidad': cantidad

}

session.modified = True

return jsonify({"success": True, "message": f"{plato\_data['nombre']} añadido al carrito."})

else:

return jsonify({"success": False, "message": "Plato no encontrado o inactivo."}), 404

@app.route('/api/get\_cart\_status', methods=['GET'])

def get\_cart\_status():

"""Retorna el número total de ítems y el precio total actual del carrito."""

total\_items = sum(item['cantidad'] for item in session.get('carrito', {}).values())

total\_precio = 0

if 'carrito' in session:

for item\_data in session['carrito'].values():

total\_precio += item\_data['precio'] \* item\_data['cantidad']

return jsonify({"success": True, "total\_items": total\_items, "total\_precio": total\_precio})

@app.route('/api/clear\_cart', methods=['POST'])

def clear\_cart():

"""Limpia completamente el carrito de la sesión."""

session.pop('carrito', None)

session.modified = True

return jsonify({"success": True, "message": "Carrito vaciado.", "total\_items": 0})

# --- Rutas de Administración/Gestión ---

# app.py

# ... (código anterior) ...

@app.route('/gestion/pedidos')

@login\_required # Proteger esta ruta

def gestion\_pedidos():

"""Muestra una lista de todos los pedidos para gestión, filtrados por empresa."""

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa') or current\_user.has\_role('empleado')):

flash("No tienes permiso para acceder a esta página.", "danger")

return redirect(url\_for('index'))

conn = conectar\_db()

cursor = conn.cursor()

query = """

SELECT p.id\_pedido, p.cliente\_nombre, p.cliente\_apellido, p.direccion\_entrega, p.horario\_entrega,

p.forma\_pago, p.costo\_total, p.estado\_pago, p.es\_envio,

r.nombre AS repartidor\_nombre, r.apellido AS repartidor\_apellido,

e.nombre AS nombre\_empresa

FROM pedidos p

LEFT JOIN repartidores r ON p.id\_repartidor = r.id\_repartidor

LEFT JOIN empresas e ON p.id\_empresa = e.id\_empresa

"""

params = []

# Aplicar filtro de empresa a los pedidos

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params(table\_alias='p')

if company\_filter\_clause:

query += f" WHERE {company\_filter\_clause}"

params.extend(company\_filter\_params)

query += " ORDER BY p.horario\_entrega DESC"

cursor.execute(query, params)

pedidos = cursor.fetchall()

conn.close()

pedidos\_procesados = []

for p in pedidos:

p\_dict = dict(p)

p\_dict['horario\_entrega\_dt'] = datetime.strptime(p\_dict['horario\_entrega'], '%Y-%m-%d %H:%M:%S')

pedidos\_procesados.append(p\_dict)

# Repartidores también filtrados por empresa

conn = conectar\_db()

cursor = conn.cursor()

query\_repartidores = "SELECT id\_repartidor, nombre, apellido, activo FROM repartidores WHERE activo = 1"

params\_repartidores = []

company\_filter\_clause\_rep, company\_filter\_params\_rep = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause\_rep:

# <--- CORRECCIÓN APLICADA AQUÍ: Cambiar ' WHERE ' por ' AND '

query\_repartidores += f" AND {company\_filter\_clause\_rep}"

params\_repartidores.extend(company\_filter\_params\_rep)

query\_repartidores += " ORDER BY nombre, apellido"

cursor.execute(query\_repartidores, params\_repartidores)

repartidores = cursor.fetchall()

conn.close()

print(f"Repartidores cargados para gestión: {repartidores}")

return render\_template('gestion\_pedidos.html',

pedidos=pedidos\_procesados,

repartidores=repartidores)

# ... (resto del código) ...

@app.route('/gestion/pedido/<int:id\_pedido>/detalle')

@login\_required # Proteger esta ruta

def detalle\_pedido(id\_pedido):

"""Muestra el detalle completo de un pedido específico (vista de administración), filtrado por empresa."""

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa') or current\_user.has\_role('empleado')):

flash("No tienes permiso para acceder a esta página.", "danger")

return redirect(url\_for('index'))

# <--- MODIFICADO: \_obtener\_pedido\_completo\_por\_id ya filtra por empresa

pedido = \_obtener\_pedido\_completo\_por\_id(id\_pedido)

if not pedido:

flash("Pedido no encontrado o no tienes permiso para verlo.", "danger") # Mensaje más específico

return redirect(url\_for('gestion\_pedidos'))

ticket\_html = pedido.generar\_ticket()

return render\_template('pedido\_confirmacion.html', pedido=pedido, ticket\_html=ticket\_html, admin\_view=True)

@app.route('/gestion/pedido/<int:id\_pedido>/asignar\_repartidor', methods=['POST'])

@login\_required # Proteger esta ruta

def asignar\_repartidor(id\_pedido):

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_pedidos'))

id\_repartidor = request.form.get('id\_repartidor')

if not id\_repartidor:

flash("Debe seleccionar un repartidor.", "danger")

return redirect(url\_for('gestion\_pedidos'))

conn = conectar\_db()

cursor = conn.cursor()

try:

# <--- MODIFICADO: Añadir filtro de empresa al UPDATE

update\_query = "UPDATE pedidos SET id\_repartidor = ? WHERE id\_pedido = ?"

update\_params = [id\_repartidor, id\_pedido]

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause:

update\_query += f" AND {company\_filter\_clause}" # Solo actualizar si el pedido pertenece a la empresa del usuario

update\_params.extend(company\_filter\_params)

cursor.execute(update\_query, update\_params)

# Si no se actualizó ninguna fila y no es super\_admin, es un intento de modificar un pedido ajeno

if cursor.rowcount == 0 and not current\_user.has\_role('super\_admin'):

flash("Pedido no encontrado o no tienes permiso para asignarle un repartidor.", "danger")

conn.rollback()

return redirect(url\_for('gestion\_pedidos'))

conn.commit()

flash(f"Repartidor asignado al pedido #{id\_pedido} con éxito.", "success")

except sqlite3.Error as e:

conn.rollback()

flash(f"Error al asignar repartidor: {e}", "danger")

finally:

conn.close()

return redirect(url\_for('gestion\_pedidos'))

@app.route('/gestion/pedido/<int:id\_pedido>/marcar\_pagado', methods=['POST'])

@login\_required # Proteger esta ruta

def marcar\_pedido\_pagado(id\_pedido):

"""

Marca un pedido como 'Pagado' en la base de datos y registra un ingreso en la caja.

Si es un envío, registra el egreso al repartidor.

"""

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa') or current\_user.has\_role('empleado')):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_pedidos'))

# <--- MODIFICADO: \_obtener\_pedido\_completo\_por\_id ya filtra por empresa

pedido = \_obtener\_pedido\_completo\_por\_id(id\_pedido)

if not pedido:

flash(f"Pedido con ID {id\_pedido} no encontrado o no tienes permiso para verlo.", "danger")

return redirect(url\_for('gestion\_pedidos'))

if pedido.estado\_pago == 'Pagado':

flash(f"El pedido #{id\_pedido} ya está marcado como pagado.", "warning")

return redirect(url\_for('gestion\_pedidos'))

conn = conectar\_db()

cursor = conn.cursor()

try:

fecha\_pago\_str = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

# <--- MODIFICADO: Añadir filtro de empresa al UPDATE del pedido

update\_query\_pedido = "UPDATE pedidos SET estado\_pago = 'Pagado', fecha\_pago = ? WHERE id\_pedido = ?"

update\_params\_pedido = [fecha\_pago\_str, id\_pedido]

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause:

update\_query\_pedido += f" AND {company\_filter\_clause}"

update\_params\_pedido.extend(company\_filter\_params)

cursor.execute(update\_query\_pedido, update\_params\_pedido)

# Si no se actualizó ninguna fila y no es super\_admin, es un intento de modificar un pedido ajeno

if cursor.rowcount == 0 and not current\_user.has\_role('super\_admin'):

flash("Pedido no encontrado o no tienes permiso para marcarlo como pagado.", "danger")

conn.rollback()

return redirect(url\_for('gestion\_pedidos'))

# <--- MODIFICADO: Insertar ingreso con id\_empresa del pedido

cursor.execute("""

INSERT INTO ingresos\_egresos (tipo, monto, descripcion, fecha\_hora, id\_pedido\_origen, id\_repartidor\_origen, id\_empresa)

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

""", ('Ingreso', pedido.costo\_total, f"Pago de Pedido #{id\_pedido} ({pedido.forma\_pago})", fecha\_pago\_str, pedido.id\_pedido, pedido.id\_empresa))

if pedido.es\_envio and pedido.id\_repartidor:

pago\_repartidor = get\_pago\_repartidor\_por\_envio() # Esta función ya carga la configuración de la empresa correcta

# <--- MODIFICADO: Insertar egreso a repartidor con id\_empresa del pedido

cursor.execute("""

INSERT INTO ingresos\_egresos (tipo, monto, descripcion, fecha\_hora, id\_pedido\_origen, id\_repartidor\_origen, id\_empresa)

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

""", ('Pago a Repartidor', pago\_repartidor, f"Pago por envío Pedido #{id\_pedido}", fecha\_pago\_str, pedido.id\_pedido, pedido.id\_repartidor, pedido.id\_empresa))

flash(f"Se registró un pago de ${pago\_repartidor:,.2f} al repartidor por este envío.", "info")

conn.commit()

flash(f"Pedido #{id\_pedido} marcado como pagado y registrado como ingreso.", "success")

except sqlite3.Error as e:

conn.rollback()

flash(f"Error al marcar el pedido como pagado: {e}", "danger")

finally:

conn.close()

return redirect(url\_for('gestion\_pedidos'))

@app.route('/gestion/catalogo')

@login\_required # Proteger esta ruta

def gestion\_catalogo():

"""Muestra una lista completa de todos los platos (activos e inactivos) para gestión, filtrados por empresa."""

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa') or current\_user.has\_role('empleado')):

flash("No tienes permiso para acceder a esta página.", "danger")

return redirect(url\_for('index'))

conn = conectar\_db()

cursor = conn.cursor()

query = "SELECT id\_plato, nombre, descripcion, precio, activo, id\_empresa FROM platos" # <--- id\_empresa incluido para consistencia

params = []

# <--- MODIFICADO: Aplicar filtro de empresa a los platos

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause:

query += f" WHERE {company\_filter\_clause}"

params.extend(company\_filter\_params)

query += " ORDER BY id\_plato ASC"

cursor.execute(query, params)

platos = cursor.fetchall()

conn.close()

return render\_template('gestion\_catalogo.html', platos=platos)

@app.route('/gestion/catalogo/agregar', methods=['GET', 'POST'])

@login\_required # Proteger esta ruta

def agregar\_plato():

"""

Ruta para agregar un nuevo plato al catálogo.

GET: Muestra el formulario para agregar plato.

POST: Procesa el formulario y guarda el nuevo plato en la DB.

"""

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_catalogo'))

if request.method == 'POST':

nombre = request.form['nombre'].strip()

descripcion = request.form['descripcion'].strip()

try:

precio = float(request.form['precio'].replace(',', '.'))

if precio <= 0:

flash("El precio debe ser un número positivo.", "danger")

return redirect(url\_for('agregar\_plato'))

except ValueError:

flash("Precio inválido. Ingrese un número.", "danger")

return redirect(url\_for('agregar\_plato'))

conn = conectar\_db()

cursor = conn.cursor()

try:

# <--- MODIFICADO: Asignar id\_empresa al plato

plato\_id\_empresa = current\_user.id\_empresa

if current\_user.has\_role('super\_admin'):

# Super Admin podría elegir una empresa. Por simplicidad, si no se especifica,

# lo asignamos a la empresa por defecto (DEFAULT\_COMPANY\_FOR\_ORDERS).

# Si el super\_admin quiere asignar a otra, se necesitaría un selector en la UI.

plato\_id\_empresa = request.form.get('id\_empresa\_asignar', DEFAULT\_COMPANY\_FOR\_ORDERS)

flash(f"Como Super Admin, el plato se ha asignado a la Empresa ID {plato\_id\_empresa} por defecto. Considere añadir un selector de empresa.", "info")

elif not plato\_id\_empresa: # Si no es super\_admin y no tiene empresa asignada (error de config)

flash("Tu usuario no tiene una empresa asignada para agregar platos.", "danger")

return redirect(url\_for('gestion\_catalogo'))

cursor.execute("INSERT INTO platos (nombre, descripcion, precio, activo, id\_empresa) VALUES (?, ?, ?, 1, ?)",

(nombre, descripcion, precio, plato\_id\_empresa))

conn.commit()

flash(f"Plato '{nombre}' agregado con éxito.", "success")

except sqlite3.Error as e:

flash(f"Error al agregar plato: {e}", "danger")

finally:

conn.close()

return redirect(url\_for('gestion\_catalogo'))

# Si el super\_admin puede elegir empresa, cargar empresas aquí

empresas\_disponibles = []

if current\_user.has\_role('super\_admin'):

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("SELECT id\_empresa, nombre FROM empresas WHERE activo = 1 ORDER BY nombre")

empresas\_disponibles = cursor.fetchall()

conn.close()

return render\_template('agregar\_plato.html', empresas\_disponibles=empresas\_disponibles)

@app.route('/gestion/catalogo/editar/<int:id\_plato>', methods=['GET', 'POST'])

@login\_required # Proteger esta ruta

def editar\_plato(id\_plato):

"""

Ruta para editar un plato existente, filtrado por empresa.

GET: Muestra el formulario con los datos actuales del plato.

POST: Procesa el formulario y actualiza los datos del plato en la DB.

"""

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_catalogo'))

conn = conectar\_db()

cursor = conn.cursor()

query = "SELECT id\_plato, nombre, descripcion, precio, activo, id\_empresa FROM platos WHERE id\_plato = ?"

params = [id\_plato]

# <--- MODIFICADO: Aplicar filtro de empresa para recuperar el plato

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause:

query += f" AND {company\_filter\_clause}"

params.extend(company\_filter\_params)

cursor.execute(query, params)

plato = cursor.fetchone()

conn.close()

if not plato:

flash("Plato no encontrado o no tienes permiso para editarlo.", "danger")

return redirect(url\_for('gestion\_catalogo'))

if request.method == 'POST':

nombre = request.form['nombre'].strip()

descripcion = request.form['descripcion'].strip()

activo = 1 if 'activo' in request.form else 0

try:

precio = float(request.form['precio'].replace(',', '.'))

if precio <= 0:

flash("El precio debe ser un número positivo.", "danger")

return redirect(url\_for('editar\_plato', id\_plato=id\_plato))

except ValueError:

flash("Precio inválido. Ingrese un número.", "danger")

return redirect(url\_for('editar\_plato', id\_plato=id\_plato))

conn = conectar\_db()

cursor = conn.cursor()

try:

# <--- MODIFICADO: Añadir filtro de empresa al UPDATE

update\_query = """

UPDATE platos SET

nombre = ?,

descripcion = ?,

precio = ?,

activo = ?

WHERE id\_plato = ?

"""

update\_params = [nombre, descripcion, precio, activo, id\_plato]

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause:

update\_query += f" AND {company\_filter\_clause}"

update\_params.extend(company\_filter\_params)

cursor.execute(update\_query, update\_params)

# Si no se actualizó ninguna fila y no es super\_admin, es un intento de modificar un plato ajeno

if cursor.rowcount == 0 and not current\_user.has\_role('super\_admin'):

flash("Plato no encontrado o no tienes permiso para editarlo.", "danger")

conn.rollback()

return redirect(url\_for('gestion\_catalogo'))

conn.commit()

flash(f"Plato '{nombre}' actualizado con éxito.", "success")

except sqlite3.Error as e:

flash(f"Error al editar plato: {e}", "danger")

finally:

conn.close()

return redirect(url\_for('gestion\_catalogo'))

return render\_template('editar\_plato.html', plato=plato)

@app.route('/gestion/catalogo/eliminar/<int:id\_plato>', methods=['POST'])

@login\_required # Proteger esta ruta

def eliminar\_plato(id\_plato):

"""

Elimina lógicamente un plato (lo marca como inactivo) para que no aparezca en la carta, filtrado por empresa.

"""

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_catalogo'))

conn = conectar\_db()

cursor = conn.cursor()

try:

# <--- MODIFICADO: Añadir filtro de empresa al UPDATE

update\_query = "UPDATE platos SET activo = 0 WHERE id\_plato = ?"

update\_params = [id\_plato]

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause:

update\_query += f" AND {company\_filter\_clause}"

update\_params.extend(company\_filter\_params)

cursor.execute(update\_query, update\_params)

# Si no se actualizó ninguna fila y no es super\_admin, es un intento de modificar un plato ajeno

if cursor.rowcount == 0 and not current\_user.has\_role('super\_admin'):

flash("Plato no encontrado o no tienes permiso para inactivarlo.", "danger")

conn.rollback()

return redirect(url\_for('gestion\_catalogo'))

conn.commit()

flash(f"Plato con ID {id\_plato} marcado como inactivo.", "success")

except sqlite3.Error as e:

flash(f"Error al inactivar plato: {e}", "danger")

finally:

conn.close()

return redirect(url\_for('gestion\_catalogo'))

@app.route('/gestion/caja', methods=['GET', 'POST'])

@login\_required # Proteger esta ruta

def arqueo\_caja():

"""

Ruta para registrar egresos manuales y realizar arqueos de caja, filtrado por empresa.

GET: Muestra los formularios.

POST: Procesa el registro de egresos o la solicitud de arqueo.

"""

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para acceder a esta página.", "danger")

return redirect(url\_for('index'))

if request.method == 'POST':

if 'registrar\_egreso' in request.form:

try:

monto = float(request.form['monto'].replace(',', '.'))

if monto <= 0:

flash("El monto debe ser un número positivo.", "danger")

return redirect(url\_for('arqueo\_caja'))

descripcion = request.form['descripcion'].strip()

if not descripcion:

flash("La descripción del egreso no puede estar vacía.", "danger")

return redirect(url\_for('arqueo\_caja'))

conn = conectar\_db()

cursor = conn.cursor()

try:

fecha\_hora\_str = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

# <--- MODIFICADO: Asignar id\_empresa al egreso

egreso\_id\_empresa = current\_user.id\_empresa

if current\_user.has\_role('super\_admin'):

# Super Admin puede elegir una empresa para el egreso.

egreso\_id\_empresa = request.form.get('id\_empresa\_asignar\_egreso', DEFAULT\_COMPANY\_FOR\_ORDERS)

flash(f"Como Super Admin, el egreso se ha asignado a la Empresa ID {egreso\_id\_empresa} por defecto.", "info")

elif not egreso\_id\_empresa:

flash("Tu usuario no tiene una empresa asignada para registrar egresos.", "danger")

return redirect(url\_for('arqueo\_caja'))

cursor.execute("""

INSERT INTO ingresos\_egresos (tipo, monto, descripcion, fecha\_hora, id\_pedido\_origen, id\_repartidor\_origen, id\_empresa)

VALUES ('Egreso', ?, ?, ?, NULL, NULL, ?)

""", (monto, descripcion, fecha\_hora\_str, egreso\_id\_empresa))

conn.commit()

flash(f"Egreso de ${monto:,.2f} registrado con éxito.", "success")

except sqlite3.Error as e:

flash(f"Error al registrar egreso: {e}", "danger")

finally:

conn.close()

except ValueError:

flash("Monto inválido. Ingrese un número.", "danger")

return redirect(url\_for('arqueo\_caja'))

elif 'realizar\_arqueo' in request.form:

fecha\_inicio\_str = request.form['fecha\_inicio'].strip()

fecha\_fin\_str = request.form['fecha\_fin'].strip()

try:

fecha\_inicio = datetime.strptime(fecha\_inicio\_str, '%Y-%m-%d').replace(hour=0, minute=0, second=0, microsecond=0)

fecha\_fin = datetime.strptime(fecha\_fin\_str, '%Y-%m-%d').replace(hour=23, minute=59, second=59, microsecond=999999)

if fecha\_inicio > fecha\_fin:

flash("La fecha de inicio no puede ser posterior a la fecha de fin.", "danger")

return redirect(url\_for('arqueo\_caja'))

conn = conectar\_db()

cursor = conn.cursor()

fecha\_inicio\_iso = fecha\_inicio.strftime('%Y-%m-%d %H:%M:%S')

fecha\_fin\_iso = fecha\_fin.strftime('%Y-%m-%d %H:%M:%S')

query = """

SELECT ie.tipo, ie.monto, ie.descripcion, ie.fecha\_hora, ie.id\_pedido\_origen,

r.nombre AS repartidor\_nombre, r.apellido AS repartidor\_apellido,

e.nombre AS nombre\_empresa

FROM ingresos\_egresos ie

LEFT JOIN repartidores r ON ie.id\_repartidor\_origen = r.id\_repartidor

LEFT JOIN empresas e ON ie.id\_empresa = e.id\_empresa

WHERE ie.fecha\_hora BETWEEN ? AND ?

"""

params = [fecha\_inicio\_iso, fecha\_fin\_iso]

# <--- MODIFICADO: Aplicar filtro de empresa a ingresos\_egresos

company\_filter\_clause\_ie, company\_filter\_params\_ie = get\_company\_filter\_clause\_and\_params(table\_alias='ie')

if company\_filter\_clause\_ie:

query += f" AND {company\_filter\_clause\_ie}"

params.extend(company\_filter\_params\_ie)

query += " ORDER BY ie.fecha\_hora ASC"

cursor.execute(query, params)

movimientos = cursor.fetchall()

conn.close()

total\_ingresos = sum(m['monto'] for m in movimientos if m['tipo'] == 'Ingreso')

total\_egresos = sum(m['monto'] for m in movimientos if m['tipo'] != 'Ingreso')

balance = total\_ingresos - total\_egresos

movimientos\_procesados = []

for m in movimientos:

m\_dict = dict(m)

fecha\_dt = datetime.strptime(m\_dict['fecha\_hora'], '%Y-%m-%d %H:%M:%S')

m\_dict['fecha\_hora\_formateada'] = fecha\_dt.strftime('%d/%m/%Y %H:%M')

if m\_dict['repartidor\_nombre'] and m\_dict['repartidor\_apellido']:

m\_dict['repartidor\_nombre\_completo'] = f"{m\_dict['repartidor\_nombre']} {m\_dict['repartidor\_apellido']}"

else:

m\_dict['repartidor\_nombre\_completo'] = None

movimientos\_procesados.append(m\_dict)

session['arqueo\_resultados'] = {

'fecha\_inicio': fecha\_inicio.strftime('%d/%m/%Y'),

'fecha\_fin': fecha\_fin.strftime('%d/%m/%Y'),

'movimientos': movimientos\_procesados,

'total\_ingresos': total\_ingresos,

'total\_egresos': total\_egresos,

'balance': balance

}

return redirect(url\_for('arqueo\_caja'))

except ValueError:

flash("Formato de fecha inválido. Use AAAA-MM-DD.", "danger")

return redirect(url\_for('arqueo\_caja'))

arqueo\_resultados = session.pop('arqueo\_resultados', None)

# Cargar empresas para el selector si es super\_admin

empresas\_para\_egreso = []

if current\_user.has\_role('super\_admin'):

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("SELECT id\_empresa, nombre FROM empresas WHERE activo = 1 ORDER BY nombre")

empresas\_para\_egreso = cursor.fetchall()

conn.close()

return render\_template('arqueo\_caja.html',

arqueo\_resultados=arqueo\_resultados,

now=datetime.now(),

empresas\_para\_egreso=empresas\_para\_egreso)

@app.route('/gestion/configuracion', methods=['GET', 'POST'])

@login\_required # Proteger esta ruta

def gestion\_configuracion():

"""

Permite visualizar y editar valores de configuración como el costo de envío

y el pago por envío al repartidor, por empresa o globalmente.

"""

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para acceder a esta página.", "danger")

return redirect(url\_for('index'))

if request.method == 'POST':

# Determinar la id\_empresa para guardar la configuración

config\_id\_empresa = None # Por defecto, será global

if current\_user.has\_role('admin\_empresa') and current\_user.id\_empresa:

config\_id\_empresa = current\_user.id\_empresa

elif current\_user.has\_role('super\_admin'):

# El super\_admin puede elegir una empresa desde un selector si existe en el formulario

config\_id\_empresa = request.form.get('config\_for\_company')

if config\_id\_empresa:

config\_id\_empresa = int(config\_id\_empresa)

# Si super\_admin no selecciona empresa, o la selecciona como 'global', se usa NULL

if config\_id\_empresa == 0: # Convención: 0 puede ser "global" en el selector

config\_id\_empresa = None

if 'update\_envio\_costo' in request.form:

try:

nuevo\_costo\_envio = float(request.form['costo\_envio'].replace(',', '.'))

if nuevo\_costo\_envio < 0:

flash("El costo de envío no puede ser negativo.", "danger")

else:

guardar\_configuracion('ENVIO\_COSTO', nuevo\_costo\_envio, config\_id\_empresa) # <--- MODIFICADO: id\_empresa añadido

flash(f"Costo de envío actualizado a ${nuevo\_costo\_envio:,.2f} {'para tu empresa' if config\_id\_empresa else 'globalmente'}.", "success")

except ValueError:

flash("Valor de costo de envío inválido. Ingrese un número.", "danger")

elif 'update\_pago\_repartidor' in request.form:

try:

nuevo\_pago\_repartidor = float(request.form['pago\_repartidor\_por\_envio'].replace(',', '.'))

if nuevo\_pago\_repartidor < 0:

flash("El pago al repartidor no puede ser negativo.", "danger")

else:

guardar\_configuracion('PAGO\_REPARTIDOR\_POR\_ENVIO', nuevo\_pago\_repartidor, config\_id\_empresa) # <--- MODIFICADO: id\_empresa añadido

flash(f"Pago por envío a repartidor actualizado a ${nuevo\_pago\_repartidor:,.2f} {'para tu empresa' if config\_id\_empresa else 'globalmente'}.", "success")

except ValueError:

flash("Valor de pago a repartidor inválido. Ingrese un número.", "danger")

return redirect(url\_for('gestion\_configuracion'))

# Cargar configuración específica para la empresa del usuario logueado o global para super\_admin

costo\_envio\_actual = get\_costo\_envio() # Ya utiliza la lógica de current\_user.id\_empresa o global

pago\_repartidor\_actual = get\_pago\_repartidor\_por\_envio() # Ya utiliza la lógica de current\_user.id\_empresa o global

# Si eres super\_admin, podrías querer mostrar un selector de empresas para configurar

empresas\_para\_config = []

if current\_user.has\_role('super\_admin'):

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("SELECT id\_empresa, nombre FROM empresas WHERE activo = 1 ORDER BY nombre")

empresas\_para\_config = cursor.fetchall()

conn.close()

return render\_template('gestion\_configuracion.html',

costo\_envio\_actual=costo\_envio\_actual,

pago\_repartidor\_actual=pago\_repartidor\_actual,

empresas\_para\_config=empresas\_para\_config,

current\_user\_company\_id=current\_user.id\_empresa if current\_user.is\_authenticated else None)

# --- RUTAS DE GESTIÓN DE REPARTIDORES ---

@app.route('/gestion/repartidores')

@login\_required # Proteger esta ruta

def gestion\_repartidores():

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para acceder a esta página.", "danger")

return redirect(url\_for('index'))

conn = conectar\_db()

cursor = conn.cursor()

query = "SELECT id\_repartidor, nombre, apellido, telefono, activo, id\_empresa FROM repartidores" # <--- id\_empresa incluido

params = []

# <--- MODIFICADO: Aplicar filtro de empresa

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause:

query += f" WHERE {company\_filter\_clause}"

params.extend(company\_filter\_params)

query += " ORDER BY apellido, nombre"

cursor.execute(query, params)

repartidores = cursor.fetchall()

conn.close()

return render\_template('gestion\_repartidores.html', repartidores=repartidores)

@app.route('/gestion/repartidores/agregar', methods=['GET', 'POST'])

@login\_required # Proteger esta ruta

def agregar\_repartidor():

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_repartidores'))

if request.method == 'POST':

nombre = request.form['nombre'].strip()

apellido = request.form['apellido'].strip()

telefono = request.form['telefono'].strip()

if not nombre or not apellido:

flash("Nombre y apellido del repartidor son obligatorios.", "danger")

return redirect(url\_for('agregar\_repartidor'))

conn = conectar\_db()

cursor = conn.cursor()

try:

# <--- MODIFICADO: Asignar id\_empresa al repartidor

repartidor\_id\_empresa = current\_user.id\_empresa

if current\_user.has\_role('super\_admin'):

repartidor\_id\_empresa = request.form.get('id\_empresa\_asignar', DEFAULT\_COMPANY\_FOR\_ORDERS)

flash(f"Como Super Admin, el repartidor se ha asignado a la Empresa ID {repartidor\_id\_empresa} por defecto.", "info")

elif not repartidor\_id\_empresa:

flash("Tu usuario no tiene una empresa asignada para agregar repartidores.", "danger")

return redirect(url\_for('gestion\_repartidores'))

cursor.execute("INSERT INTO repartidores (nombre, apellido, telefono, activo, id\_empresa) VALUES (?, ?, ?, 1, ?)",

(nombre, apellido, telefono, repartidor\_id\_empresa))

conn.commit()

flash(f"Repartidor '{nombre} {apellido}' agregado con éxito.", "success")

except sqlite3.Error as e:

flash(f"Error al agregar repartidor: {e}", "danger")

finally:

conn.close()

return redirect(url\_for('gestion\_repartidores'))

# Si el super\_admin puede elegir empresa, cargar empresas aquí

empresas\_disponibles = []

if current\_user.has\_role('super\_admin'):

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("SELECT id\_empresa, nombre FROM empresas WHERE activo = 1 ORDER BY nombre")

empresas\_disponibles = cursor.fetchall()

conn.close()

return render\_template('agregar\_repartidor.html', empresas\_disponibles=empresas\_disponibles)

@app.route('/gestion/repartidores/editar/<int:id\_repartidor>', methods=['GET', 'POST'])

@login\_required # Proteger esta ruta

def editar\_repartidor(id\_repartidor):

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_repartidores'))

conn = conectar\_db()

cursor = conn.cursor()

query = "SELECT id\_repartidor, nombre, apellido, telefono, activo, id\_empresa FROM repartidores WHERE id\_repartidor = ?"

params = [id\_repartidor]

# <--- MODIFICADO: Aplicar filtro de empresa para recuperar el repartidor

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause:

query += f" AND {company\_filter\_clause}"

params.extend(company\_filter\_params)

cursor.execute(query, params)

repartidor = cursor.fetchone()

conn.close()

if not repartidor:

flash("Repartidor no encontrado o no tienes permiso para editarlo.", "danger")

return redirect(url\_for('gestion\_repartidores'))

if request.method == 'POST':

nombre = request.form['nombre'].strip()

apellido = request.form['apellido'].strip()

telefono = request.form['telefono'].strip()

activo = 1 if 'activo' in request.form else 0

if not nombre or not apellido:

flash("Nombre y apellido del repartidor son obligatorios.", "danger")

return redirect(url\_for('editar\_repartidor', id\_repartidor=id\_repartidor))

conn = conectar\_db()

cursor = conn.cursor()

try:

# <--- MODIFICADO: Añadir filtro de empresa al UPDATE

update\_query = """

UPDATE repartidores SET

nombre = ?,

apellido = ?,

telefono = ?,

activo = ?

WHERE id\_repartidor = ?

"""

update\_params = [nombre, apellido, telefono, activo, id\_repartidor]

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause:

update\_query += f" AND {company\_filter\_clause}"

update\_params.extend(company\_filter\_params)

cursor.execute(update\_query, update\_params)

# Si no se actualizó ninguna fila y no es super\_admin, es un intento de modificar un repartidor ajeno

if cursor.rowcount == 0 and not current\_user.has\_role('super\_admin'):

flash("Repartidor no encontrado o no tienes permiso para editarlo.", "danger")

conn.rollback()

return redirect(url\_for('gestion\_repartidores'))

conn.commit()

flash(f"Repartidor '{nombre} {apellido}' actualizado con éxito.", "success")

except sqlite3.Error as e:

flash(f"Error al editar repartidor: {e}", "danger")

finally:

conn.close()

return redirect(url\_for('gestion\_repartidores'))

return render\_template('editar\_repartidor.html', repartidor=repartidor)

@app.route('/gestion/repartidores/eliminar/<int:id\_repartidor>', methods=['POST'])

@login\_required # Proteger esta ruta

def eliminar\_repartidor(id\_repartidor):

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_repartidores'))

conn = conectar\_db()

cursor = conn.cursor()

try:

# <--- MODIFICADO: Añadir filtro de empresa al UPDATE

update\_query = "UPDATE repartidores SET activo = 0 WHERE id\_repartidor = ?"

update\_params = [id\_repartidor]

company\_filter\_clause, company\_filter\_params = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause:

update\_query += f" AND {company\_filter\_clause}"

update\_params.extend(company\_filter\_params)

cursor.execute(update\_query, update\_params)

# Si no se actualizó ninguna fila y no es super\_admin, es un intento de modificar un repartidor ajeno

if cursor.rowcount == 0 and not current\_user.has\_role('super\_admin'):

flash("Repartidor no encontrado o no tienes permiso para inactivarlo.", "danger")

conn.rollback()

return redirect(url\_for('gestion\_repartidores'))

conn.commit()

flash(f"Repartidor con ID {id\_repartidor} marcado como inactivo.", "success")

except sqlite3.Error as e:

flash(f"Error al inactivar repartidor: {e}", "danger")

finally:

conn.close()

return redirect(url\_for('gestion\_repartidores'))

@app.route('/gestion/reporte\_repartidores', methods=['GET', 'POST'])

@login\_required # Proteger esta ruta

def reporte\_repartidores():

if not (current\_user.has\_role('super\_admin') or current\_user.has\_role('admin\_empresa')):

flash("No tienes permiso para acceder a esta página.", "danger")

return redirect(url\_for('index'))

conn = conectar\_db()

cursor = conn.cursor()

query\_repartidores = "SELECT id\_repartidor, nombre, apellido FROM repartidores WHERE activo = 1"

params\_repartidores = []

# <--- MODIFICADO: Aplicar filtro de empresa a la lista de repartidores

company\_filter\_clause\_rep, company\_filter\_params\_rep = get\_company\_filter\_clause\_and\_params()

if company\_filter\_clause\_rep:

query\_repartidores += f" AND {company\_filter\_clause\_rep}"

params\_repartidores.extend(company\_filter\_params\_rep)

query\_repartidores += " ORDER BY apellido, nombre"

cursor.execute(query\_repartidores, params\_repartidores)

repartidores\_activos = cursor.fetchall()

conn.close()

reporte\_generado = None

if request.method == 'POST':

id\_repartidor\_seleccionado = request.form.get('id\_repartidor')

fecha\_inicio\_str = request.form.get('fecha\_inicio')

fecha\_fin\_str = request.form.get('fecha\_fin')

try:

fecha\_inicio = datetime.strptime(fecha\_inicio\_str, '%Y-%m-%d').replace(hour=0, minute=0, second=0, microsecond=0)

fecha\_fin = datetime.strptime(fecha\_fin\_str, '%Y-%m-%d').replace(hour=23, minute=59, second=59, microsecond=999999)

if fecha\_inicio > fecha\_fin:

flash("La fecha de inicio no puede ser posterior a la fecha de fin.", "danger")

return redirect(url\_for('reporte\_repartidores'))

conn = conectar\_db()

cursor = conn.cursor()

query = """

SELECT ie.fecha\_hora, ie.monto, ie.id\_pedido\_origen,

r.nombre AS repartidor\_nombre, r.apellido AS repartidor\_apellido,

e.nombre AS nombre\_empresa

FROM ingresos\_egresos ie

JOIN repartidores r ON ie.id\_repartidor\_origen = r.id\_repartidor

LEFT JOIN empresas e ON ie.id\_empresa = e.id\_empresa

WHERE ie.tipo = 'Pago a Repartidor'

AND ie.fecha\_hora BETWEEN ? AND ?

"""

params = [fecha\_inicio.strftime('%Y-%m-%d %H:%M:%S'), fecha\_fin.strftime('%Y-%m-%d %H:%M:%S')]

if id\_repartidor\_seleccionado and id\_repartidor\_seleccionado != 'todos':

query += " AND ie.id\_repartidor\_origen = ?"

params.append(id\_repartidor\_seleccionado)

# <--- MODIFICADO: Aplicar filtro de empresa al reporte de ingresos\_egresos

company\_filter\_clause\_ie, company\_filter\_params\_ie = get\_company\_filter\_clause\_and\_params(table\_alias='ie')

if company\_filter\_clause\_ie:

query += f" AND {company\_filter\_clause\_ie}"

params.extend(company\_filter\_params\_ie)

query += " ORDER BY ie.fecha\_hora ASC"

cursor.execute(query, params)

pagos = cursor.fetchall()

conn.close()

total\_pagado = sum(p['monto'] for p in pagos)

repartidor\_nombre\_reporte = "Todos los Repartidores"

if id\_repartidor\_seleccionado and id\_repartidor\_seleccionado != 'todos':

for rep in repartidores\_activos:

if str(rep['id\_repartidor']) == id\_repartidor\_seleccionado:

repartidor\_nombre\_reporte = f"{rep['nombre']} {rep['apellido']}"

break

pagos\_procesados = []

for p in pagos:

p\_dict = dict(p)

p\_dict['fecha\_hora\_formateada'] = datetime.strptime(p\_dict['fecha\_hora'], '%Y-%m-%d %H:%M:%S').strftime('%d/%m/%Y %H:%M')

pagos\_procesados.append(p\_dict)

reporte\_generado = {

'fecha\_inicio': fecha\_inicio.strftime('%d/%m/%Y'),

'fecha\_fin': fecha\_fin.strftime('%d/%m/%Y'),

'repartidor\_nombre': repartidor\_nombre\_reporte,

'pagos': pagos\_procesados,

'total\_pagado': total\_pagado

}

except ValueError:

flash("Formato de fecha inválido. Use AAAA-MM-DD.", "danger")

return redirect(url\_for('reporte\_repartidores'))

return render\_template('reporte\_repartidores.html',

repartidores=repartidores\_activos,

reporte\_generado=reporte\_generado,

now=datetime.now())

# --- NUEVAS RUTAS DE GESTIÓN DE EMPRESAS Y USUARIOS (SUPER ADMIN) ---

# Estas rutas no se modifican con get\_company\_filter\_clause\_and\_params

# porque el super\_admin gestiona todas las empresas y usuarios.

# La asignación de id\_empresa a usuarios se realiza al agregar/editar usuarios.

@app.route('/gestion/empresas')

@login\_required

def gestion\_empresas():

if not current\_user.has\_role('super\_admin'):

flash("No tienes permiso para acceder a esta página.", "danger")

return redirect(url\_for('index'))

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("SELECT id\_empresa, nombre, activo FROM empresas ORDER BY nombre")

empresas = cursor.fetchall()

conn.close()

return render\_template('gestion\_empresas.html', empresas=empresas)

@app.route('/gestion/empresas/agregar', methods=['GET', 'POST'])

@login\_required

def agregar\_empresa():

if not current\_user.has\_role('super\_admin'):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_empresas'))

if request.method == 'POST':

nombre = request.form['nombre'].strip()

if not nombre:

flash("El nombre de la empresa es obligatorio.", "danger")

return render\_template('agregar\_empresa.html', request\_form=request.form)

conn = conectar\_db()

cursor = conn.cursor()

try:

cursor.execute("INSERT INTO empresas (nombre, activo) VALUES (?, 1)", (nombre,))

conn.commit()

flash(f"Empresa '{nombre}' agregada con éxito.", "success")

except sqlite3.IntegrityError:

flash(f"Ya existe una empresa con el nombre '{nombre}'.", "danger")

return render\_template('agregar\_empresa.html', request\_form=request.form)

except sqlite3.Error as e:

flash(f"Error al agregar empresa: {e}", "danger")

return render\_template('agregar\_empresa.html', request\_form=request.form)

finally:

conn.close()

return redirect(url\_for('gestion\_empresas'))

return render\_template('agregar\_empresa.html', request\_form={})

@app.route('/gestion/empresas/editar/<int:id\_empresa>', methods=['GET', 'POST'])

@login\_required

def edita\_empresa(id\_empresa): # Renombrado de editar\_empresa a edita\_empresa

if not current\_user.has\_role('super\_admin'):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_empresas'))

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("SELECT id\_empresa, nombre, activo FROM empresas WHERE id\_empresa = ?", (id\_empresa,))

empresa = cursor.fetchone()

conn.close()

if not empresa:

flash("Empresa no encontrada.", "danger")

return redirect(url\_for('gestion\_empresas'))

if request.method == 'POST':

nombre = request.form['nombre'].strip()

activo = 1 if 'activo' in request.form else 0

if not nombre:

flash("El nombre de la empresa es obligatorio.", "danger")

return render\_template('editar\_empresa.html', empresa=empresa, request\_form=request.form)

conn = conectar\_db()

cursor = conn.cursor()

try:

cursor.execute("UPDATE empresas SET nombre = ?, activo = ? WHERE id\_empresa = ?",

(nombre, activo, id\_empresa))

conn.commit()

flash(f"Empresa '{nombre}' actualizada con éxito.", "success")

except sqlite3.IntegrityError:

flash(f"Ya existe una empresa con el nombre '{nombre}'.", "danger")

return render\_template('editar\_empresa.html', empresa=empresa, request\_form=request.form)

except sqlite3.Error as e:

flash(f"Error al editar empresa: {e}", "danger")

return render\_template('editar\_empresa.html', empresa=empresa, request\_form=request.form)

finally:

conn.close()

return redirect(url\_for('gestion\_empresas'))

return render\_template('editar\_empresa.html', empresa=empresa, request\_form={})

# <--- AÑADIDO: Ruta para eliminar (inactivar) empresa

@app.route('/gestion/empresas/eliminar/<int:id\_empresa>', methods=['POST'])

@login\_required

def eliminar\_empresa(id\_empresa):

if not current\_user.has\_role('super\_admin'):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_empresas'))

conn = conectar\_db()

cursor = conn.cursor()

try:

# Inactivar la empresa

cursor.execute("UPDATE empresas SET activo = 0 WHERE id\_empresa = ?", (id\_empresa,))

# Opcionalmente, inactivar a todos los usuarios asociados a esa empresa

cursor.execute("UPDATE usuarios SET activo = 0 WHERE id\_empresa = ?", (id\_empresa,))

# Considera qué hacer con los platos, repartidores, pedidos, etc. de esta empresa:

# Podrías inactivarlos también, o simplemente depender del filtro 'id\_empresa'

# que los hace invisibles para los admin\_empresa.

# Por simplicidad aquí, solo inactivamos la empresa y sus usuarios.

conn.commit()

flash(f"Empresa con ID {id\_empresa} marcada como inactiva y sus usuarios asociados inactivados.", "success")

except sqlite3.Error as e:

conn.rollback()

flash(f"Error al inactivar empresa: {e}", "danger")

finally:

conn.close()

return redirect(url\_for('gestion\_empresas'))

@app.route('/gestion/usuarios')

@login\_required

def gestion\_usuarios():

if not current\_user.has\_role('super\_admin'):

flash("No tienes permiso para acceder a esta página.", "danger")

return redirect(url\_for('index'))

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("""

SELECT u.id\_usuario, u.email, u.nombre, u.apellido, u.activo, u.primer\_login\_requerido,

r.nombre\_rol, e.nombre AS nombre\_empresa

FROM usuarios u

JOIN roles r ON u.id\_rol = r.id\_rol

LEFT JOIN empresas e ON u.id\_empresa = e.id\_empresa

ORDER BY u.apellido, u.nombre

""")

usuarios = cursor.fetchall()

conn.close()

return render\_template('gestion\_usuarios.html', usuarios=usuarios)

@app.route('/gestion/usuarios/agregar', methods=['GET', 'POST'])

@login\_required

def agregar\_usuario():

if not current\_user.has\_role('super\_admin'):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_usuarios'))

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("SELECT id\_rol, nombre\_rol FROM roles ORDER BY nombre\_rol")

roles = cursor.fetchall()

cursor.execute("SELECT id\_empresa, nombre FROM empresas WHERE activo = 1 ORDER BY nombre")

empresas = cursor.fetchall()

conn.close()

if request.method == 'POST':

email = request.form['email'].strip()

nombre = request.form['nombre'].strip()

apellido = request.form['apellido'].strip()

id\_rol = request.form['id\_rol']

id\_empresa = request.form.get('id\_empresa') # Puede ser None

password\_inicial = request.form['password\_inicial']

primer\_login\_requerido = 1 if 'primer\_login\_requerido' in request.form else 0

if not all([email, nombre, apellido, id\_rol, password\_inicial]):

flash("Todos los campos obligatorios deben ser completados.", "danger")

return render\_template('agregar\_usuario.html', roles=roles, empresas=empresas, request\_form=request.form)

if id\_empresa == '': # Convertir string vacío a None para la DB

id\_empresa = None

else:

id\_empresa = int(id\_empresa) # Asegurarse de que sea un entero

conn = conectar\_db()

cursor = conn.cursor()

try:

hashed\_password = generate\_password\_hash(password\_inicial, method='pbkdf2:sha256')

cursor.execute("""

INSERT INTO usuarios (email, password, nombre, apellido, id\_rol, id\_empresa, activo, primer\_login\_requerido)

VALUES (?, ?, ?, ?, ?, ?, 1, ?)

""", (email, hashed\_password, nombre, apellido, id\_rol, id\_empresa, primer\_login\_requerido))

conn.commit()

flash(f"Usuario '{email}' agregado con éxito. Contraseña inicial: {password\_inicial}", "success")

except sqlite3.IntegrityError:

flash(f"Ya existe un usuario con el email '{email}'.", "danger")

return render\_template('agregar\_usuario.html', roles=roles, empresas=empresas, request\_form=request.form)

except sqlite3.Error as e:

flash(f"Error al agregar usuario: {e}", "danger")

return render\_template('agregar\_usuario.html', roles=roles, empresas=empresas, request\_form=request.form)

finally:

conn.close()

return redirect(url\_for('gestion\_usuarios'))

return render\_template('agregar\_usuario.html', roles=roles, empresas=empresas, request\_form={})

@app.route('/gestion/usuarios/editar/<int:id\_usuario>', methods=['GET', 'POST'])

@login\_required

def editar\_usuario(id\_usuario):

if not current\_user.has\_role('super\_admin'):

flash("No tienes permiso para acceder a esta página.", "danger")

return redirect(url\_for('gestion\_usuarios'))

conn = conectar\_db()

cursor = conn.cursor()

cursor.execute("""

SELECT u.id\_usuario, u.email, u.nombre, u.apellido, u.activo, u.primer\_login\_requerido,

u.id\_rol, u.id\_empresa

FROM usuarios u

WHERE u.id\_usuario = ?

""", (id\_usuario,))

usuario = cursor.fetchone()

if not usuario:

flash("Usuario no encontrado.", "danger")

conn.close()

return redirect(url\_for('gestion\_usuarios'))

cursor.execute("SELECT id\_rol, nombre\_rol FROM roles ORDER BY nombre\_rol")

roles = cursor.fetchall()

cursor.execute("SELECT id\_empresa, nombre FROM empresas WHERE activo = 1 ORDER BY nombre")

empresas = cursor.fetchall()

conn.close()

if request.method == 'POST':

email = request.form['email'].strip()

nombre = request.form['nombre'].strip()

apellido = request.form['apellido'].strip()

id\_rol = request.form['id\_rol']

id\_empresa = request.form.get('id\_empresa')

activo = 1 if 'activo' in request.form else 0

primer\_login\_requerido = 1 if 'primer\_login\_requerido' in request.form else 0

nueva\_password = request.form.get('nueva\_password', '').strip()

if not all([email, nombre, apellido, id\_rol]):

flash("Todos los campos obligatorios deben ser completados.", "danger")

return render\_template('editar\_usuario.html', usuario=usuario, roles=roles, empresas=empresas, request\_form=request.form)

if id\_empresa == '': # Convertir string vacío a None para la DB

id\_empresa = None

else:

id\_empresa = int(id\_empresa) # Asegurarse de que sea un entero

conn = conectar\_db()

cursor = conn.cursor()

try:

update\_query = """

UPDATE usuarios SET email = ?, nombre = ?, apellido = ?, id\_rol = ?,

id\_empresa = ?, activo = ?, primer\_login\_requerido = ?

"""

update\_params = [email, nombre, apellido, id\_rol, id\_empresa, activo, primer\_login\_requerido]

if nueva\_password:

hashed\_password = generate\_password\_hash(nueva\_password, method='pbkdf2:sha256')

update\_query += ", password = ?"

update\_params.append(hashed\_password)

flash("Contraseña actualizada.", "info")

update\_query += " WHERE id\_usuario = ?"

update\_params.append(id\_usuario)

cursor.execute(update\_query, tuple(update\_params))

conn.commit()

flash(f"Usuario '{email}' actualizado con éxito.", "success")

except sqlite3.IntegrityError:

flash(f"Ya existe un usuario con el email '{email}'.", "danger")

return render\_template('editar\_usuario.html', usuario=usuario, roles=roles, empresas=empresas, request\_form=request.form)

except sqlite3.Error as e:

flash(f"Error al actualizar usuario: {e}", "danger")

return render\_template('editar\_usuario.html', usuario=usuario, roles=roles, empresas=empresas, request\_form=request.form)

finally:

conn.close()

return redirect(url\_for('gestion\_usuarios'))

return render\_template('editar\_usuario.html', usuario=usuario, roles=roles, empresas=empresas, request\_form={})

@app.route('/gestion/usuarios/eliminar/<int:id\_usuario>', methods=['POST'])

@login\_required

def eliminar\_usuario(id\_usuario):

if not current\_user.has\_role('super\_admin'):

flash("No tienes permiso para realizar esta acción.", "danger")

return redirect(url\_for('gestion\_usuarios'))

if id\_usuario == current\_user.id:

flash("No puedes eliminar tu propio usuario mientras estás conectado.", "danger")

return redirect(url\_for('gestion\_usuarios'))

conn = conectar\_db()

cursor = conn.cursor()

try:

cursor.execute("UPDATE usuarios SET activo = 0 WHERE id\_usuario = ?", (id\_usuario,))

conn.commit()

flash(f"Usuario con ID {id\_usuario} marcado como inactivo.", "success")

except sqlite3.Error as e:

flash(f"Error al inactivar usuario: {e}", "danger")

finally:

conn.close()

return redirect(url\_for('gestion\_usuarios'))

if \_\_name\_\_ == '\_\_main\_\_':

# --- SUGERENCIA: Descomenta las siguientes líneas si quieres forzar la recreación de la DB

# --- Esto es útil para desarrollo cuando se hacen cambios en las tablas.

# --- ADVERTENCIA: ¡Esto borrará todos tus datos actuales de la base de datos!

# if os.path.exists(DB\_NAME):

# os.remove(DB\_NAME)

# print(f"Base de datos '{DB\_NAME}' eliminada para recreación.")

init\_app()

app.run(debug=True)