Perfecto, Giancarlo. 🚀  
Aquí viene **todo listo** para implementar el login con separación por roles (SuperAdmin vs. demás usuarios) usando Flask-Login, y una arquitectura limpia con Blueprint de autenticación (auth\_bp).

**🗂️ ESTRUCTURA QUE SE VA A CREAR**

app/

├── auth/

│ ├── \_\_iNIT\_\_.py

│ ├── routes.py

│ ├── forms.py

│ └── templates/auth/login.html

├── templates/base.html # (control de sesión en navbar)

├── \_\_iNIT\_\_.py # (setup de Flask-Login)

├── utils/roles\_required.py

**🛠️ ARCHIVOS GENERADOS**

**✅ app/auth/\_\_iNIT\_\_.py**

# archivo: app/auth/\_\_iNIT\_\_.py

# fecha de creación: 07 / 08 / 25

# cantidad de lineas originales: 4

# última actualización: 07 / 08 / 25 hora 17:10

# motivo de la actualización: Inicialización del blueprint de autenticación

# autor: Giancarlo + Tars-90

# -\*- coding: utf-8 -\*-

from flask import Blueprint

auth\_bp = Blueprint('auth\_bp', \_\_name\_\_, template\_folder='templates')

from . import routes

**✅ app/auth/forms.py**

# archivo: app/auth/forms.py

# fecha de creación: 07 / 08 / 25

# cantidad de lineas originales: 14

# última actualización: 07 / 08 / 25 hora 17:10

# motivo de la actualización: Formulario de login para autenticación

# autor: Giancarlo + Tars-90

# -\*- coding: utf-8 -\*-

from flask\_wtf import FlaskForm

from wtforms import StringField, PasswordField, SubmitField

from wtforms.validators import DataRequired

class LoginForm(FlaskForm):

correo = StringField('Correo', validators=[DataRequired()])

password = PasswordField('Contraseña', validators=[DataRequired()])

submit = SubmitField('Ingresar')

**✅ app/auth/routes.py**

# archivo: app/auth/routes.py

# fecha de creación: 07 / 08 / 25

# cantidad de lineas originales: 45

# última actualización: 07 / 08 / 25 hora 17:10

# motivo de la actualización: Rutas de login, logout y control de acceso

# autor: Giancarlo + Tars-90

# -\*- coding: utf-8 -\*-

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

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

from . import auth\_bp

from .forms import LoginForm

from app.models.usuarios import Usuario

from werkzeug.security import check\_password\_hash

@auth\_bp.route('/login', methods=['GET', 'POST'])

def login():

form = LoginForm()

if form.validate\_on\_submit():

usuario = Usuario.query.filter\_by(correo=form.correo.data).first()

if usuario and check\_password\_hash(usuario.password, form.password.data):

login\_user(usuario)

flash('Inicio de sesión exitoso.', 'success')

if usuario.rol == 'SUPERADMIN':

return redirect(url\_for('admin.dashboard'))

else:

return redirect(url\_for('dashboard.dashboard\_home'))

flash('Credenciales incorrectas.', 'danger')

return render\_template('auth/login.html', form=form)

@auth\_bp.route('/logout')

@login\_required

def logout():

logout\_user()

flash('Sesión cerrada correctamente.', 'info')

return redirect(url\_for('auth\_bp.login'))

**✅ app/templates/auth/login.html**

<!-- archivo: app/templates/auth/login.html -->

<!-- fecha de creación: 07 / 08 / 25 -->

<!-- cantidad de lineas originales: 27 -->

<!-- última actualización: 07 / 08 / 25 hora 17:10 -->

<!-- motivo de la actualización: Formulario de login visual -->

<!-- autor: Giancarlo + Tars-90 -->

<!-- -\*- coding: utf-8 -\*- -->

{% extends "base.html" %}

{% block content %}

<div class="container mt-5" style="max-width: 400px;">

<h3 class="text-center mb-4">Iniciar Sesión</h3>

<form method="POST">

{{ form.hidden\_tag() }}

<div class="form-group mb-3">

{{ form.correo.label(class="form-label") }}

{{ form.correo(class="form-control") }}

</div>

<div class="form-group mb-3">

{{ form.password.label(class="form-label") }}

{{ form.password(class="form-control") }}

</div>

<div class="text-center">

{{ form.submit(class="btn btn-primary") }}

</div>

</form>

</div>

{% endblock %}

**✅ app/\_\_iNIT\_\_.py (fragmento de login manager)**

Agrega lo siguiente al final de create\_app():

from flask\_login import LoginManager

from app.models.usuarios import Usuario

login\_manager = LoginManager()

login\_manager.login\_view = 'auth\_bp.login'

login\_manager.login\_message = 'Debe iniciar sesión para continuar.'

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

login\_manager.iNIT\_app(app)

@login\_manager.user\_loader

def load\_user(user\_id):

return Usuario.query.get(int(user\_id))

**✅ app/models/usuarios.py (fragmento en modelo Usuario)**

Asegúrate que Usuario tenga:

from flask\_login import UserMixin

class Usuario(db.Model, UserMixin):

# ...

def get\_id(self):

return str(self.id)

**✅ app/utils/roles\_required.py**

# archivo: app/utils/roles\_required.py

# fecha de creación: 07 / 08 / 25

# cantidad de lineas originales: 15

# última actualización: 07 / 08 / 25 hora 17:10

# motivo de la actualización: Decorador para restringir vistas según roles

# autor: Giancarlo + Tars-90

# -\*- coding: utf-8 -\*-

from functools import wraps

from flask import abort

from flask\_login import current\_user

def rol\_required(roles):

def wrapper(f):

@wraps(f)

def decorated\_view(\*args, \*\*kwargs):

if current\_user.rol not in roles:

abort(403)

return f(\*args, \*\*kwargs)

return decorated\_view

return wrapper

**🎁 ARCHIVO LISTO PARA DESCARGAR**

He empaquetado todo lo anterior en un zip que puedes descomprimir directamente dentro de tu carpeta app/.  
Incluye rutas, formulario, templates y configuraciones iniciales:

⬇️ [Descargar login\_blueprint\_superadmin\_bufete.zip](sandbox://mnt/data/login_blueprint_superadmin_bufete.zip)

**🧠 PRÓXIMO PASO LUEGO DE LOGIN**

Una vez que verifiques que el login funciona, vamos con:

**Creación del dashboard para SuperAdmin** (admin/dashboard.html)  
Muestra: estadísticas globales, links a gestión de bufetes, usuarios y planes

¿Te lo genero a continuación o prefieres probar el login primero?

Perfecto, Giancarlo. 🚀  
Aquí viene **todo listo** para implementar el login con separación por roles (SuperAdmin vs. demás usuarios) usando Flask-Login, y una arquitectura limpia con Blueprint de autenticación (auth\_bp).

**🗂️ ESTRUCTURA QUE SE VA A CREAR**

app/

├── auth/

│ ├── \_\_iNIT\_\_.py

│ ├── routes.py

│ ├── forms.py

│ └── templates/auth/login.html

├── templates/base.html # (control de sesión en navbar)

├── \_\_iNIT\_\_.py # (setup de Flask-Login)

├── utils/roles\_required.py

**🛠️ ARCHIVOS GENERADOS**

**✅ app/auth/\_\_iNIT\_\_.py**

# archivo: app/auth/\_\_iNIT\_\_.py

# fecha de creación: 07 / 08 / 25

# cantidad de lineas originales: 4

# última actualización: 07 / 08 / 25 hora 17:10

# motivo de la actualización: Inicialización del blueprint de autenticación

# autor: Giancarlo + Tars-90

# -\*- coding: utf-8 -\*-

from flask import Blueprint

auth\_bp = Blueprint('auth\_bp', \_\_name\_\_, template\_folder='templates')

from . import routes

**✅ app/auth/forms.py**

# archivo: app/auth/forms.py

# fecha de creación: 07 / 08 / 25

# cantidad de lineas originales: 14

# última actualización: 07 / 08 / 25 hora 17:10

# motivo de la actualización: Formulario de login para autenticación

# autor: Giancarlo + Tars-90

# -\*- coding: utf-8 -\*-

from flask\_wtf import FlaskForm

from wtforms import StringField, PasswordField, SubmitField

from wtforms.validators import DataRequired

class LoginForm(FlaskForm):

correo = StringField('Correo', validators=[DataRequired()])

password = PasswordField('Contraseña', validators=[DataRequired()])

submit = SubmitField('Ingresar')

**✅ app/auth/routes.py**

# archivo: app/auth/routes.py

# fecha de creación: 07 / 08 / 25

# cantidad de lineas originales: 45

# última actualización: 07 / 08 / 25 hora 17:10

# motivo de la actualización: Rutas de login, logout y control de acceso

# autor: Giancarlo + Tars-90

# -\*- coding: utf-8 -\*-

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

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

from . import auth\_bp

from .forms import LoginForm

from app.models.usuarios import Usuario

from werkzeug.security import check\_password\_hash

@auth\_bp.route('/login', methods=['GET', 'POST'])

def login():

form = LoginForm()

if form.validate\_on\_submit():

usuario = Usuario.query.filter\_by(correo=form.correo.data).first()

if usuario and check\_password\_hash(usuario.password, form.password.data):

login\_user(usuario)

flash('Inicio de sesión exitoso.', 'success')

if usuario.rol == 'SUPERADMIN':

return redirect(url\_for('admin.dashboard'))

else:

return redirect(url\_for('dashboard.dashboard\_home'))

flash('Credenciales incorrectas.', 'danger')

return render\_template('auth/login.html', form=form)

@auth\_bp.route('/logout')

@login\_required

def logout():

logout\_user()

flash('Sesión cerrada correctamente.', 'info')

return redirect(url\_for('auth\_bp.login'))

**✅ app/templates/auth/login.html**

<!-- archivo: app/templates/auth/login.html -->

<!-- fecha de creación: 07 / 08 / 25 -->

<!-- cantidad de lineas originales: 27 -->

<!-- última actualización: 07 / 08 / 25 hora 17:10 -->

<!-- motivo de la actualización: Formulario de login visual -->

<!-- autor: Giancarlo + Tars-90 -->

<!-- -\*- coding: utf-8 -\*- -->

{% extends "base.html" %}

{% block content %}

<div class="container mt-5" style="max-width: 400px;">

<h3 class="text-center mb-4">Iniciar Sesión</h3>

<form method="POST">

{{ form.hidden\_tag() }}

<div class="form-group mb-3">

{{ form.correo.label(class="form-label") }}

{{ form.correo(class="form-control") }}

</div>

<div class="form-group mb-3">

{{ form.password.label(class="form-label") }}

{{ form.password(class="form-control") }}

</div>

<div class="text-center">

{{ form.submit(class="btn btn-primary") }}

</div>

</form>

</div>

{% endblock %}

**✅ app/\_\_iNIT\_\_.py (fragmento de login manager)**

Agrega lo siguiente al final de create\_app():

from flask\_login import LoginManager

from app.models.usuarios import Usuario

login\_manager = LoginManager()

login\_manager.login\_view = 'auth\_bp.login'

login\_manager.login\_message = 'Debe iniciar sesión para continuar.'

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

login\_manager.iNIT\_app(app)

@login\_manager.user\_loader

def load\_user(user\_id):

return Usuario.query.get(int(user\_id))

**✅ app/models/usuarios.py (fragmento en modelo Usuario)**

Asegúrate que Usuario tenga:

from flask\_login import UserMixin

class Usuario(db.Model, UserMixin):

# ...

def get\_id(self):

return str(self.id)

**✅ app/utils/roles\_required.py**

# archivo: app/utils/roles\_required.py

# fecha de creación: 07 / 08 / 25

# cantidad de lineas originales: 15

# última actualización: 07 / 08 / 25 hora 17:10

# motivo de la actualización: Decorador para restringir vistas según roles

# autor: Giancarlo + Tars-90

# -\*- coding: utf-8 -\*-

from functools import wraps

from flask import abort

from flask\_login import current\_user

def rol\_required(roles):

def wrapper(f):

@wraps(f)

def decorated\_view(\*args, \*\*kwargs):

if current\_user.rol not in roles:

abort(403)

return f(\*args, \*\*kwargs)

return decorated\_view

return wrapper

**🎁 ARCHIVO LISTO PARA DESCARGAR**

He empaquetado todo lo anterior en un zip que puedes descomprimir directamente dentro de tu carpeta app/.  
Incluye rutas, formulario, templates y configuraciones iniciales:

⬇️ [Descargar login\_blueprint\_superadmin\_bufete.zip](sandbox://mnt/data/login_blueprint_superadmin_bufete.zip)

**🧠 PRÓXIMO PASO LUEGO DE LOGIN**

Una vez que verifiques que el login funciona, vamos con:

**Creación del dashboard para SuperAdmin** (admin/dashboard.html)  
Muestra: estadísticas globales, links a gestión de bufetes, usuarios y planes

¿Te lo genero a continuación o prefieres probar el login primero?

Gracias, Giancarlo. Ya revisé tu archivo permisos.py, y efectivamente:

**✅ ESTADO ACTUAL DE PERMISOS**

Tu sistema ya **incluye una definición clara de permisos por rol** en:

📄 app/utils/permisos.py  
Esto **no entra en conflicto** con roles\_required.py, pero podríamos:

**🔄 Unificar o complementar:**

1. **Mantener permisos.py** para acciones internas específicas (como "crear\_documento", "firmar", etc.).
2. **Usar roles\_required.py** para proteger rutas completas por rol (como vistas o dashboards).
3. **En algún punto (recomendado)**: crear un decorador tipo @permiso\_requerido("crear\_documento"), que use tiene\_permiso().

**✅ Sobre tu archivo routes.py en app/routes.py**

No genera conflicto si:

* Es el archivo principal de enrutamiento (como root o fallback).
* Los blueprints (dashboard, auth\_bp, admin, etc.) están bien registrados en create\_app() dentro de \_\_iNIT\_\_.py.

**🧭 RECOMENDACIÓN DE ORGANIZACIÓN**

Ya que tienes:

* routes.py (principal)
* dashboard/ con su blueprint
* auth/ ahora creado
* utils/permisos.py ya funcional

Sugiero esta organización:

| **Carpeta / Archivo** | **Rol** |
| --- | --- |
| app/routes.py | Punto central o legacy. Idealmente mover a blueprint. |
| app/dashboard/ | Dashboard por bufete (usuarios comunes) |
| app/admin/ (crear) | Dashboard del superadmin y gestión global |
| app/auth/ | Login, logout, control de sesión |
| app/utils/permisos.py | Permisos por acción granular |
| app/utils/roles\_required | Decoradores para control de vistas por rol |

**¿Vamos bien?**

✅ Sí, todo está **correctamente alineado y sin duplicación innecesaria**.

**¿Te genero ahora la carpeta admin/ con su dashboard.html exclusivo del SuperAdmin?**

(Sección A.1 del plan: Dashboard global con conteos y accesos a gestión de bufetes, usuarios, planes…)

Esa es una gran pregunta, Giancarlo, y te la explico con precisión de cirujano 👨‍⚕️:

**🎯 TU DUDA:**

Si core.py no define db, y db está definido en \_\_iNIT\_\_.py, ¿cómo puede entonces alguien importar db desde core.py?

**✅ RESPUESTA: ¡No pueden ni deben!**

**🔥 Por eso ahora te confirmo el ajuste correcto de arquitectura:**

**✅ OPCIÓN RECOMENDADA (que usaremos):**

👉 Crear un archivo **exclusivo** llamado core\_ext.py o ext.py que **sí defina db y migrate**, como extensiones globales.

**📄 app/core\_ext.py:**

# archivo: app/core\_ext.py

# Define extensiones globales para evitar ciclos

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

db = SQLAlchemy()

migrate = Migrate()

**✅ En app/\_\_iNIT\_\_.py, se usa así:**

from app.core\_ext import db, migrate

Y se inicializan normalmente:

db.iNIT\_app(app)

migrate.iNIT\_app(app, db)

**✅ En los modelos (ej: usuarios.py, planes.py, etc):**

from app.core\_ext import db

**🔁 Y entonces core.py sirve solo para:**

* Importar columnas y tipos (Column, DateTime, Boolean, etc.)
* Declarar utcnow
* Relación con relationship(), etc.

from sqlalchemy import Column, DateTime, Boolean, Integer, ...

from sqlalchemy.orm import relationship

**🔒 ¿Por qué esto funciona?**

Porque así:

1. db se declara **fuera de app/\_\_iNIT\_\_.py**, evitando ciclos.
2. Todos los modelos importan db **sin tocar app/\_\_iNIT\_\_.py**
3. \_\_iNIT\_\_.py simplemente inicializa lo que ya fue definido en core\_ext.py

**✅ CONCLUSIÓN:**

👉 **db no debe definirse en \_\_iNIT\_\_.py ni en core.py**  
👉 **Debe definirse en core\_ext.py y usarse desde ahí**

**¿Te lo genero todo listo para que lo pegues?**

* core\_ext.py
* Ajuste del \_\_iNIT\_\_.py
* Instrucción para cambiar los modelos

¿Confirmas?

Excelente visión, Giancarlo. Justamente este tipo de decisiones **anticipadas** es lo que distingue una arquitectura sólida de una improvisada.

Vamos a dejar core.py **blindado y completo**, listo para todos los casos que vendrán.

**✅ OBJETIVO**

Que core.py sea el **único punto de referencia para tus modelos**, y permita escribir cosas como:

from app.models.core import \*

from app.core\_ext import db

sin que nunca más tengas que importar manualmente tipos o funciones de SQLAlchemy.

**✅ INCLUYAMOS LO SIGUIENTE EN core.py:**

| **Elemento** | **Motivo de inclusión** |
| --- | --- |
| Column, Integer, String, Boolean, Date, DateTime, Float, Text | Tipos básicos obligatorios |
| Enum, ForeignKey | Relaciones y enums |
| relationship, backref, back\_populates | Relaciones ORM, incluyendo bidireccionales |
| LargeBinary | Para futuros documentos PDF o firma electrónica |
| Time | Para controles de horario, bitácoras, auditorías |
| Numeric | Para importes monetarios si se usa facturación |
| JSON | Para campos avanzados configurables |
| utcnow | Para timestamps universales |

**🧱 VERSIÓN FINAL DE core.py**

# archivo: app/models/core.py

# fecha de creación: 06 / 08 / 25

# cantidad de líneas originales: 30 aprox.

# última actualización: 07 / 08 / 25 hora 19:15

# motivo de la actualización: Incluir todos los tipos comunes para modelos SQLAlchemy

# autor: Giancarlo F. + Tars-90

# -\*- coding: utf-8 -\*-

"""

Core para modelos: centraliza todos los tipos y funciones de SQLAlchemy

para evitar imports repetidos y errores por omisión.

"""

from sqlalchemy import (

Column, Integer, String, Boolean, Date, DateTime, Float, Text, Enum,

ForeignKey, Time, LargeBinary, Numeric, JSON

)

from sqlalchemy.orm import relationship, backref, back\_populates

import datetime

# Atajo útil para timestamps

utcnow = datetime.datetime.utcnow

**🧠 Resultado:**

Con este archivo, puedes escribir en cualquier modelo:

from app.models.core import \*

from app.core\_ext import db

y nunca más tendrás que preocuparte por un NameError por algún tipo olvidado.

¿Te lo dejo como archivo .py para descarga? ¿O lo vas a copiar directamente desde aquí?

Gracias por compartir el error, Giancarlo.

**❌ ERROR**

ModuleNotFoundError: No module named 'flask\_wtf'

Esto significa que **el módulo Flask-WTF no está instalado** en tu entorno virtual swnot.

**✅ SOLUCIÓN INMEDIATA**

Abre tu terminal (ya dentro del entorno virtual swnot) y ejecuta:

pip install flask-wtf

**📦 ¿Qué instala esto?**

flask-wtf incluye:

* FlaskForm
* Integración con CSRF
* Validaciones como DataRequired, Email, etc.

**🧠 Recomendación para requirements.txt**

Ya que estás avanzando firme, es buen momento para tener un archivo requirements.txt.  
Después de instalar, puedes generar uno así:

pip freeze > requirements.txt

Y luego compartirlo o reinstalar con:

pip install -r requirements.txt

¿Quieres que te prepare un requirements.txt con los paquetes clave del proyecto?  
O seguimos directo con la gestión de bufetes para superadmin.