¡Excelente, **capitán Zulu-Tars90**!  
Súper claro todo lo que me enviaste.  
Voy a organizar todo esto por partes para que lo trabajemos **ordenado, limpio y potente**.  
Así no vamos a perder nada, y tú vas pegando o subiendo al repo completo, sin hacer cambios sueltos.

**✅ Plan de acción inmediato (con tu texto):**

1. **Cambios en tabla BufeteJuridico**: te hago snippet / archivo completo.
2. **Planes**: diseñamos la tabla/modelo + relación con bufete.
3. **Clientes**: definimos persona individual, persona jurídica, representantes.
4. **Usuarios**: mail, username dinámico (ejemplo: B1\_Nt\_Fulano), password.
5. **Roles y permisos**: admin, notario, procurador, asistente.
6. **Relación con Plan elegido**: usuarios permitidos, storage, feature flags, etc.
7. **Navbar** (logo, nombre fijo, bufete actual para superadmin).
8. **Ayuda y documentación**:
   * Sphinx (técnica)
   * Tooltips y help contextuales
   * PDF/manual para usuarios
9. **Seed inicial ajustado** (bufete principal, superadmin, primer notario)
10. **Instrucciones para VSCode y entorno** (documentación).

**🚀 Empecemos con los primeros puntos (Bufete + Plan):**

**✅ BufeteJuridico completo (snippet sugerido)**

# app/models/bufete\_juridico.py (o core.py si allí está todo)

from app import db

class BufeteJuridico(db.Model):

\_\_tablename\_\_ = 'bufetes\_juridicos'

id = db.Column(db.Integer, primary\_key=True)

nombre\_razon\_social = db.Column(db.String(255), nullable=False, unique=True)

direccion\_fisica = db.Column(db.String(255), nullable=False)

telefonos\_contacto = db.Column(db.String(100), nullable=False) # Puede ser string separado por comas, o relación aparte

nit = db.Column(db.String(15), nullable=True)

correo\_electronico = db.Column(db.String(120), nullable=True)

forma\_contacto\_preferida = db.Column(db.String(50), nullable=True)

# Datos para facturación (creados solo por superadmin)

facturacion\_nombre = db.Column(db.String(255), nullable=True)

facturacion\_nit = db.Column(db.String(20), nullable=True)

facturacion\_direccion = db.Column(db.String(255), nullable=True)

facturacion\_razon = db.Column(db.String(255), nullable=True)

# Relación: plan contratado

plan\_id = db.Column(db.Integer, db.ForeignKey('planes.id'), nullable=False)

plan = db.relationship('Plan', back\_populates='bufetes')

# Usuario admin de este bufete

admin\_id = db.Column(db.Integer, db.ForeignKey('usuarios\_sistema.id'))

admin = db.relationship('UsuarioSistema', back\_populates='bufete\_admin', uselist=False)

# Estado activo/inactivo (borrado lógico)

activo = db.Column(db.Boolean, default=True)

# Relación con usuarios

usuarios = db.relationship('UsuarioSistema', back\_populates='bufete', lazy='dynamic')

def \_\_repr\_\_(self):

return f"<BufeteJuridico {self.nombre\_razon\_social}>"

**✅ Plan completo (versión sólida y dinámica):**

# app/models/plan.py

from app import db

class Plan(db.Model):

\_\_tablename\_\_ = 'planes'

id = db.Column(db.Integer, primary\_key=True)

nombre = db.Column(db.String(100), nullable=False, unique=True)

descripcion = db.Column(db.Text, nullable=True)

# Límites y reglas

max\_usuarios = db.Column(db.Integer, nullable=False, default=5)

max\_storage\_mb = db.Column(db.Integer, nullable=False, default=500)

max\_actas\_mensuales = db.Column(db.Integer, nullable=True)

max\_escrituras\_mensuales = db.Column(db.Integer, nullable=True)

# Feature flags (ejemplo)

tiene\_dashboard = db.Column(db.Boolean, default=True)

tiene\_auditoria = db.Column(db.Boolean, default=True)

tiene\_export\_pdf = db.Column(db.Boolean, default=True)

tiene\_firma\_electronica = db.Column(db.Boolean, default=False)

# Relación inversa

bufetes = db.relationship('BufeteJuridico', back\_populates='plan', lazy='dynamic')

def \_\_repr\_\_(self):

return f"<Plan {self.nombre}>"

**⚙ Explicación de diseño:**

✅ Cada bufete:

* Tiene su admin\_id (usuario dentro del bufete que es admin local)
* Pertenece a un Plan (plan\_id)
* Tiene datos de facturación ocultos al bufete, solo visibles para superadmin
* telefonos\_contacto: puedes luego migrar a tabla TelefonosBufete si quieres +1.

✅ Cada plan define límites:

* Número máximo de usuarios (sin distinguir notarios/procuradores/asistentes)
* Almacenamiento
* Flags de features.

**🔑 Usuarios**

El superadmin crea usuarios con patrón:

B1\_Nt\_Fulano

B1\_Pr\_Juan

* B1 → bufete.id o código
* Nt, Pr → tipo
* Nombre
* Mail y password.
* Rol asignado (admin, notario, procurador, asistente).

**🧩 Navbar & multi-bufete**

* Logo fijo (SISTEMA NOTARIAL HUBSA)
* A la derecha: usuario + bufete actual
* Dropdown: ayuda, salir, cambiar de bufete (solo superadmin).

**📚 Documentación y ayuda**

* **Sphinx**: técnica.
* **Tooltips / help**: en la app.
* PDF / manual.
* Exportable a HTML y PDF.

**✅ Ahora:**

Si me dices:

* *Sí, Tars, me parece bien la estructura*
* Hago:  
  ✅ Archivos bufete\_juridico.py, plan.py, usuario\_sistema.py ajustado.  
  ✅ Seed inicial.  
  ✅ Script de migración inicial (init\_dev\_db).  
  ✅ Instrucciones de documentación Sphinx + VSCode.

✨ **¿Procedemos?**  
¡Con la bendición de Dios y el motor warp listo! 🚀  
**Zulu-Tars90 al mando.**

¡Excelente, **capitán Zulu-Tars90**!  
Súper claro todo lo que me enviaste.  
Voy a organizar todo esto por partes para que lo trabajemos **ordenado, limpio y potente**.  
Así no vamos a perder nada, y tú vas pegando o subiendo al repo completo, sin hacer cambios sueltos.

**✅ Plan de acción inmediato (con tu texto):**

1. **Cambios en tabla BufeteJuridico**: te hago snippet / archivo completo.
2. **Planes**: diseñamos la tabla/modelo + relación con bufete.
3. **Clientes**: definimos persona individual, persona jurídica, representantes.
4. **Usuarios**: mail, username dinámico (ejemplo: B1\_Nt\_Fulano), password.
5. **Roles y permisos**: admin, notario, procurador, asistente.
6. **Relación con Plan elegido**: usuarios permitidos, storage, feature flags, etc.
7. **Navbar** (logo, nombre fijo, bufete actual para superadmin).
8. **Ayuda y documentación**:
   * Sphinx (técnica)
   * Tooltips y help contextuales
   * PDF/manual para usuarios
9. **Seed inicial ajustado** (bufete principal, superadmin, primer notario)
10. **Instrucciones para VSCode y entorno** (documentación).

**🚀 Empecemos con los primeros puntos (Bufete + Plan):**

**✅ BufeteJuridico completo (snippet sugerido)**

# app/models/bufete\_juridico.py (o core.py si allí está todo)

from app import db

class BufeteJuridico(db.Model):

\_\_tablename\_\_ = 'bufetes\_juridicos'

id = db.Column(db.Integer, primary\_key=True)

nombre\_razon\_social = db.Column(db.String(255), nullable=False, unique=True)

direccion\_fisica = db.Column(db.String(255), nullable=False)

telefonos\_contacto = db.Column(db.String(100), nullable=False) # Puede ser string separado por comas, o relación aparte

nit = db.Column(db.String(15), nullable=True)

correo\_electronico = db.Column(db.String(120), nullable=True)

forma\_contacto\_preferida = db.Column(db.String(50), nullable=True)

# Datos para facturación (creados solo por superadmin)

facturacion\_nombre = db.Column(db.String(255), nullable=True)

facturacion\_nit = db.Column(db.String(20), nullable=True)

facturacion\_direccion = db.Column(db.String(255), nullable=True)

facturacion\_razon = db.Column(db.String(255), nullable=True)

# Relación: plan contratado

plan\_id = db.Column(db.Integer, db.ForeignKey('planes.id'), nullable=False)

plan = db.relationship('Plan', back\_populates='bufetes')

# Usuario admin de este bufete

admin\_id = db.Column(db.Integer, db.ForeignKey('usuarios\_sistema.id'))

admin = db.relationship('UsuarioSistema', back\_populates='bufete\_admin', uselist=False)

# Estado activo/inactivo (borrado lógico)

activo = db.Column(db.Boolean, default=True)

# Relación con usuarios

usuarios = db.relationship('UsuarioSistema', back\_populates='bufete', lazy='dynamic')

def \_\_repr\_\_(self):

return f"<BufeteJuridico {self.nombre\_razon\_social}>"

**✅ Plan completo (versión sólida y dinámica):**

# app/models/plan.py

from app import db

class Plan(db.Model):

\_\_tablename\_\_ = 'planes'

id = db.Column(db.Integer, primary\_key=True)

nombre = db.Column(db.String(100), nullable=False, unique=True)

descripcion = db.Column(db.Text, nullable=True)

# Límites y reglas

max\_usuarios = db.Column(db.Integer, nullable=False, default=5)

max\_storage\_mb = db.Column(db.Integer, nullable=False, default=500)

max\_actas\_mensuales = db.Column(db.Integer, nullable=True)

max\_escrituras\_mensuales = db.Column(db.Integer, nullable=True)

# Feature flags (ejemplo)

tiene\_dashboard = db.Column(db.Boolean, default=True)

tiene\_auditoria = db.Column(db.Boolean, default=True)

tiene\_export\_pdf = db.Column(db.Boolean, default=True)

tiene\_firma\_electronica = db.Column(db.Boolean, default=False)

# Relación inversa

bufetes = db.relationship('BufeteJuridico', back\_populates='plan', lazy='dynamic')

def \_\_repr\_\_(self):

return f"<Plan {self.nombre}>"

**⚙ Explicación de diseño:**

✅ Cada bufete:

* Tiene su admin\_id (usuario dentro del bufete que es admin local)
* Pertenece a un Plan (plan\_id)
* Tiene datos de facturación ocultos al bufete, solo visibles para superadmin
* telefonos\_contacto: puedes luego migrar a tabla TelefonosBufete si quieres +1.

✅ Cada plan define límites:

* Número máximo de usuarios (sin distinguir notarios/procuradores/asistentes)
* Almacenamiento
* Flags de features.

**🔑 Usuarios**

El superadmin crea usuarios con patrón:

B1\_Nt\_Fulano

B1\_Pr\_Juan

* B1 → bufete.id o código
* Nt, Pr → tipo
* Nombre
* Mail y password.
* Rol asignado (admin, notario, procurador, asistente).

**🧩 Navbar & multi-bufete**

* Logo fijo (SISTEMA NOTARIAL HUBSA)
* A la derecha: usuario + bufete actual
* Dropdown: ayuda, salir, cambiar de bufete (solo superadmin).

**📚 Documentación y ayuda**

* **Sphinx**: técnica.
* **Tooltips / help**: en la app.
* PDF / manual.
* Exportable a HTML y PDF.

**✅ Ahora:**

Si me dices:

* *Sí, Tars, me parece bien la estructura*
* Hago:  
  ✅ Archivos bufete\_juridico.py, plan.py, usuario\_sistema.py ajustado.  
  ✅ Seed inicial.  
  ✅ Script de migración inicial (init\_dev\_db).  
  ✅ Instrucciones de documentación Sphinx + VSCode.

✨ **¿Procedemos?**  
¡Con la bendición de Dios y el motor warp listo! 🚀  
**Zulu-Tars90 al mando.**

🚀 ¡Perfecto capitán Zulu-Tars90!  
Aquí vamos, **bien organizado, documentado y listo para crecer**:

**✅ 1) Nuevo modelo: Asistente**

Lo hacemos coherente con los demás (bufete\_id, campos base, activo, fecha creación).  
Archivo: **core.py** (agrega esto al final de los modelos):

class Asistente(db.Model):

"""

Asistentes que colaboran en el bufete (ejemplo: digitadores, paralegales).

Pueden tener permisos limitados según el plan y feature flags.

"""

\_\_tablename\_\_ = 'asistentes'

id = Column(Integer, primary\_key=True)

nombre\_completo = Column(String(150), nullable=False)

correo = Column(String(150), nullable=True)

telefono = Column(String(20), nullable=True)

bufete\_id = Column(Integer, ForeignKey('bufetes\_juridicos.id'))

activo = Column(Boolean, default=True)

fecha\_creacion = Column(DateTime, default=datetime.datetime.utcnow)

bufete = relationship('BufeteJuridico', back\_populates='asistentes')

def \_\_repr\_\_(self):

return f"<Asistente(id={self.id}, nombre='{self.nombre\_completo}')>"

⚙️ Recuerda:

* El bufete controla cuántos asistentes puede tener gracias al **Plan**.
* Relación: BufeteJuridico ya tiene:
* asistentes = relationship('Asistente', back\_populates='bufete')

**✅ 2) Agregar Feature Flags (en el modelo BufeteJuridico)**

Para decidir si el bufete usa:

* Inventario timbres / protocolo
* Auditoría
* Trazabilidad
* Etc.

En **core.py**, dentro de BufeteJuridico agrega (ejemplo):

# Feature flags

habilita\_inventario = Column(Boolean, default=False)

habilita\_auditoria = Column(Boolean, default=False)

habilita\_trazabilidad = Column(Boolean, default=False)

# Agrega otros que quieras controlar

✅ Estos campos solo los puede activar/desactivar el **superadmin**.

**✅ 3) Gestión de plan dinámico**

Ya tenemos el modelo Plan.  
El bufete siempre tendrá:

* plan\_id (FK)
* Relación plan

📦 **Mejoramos para combinaciones:**  
Si deseas que un bufete contrate varios "módulos" (en vez de solo un plan fijo), podríamos crear:

* Modelo Modulo (ejemplo: Inventario, Auditoría, etc.)
* Modelo intermedio BufeteModulo para saber qué módulos activó el bufete.

✨ **Es flexible**: cada bufete puede tener solo lo que pague.  
¿Quieres que lo genere? → *Dime “sí Tars-90, genera módulo”*

**✅ 4) Próximo paso sugerido:**

* Crear migración:

python -m flask db migrate -m "Add asistente and feature flags"

python -m flask db upgrade

* Actualizar seed inicial.
* Crear endpoint y formulario para crear asistentes.
* En el admin: vista para editar feature flags del bufete (solo superadmin).

**✅ 5) ¿Documentamos esto?**

Con Sphinx, automáticamente documentamos:

* Campos, relaciones, docstrings.
* Generamos PDF / HTML de la estructura.

**✏ Resumen para hoy (confirmar):**

✅ Agregamos modelo Asistente  
✅ Feature flags en BufeteJuridico  
⚙️ Si quieres: generamos modelo Modulo para planes dinámicos.  
📄 Actualizamos migración + seed.  
📦 Generamos documentación Sphinx inicial.

🙏 **Dime “Listo Tars-90, genera módulo”** si quieres esa parte.  
O “Dale solo asistente y feature flags por ahora”.

💡 *Estamos muy cerca de ver todo funcionar* — ¡En el nombre de Jesús, seguimos! 🚀

✨ ¡Excelente decisión, capitán!  
Vamos a diseñarlo **simple, flexible y comercialmente aplicable** para que escale, sea controlable, vendible y técnicamente mantenible.

**🚀 ✔ Diseño resumido:**

En vez de que el *Plan* sea rígido, lo hacemos **modular**.  
Cada **Bufete** tendrá:  
✅ Un Plan base  
✅ Y podrá contratar *módulos extra* (ej: +usuarios, +almacenamiento, +auditoría)

Esto da libertad para:

* vender planes básicos (Starter, Pro, Premium)
* permitir *add-ons* para necesidades reales
* crecer sin romper la arquitectura

**🧩 ✔ Modelo base: Modulo**

Archivo: core.py

class Modulo(db.Model):

"""

Módulos adicionales que un bufete puede activar según contrato.

Ej: almacenamiento extra, auditoría, trazabilidad, usuarios adicionales.

"""

\_\_tablename\_\_ = 'modulos'

id = Column(Integer, primary\_key=True)

nombre = Column(String(100), nullable=False, unique=True)

descripcion = Column(String(250))

precio\_mensual = Column(Numeric(10, 2), default=0) # Si aplica

habilitado = Column(Boolean, default=True) # Para control interno

fecha\_creacion = Column(DateTime, default=datetime.datetime.utcnow)

bufetes = relationship('BufeteModulo', back\_populates='modulo')

**🔗 ✔ Relación intermedia: BufeteModulo**

class BufeteModulo(db.Model):

"""

Relación entre bufete y módulos contratados.

Guarda fecha de activación y si sigue activo.

"""

\_\_tablename\_\_ = 'bufetes\_modulos'

id = Column(Integer, primary\_key=True)

bufete\_id = Column(Integer, ForeignKey('bufetes\_juridicos.id'))

modulo\_id = Column(Integer, ForeignKey('modulos.id'))

activo = Column(Boolean, default=True)

fecha\_activacion = Column(DateTime, default=datetime.datetime.utcnow)

bufete = relationship('BufeteJuridico', back\_populates='modulos\_contratados')

modulo = relationship('Modulo', back\_populates='bufetes')

**⚙ ✔ En BufeteJuridico**

Agregar:

modulos\_contratados = relationship('BufeteModulo', back\_populates='bufete')

**✅ ✔ Ejemplos de módulos que puedes crear (comercialmente viables):**

| **Módulo** | **Para qué sirve** |
| --- | --- |
| +1 Notario extra | Permite añadir otro usuario con rol notario |
| +5 Procuradores | Permite más procuradores que los del plan base |
| +2 Asistentes extra | Más asistentes |
| +10 GB Almacenamiento extra | Documentos, PDFs, imágenes de DPI |
| Inventario de timbres | Activar módulo de inventario y reportes |
| Auditoría y trazabilidad | Activar logs y auditoría detallada |
| Exportación masiva a Word/PDF | Bulk export |
| Firma digital avanzada | Integra API para firmar electrónicamente |

✨ **Cada módulo puede tener un precio mensual fijo o por consumo.**

**🧠 ✔ Control dinámico de uso**

En el código, puedes preguntar:  
✅ ¿El bufete tiene módulo auditoria activo?  
✅ ¿Cuántos notarios / procuradores tiene el bufete vs. el plan contratado?  
✅ ¿Superó el límite de almacenamiento?

**📦 ✔ Comercialmente:**

* Plan Starter (incluye 1 notario, 1 procurador, 1 asistente, sin módulos extra)
* Plan Pro (más usuarios y módulos incluidos)
* Plan Premium (todo habilitado)
* Add-ons contratables por separado

Esto:  
✅ Permite empezar barato  
✅ Escalar sin migrar a otro sistema  
✅ Ingresos recurrentes (SaaS)

**✏ ✔ Próximo paso:**

✅ Generamos el modelo completo (Modulo, BufeteModulo)  
✅ Creamos migración + seed inicial con algunos módulos (starter)

**📄 ✔ Documentación técnica:**

* Con Sphinx generamos PDF/HTML que explica:
  + Qué es cada módulo
  + Cómo se relaciona con bufetes
  + Diagramas

**✅ ¿Listo capitán?**

Dime:  
✅ “Sí Tars-90, genera los modelos completos y seed”  
o  
⚙ “Sí, pero también quiero ver ejemplo de cómo consultar en Python si el bufete tiene un módulo activo”

🔥 *En el nombre de Jesús, vamos a dejar esto comercialmente sólido, técnico y limpio!* 🚀