

¡Excelente iniciativa! Para dar vida al "**Nexo Sinérgico**" y soportar la arquitectura de "**Proyectos Híbridos**" que definimos (donde conviven científicos y creativos), necesitamos una base de datos robusta.

No haremos una tabla simple de usuarios. Diseñaremos un esquema **RBAC (Role-Based Access Control)** escalable. Esto es vital porque, como vimos, un "Académico" podría necesitar permisos diferentes a un "Operador" dentro del mismo ecosistema.

Además, dado que mencionas **JWS (JSON Web Signature)** para la autenticación, he incluido un campo de **token_version** para permitir la invalidación de tokens (seguridad crítica) y una tabla de auditoría básica.

Diagrama Conceptual

El Esquema SQL (PostgreSQL Dialect)

Este script crea las tablas, las relaciones y pre-carga los roles que solicitaste.

SQL

```
-- 1. Tabla de Roles (Categorías definidas)
-- Separamos los roles en una tabla para normalizar y escalar fácilmente.
CREATE TABLE roles (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL,
    description TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);

-- 2. Tabla de Usuarios (La identidad digital)
-- Usamos UUID para IDs más seguros y distribuidos.
-- IMPORTANTE: Nunca guardamos la contraseña plana, solo el HASH.
CREATE EXTENSION IF NOT EXISTS "uuid-ossp"; -- Asegurar extensión para UUID

CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    full_name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL, -- Aquí se guarda el hash (ej. Bcrypt/Argon2)
    is_active BOOLEAN DEFAULT TRUE, -- Para banear usuarios sin borrar datos

    -- Seguridad para JWS/JWT:
    -- Si el usuario cambia password o es hackeado, incrementamos esto
    -- y los tokens viejos (con versión anterior) quedan inválidos.
    token_version INT DEFAULT 1,
```

```
last_login_at TIMESTAMP WITH TIME ZONE,  
created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,  
updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP  
);
```

-- 3. Tabla Intermedia Usuarios_Roles (Many-to-Many)

-- Permite que un usuario sea "Académico" y "Colaborador" a la vez.

```
CREATE TABLE user_roles (
```

```
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,  
    role_id INT REFERENCES roles(id) ON DELETE CASCADE,  
    assigned_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,  
    PRIMARY KEY (user_id, role_id)
```

```
);
```

-- 4. Tabla de Sesiones / Refresh Tokens (Opcional pero recomendada para JWS)

-- JWS suele ser stateless (no se guarda), pero para "cerrar sesión" real

-- o mantener sesiones largas ("Recordarme"), necesitamos Refresh Tokens.

```
CREATE TABLE refresh_tokens (
```

```
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),  
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,  
    token_hash VARCHAR(255) NOT NULL,  
    expires_at TIMESTAMP WITH TIME ZONE NOT NULL,  
    revoked BOOLEAN DEFAULT FALSE,  
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
```

```
);
```

-- INSERCIÓN DE DATOS INICIALES (SEMILLA) -----

-- Insertamos las categorías exactas que pediste

```
INSERT INTO roles (name, description) VALUES
```

```
('Admin', 'Acceso total al sistema, gestión de usuarios y configuración global.'),
```

```
('Operador', 'Acceso de escritura y ejecución en Pyrolysis Hub (Ala Analítica).'),
```

```
('Viewer', 'Acceso de solo lectura a reportes finales y dashboards.'),
```

```
('Academico', 'Acceso a datos crudos y exportación científica, validación de modelos.'),
```

```
('Colaborador', 'Acceso principal al Creador de Prompt (Ala Creativa) y edición de narrativa.');
```

-- Ejemplo de creación de un usuario (La contraseña debe ser hasheada por el backend antes de llegar aquí)

```
-- INSERT INTO users (full_name, email, password_hash)
```

```
-- VALUES ('Dr. Ejemplo', 'dr.ejemplo@nexosinergico.com',
```

```
'$2b$12$R9h/cIPz0gi.URNNXRFXjO...');
```

Por qué este diseño es ideal para tu "Nexo Sinérgico":

- 1. Seguridad JWS (`token_version`):**
 - En un sistema JWT estándar, si roban el token, el hacker tiene acceso hasta que el token expire.
 - Con el campo `token_version` en la tabla `users`: Cuando generas el JWS, incluyes la versión actual (ej: `v=1`). Si el usuario reporta un robo o cambia su contraseña, en la base de datos subes la versión a `2`. El token viejo (`v=1`) será rechazado inmediatamente por el backend aunque no haya expirado por fecha.
- 2. Flexibilidad de Roles (`user_roles`):**
 - En tu ecosistema, las líneas son borrosas. Un "Académico" podría querer colaborar en la parte creativa. Con la tabla intermedia, puedes asignarle ambos roles (`Academico + Colaborador`) sin duplicar el usuario.
- 3. Preparado para Auditoría:**
 - Los campos `created_at` y `last_login_at` son fundamentales para saber quién está usando la plataforma y cuándo.

Siguiente paso sugerido

Para completar la seguridad del login, necesitarás el código del Backend (API) que reciba la contraseña plana, la convierta en Hash y firme el JWS.