Esta es la pieza clave que convierte un sistema de JWT "estándar" (que suele ser vulnerable si te roban el token) en un sistema **"Stateful JWT"** de alta seguridad.

En **FastAPI**, el "middleware" de protección se implementa mejor como una **Dependencia (`Depends`)**. Esto permite inyectar la seguridad en rutas específicas.

Aquí tienes la lógica del **Guardián del Nexo Sinérgico**.

Diagrama de Flujo de Verificación

El Código del Guardián (`dependencies.py`)

Este código intercepta la petición, decodifica el token y realiza la **verificación crítica de versión** contra la base de datos.

Python

```
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from jose import jwt, JWTError
from sqlalchemy.orm import Session

# Importamos lo necesario del archivo anterior (main.py)
from main import get_db, User, SECRET_KEY, ALGORITHM

# Define la fuente del token (el Header "Authorization: Bearer <token>")
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="auth/login")

def get_current_user(
    token: str = Depends(oauth2_scheme),
    db: Session = Depends(get_db)
):
    """
    Esta función es el 'Portero'. Se ejecuta antes de entrar a cualquier ruta protegida.
    Valida firma, expiración y LA VERSIÓN DEL TOKEN.
    """
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Credenciales inválidas o token revocado",
        headers={"WWW-Authenticate": "Bearer"},
    )

    try:
        # 1. Decodificar el JWS (Verifica firma y fecha de expiración automáticamente)
```

```
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])

        user_id: str = payload.get("sub")
        token_ver: int = payload.get("ver") # <--- Aquí extraemos la versión del token

        if user_id is None or token_ver is None:
            raise credentials_exception

    except JWTError:
        raise credentials_exception

    # 2. Consultar al Usuario en la DB
    user = db.query(User).filter(User.id == user_id).first()

    if user is None:
        raise credentials_exception

    # 3. VERIFICACIÓN CRÍTICA: ¿Coinciden las versiones?
    # Si la versión en la DB es mayor que la del token, significa que
    # el usuario cambió password o fue revocado después de emitir este token.
    if user.token_version != token_ver:
        print(f"ALERTA: Intento de uso de token revocado (v{token_ver}) para usuario
v{user.token_version}")
        raise credentials_exception

    # 4. Verificar si está activo (Ban global)
    if not user.is_active:
        raise HTTPException(status_code=400, detail="Usuario inactivo")

    return user
```

Implementación en las Rutas ("El Nexo en Acción")

Ahora veamos cómo aplicamos este guardián en tus dos mundos: el **Analítico** y el **Creativo**, usando también validación de Roles.

Python
```
from fastapi import APIRouter

# Helper para verificar roles dentro de la ruta
def require_role(user: User, required_role: str):
    # Extraemos los nombres de los roles del usuario
```

```
    user_roles = [r.name for r in user.roles]
    if required_role not in user_roles and "Admin" not in user_roles:
        raise HTTPException(
            status_code=403,
            detail=f"Se requiere rol de {required_role} para esta acción"
        )

# --- RUTAS PROTEGIDAS ---

app = FastAPI() # (O el mismo app de main.py)

@app.get("/pyrolysis/simulation-data")
def read_simulation_data(current_user: User = Depends(get_current_user)):
    """
    Ruta del Hub Analítico.
    Solo accesible si el token es válido y no ha sido revocado.
    """
    # Validación extra de rol (opcional aquí o en middleware)
    require_role(current_user, "Operador") # O Académico

    return {
        "user": current_user.email,
        "data": "Resultados de Pirólisis: Eficiencia 75%",
        "status": "Secure"
    }

@app.post("/creative/generate-prompt")
def generate_creative_prompt(current_user: User = Depends(get_current_user)):
    """
    Ruta del Creador de Prompt.
    """
    require_role(current_user, "Colaborador")

    return {
        "user": current_user.email,
        "prompt": "Genera una imagen futurista de una planta de biomasa...",
        "context_source": "Pyrolysis Hub Data"
    }

# --- RUTA DE EMERGENCIA (ADMIN) ---

@app.post("/admin/revoke-user-tokens/{user_email}")
def revoke_tokens(user_email: str, current_user: User = Depends(get_current_user), db:
Session = Depends(get_db)):
```

```
    """
    Botón de Pánico: Invalida TODOS los tokens existentes de un usuario al instante.
    """
    require_role(current_user, "Admin")

    target_user = db.query(User).filter(User.email == user_email).first()
    if not target_user:
        raise HTTPException(status_code=404, detail="Usuario no encontrado")

    # LA MAGIA: Simplemente incrementamos la versión
    target_user.token_version += 1
    db.commit()

    return {"msg": f"Tokens revocados. El usuario {user_email} deberá loguearse de nuevo."}
```

Escenario de Uso Real

Imagina la siguiente situación en tu laboratorio:

1. **El Incidente:** Un "Colaborador" pierde su laptop en el tren. Tiene la sesión abierta (Token válido por 24h).
2. **La Acción:** El Administrador recibe la llamada y usa el endpoint `/admin/revoke-user-tokens`.
3. **El Resultado en DB:** El campo `token_version` del colaborador cambia de `1` a `2`.
4. **El Intento de Acceso:**
    - Alguien encuentra la laptop e intenta generar un prompt.
    - El navegador envía el token viejo (que tiene `ver: 1` incrustado).
    - Tu API recibe el token. `get_current_user` consulta la DB.
    - **Comparación:** `token (v1) != DB (v2)`.
    - **Rechazo:** La API devuelve `401 Unauthorized` inmediatamente.
    - El Frontend redirige automáticamente al Login.

Siguiente Paso de Valor

Ya tienes la **Base de Datos** y la **Seguridad Backend**