Aquí tienes la implementación robusta en **Python (FastAPI)** para el flujo de Recuperación de Contraseña.

Este código se integra con tu esquema anterior y añade tres capas de seguridad críticas:

1. **Protección contra Enumeración:** La API responde "200 OK" aunque el email no exista, para que los hackers no puedan "pescar" qué correos están registrados.
2. **Tokens de Vida Corta:** El enlace expira en 15 minutos.
3. **Invalidación Global (Nexo Security):** Al cambiar la contraseña, incrementamos el `token_version`, expulsando al hacker de cualquier sesión activa inmediatamente.

1. Preparación del Modelo y Schemas

Agrega esto a tu archivo de modelos o `main.py`. Necesitamos definir la tabla `password_resets` y los esquemas de validación.

Python
```
from datetime import datetime, timedelta
from uuid import uuid4
import secrets # Para generar tokens criptográficamente seguros

from fastapi import APIRouter, HTTPException, status, Depends, BackgroundTasks
from pydantic import BaseModel, EmailStr, Field
from sqlalchemy import Column, String, Boolean, DateTime, ForeignKey
from sqlalchemy.orm import Session

# Importamos objetos previos de tu proyecto
from main import Base, User, get_db, get_password_hash, engine

# --- A. MODELO DE BASE DE DATOS (SQLAlchemy) ---
class PasswordReset(Base):
    __tablename__ = "password_resets"

    id = Column(String, primary_key=True, default=lambda: str(uuid4()))
    user_id = Column(String, ForeignKey("users.id"))
    token = Column(String, unique=True, index=True) # El "Código Secreto" del link
    expires_at = Column(DateTime)
    used = Column(Boolean, default=False)
    created_at = Column(DateTime, default=datetime.utcnow)

# Crear la tabla si no existe (Solo para demo)
Base.metadata.create_all(bind=engine)
```

```
# --- B. SCHEMAS PYDANTIC (Datos que envía el Frontend) ---

class ForgotPasswordRequest(BaseModel):
    email: EmailStr

class ResetPasswordAction(BaseModel):
    token: str
    new_password: str = Field(..., min_length=8, description="Mínimo 8 caracteres")
    confirm_password: str

# --- C. UTILIDAD SIMULADA DE EMAIL ---
def send_email_mock(email: str, token: str):
    """
    En producción, aquí usarías SendGrid, AWS SES o SMTP.
    Para este prototipo, imprimimos el link en la consola del servidor.
    """
    print(f"\n[EMAIL SERVICE] ---------------------------------------------")
    print(f"Para: {email}")
    print(f"Asunto: Recuperación de Acceso - Nexo Sinérgico")
    print(f"Mensaje: Usa este enlace para resetear tu password (válido 15 min):")
    print(f"http://localhost:3000/auth/reset-password?token={token}")
    print(f"-------------------------------------------------------------\n")
```

---

2. Los Endpoints (El Cerebro de la Recuperación)

Crea un `APIRouter` o agrégalos a tu `app` principal.

Python
```
router = APIRouter(prefix="/auth", tags=["Recuperación de Contraseña"])

@router.post("/forgot-password", status_code=status.HTTP_200_OK)
def request_password_reset(
    request: ForgotPasswordRequest,
    background_tasks: BackgroundTasks, # Para enviar email sin bloquear la API
    db: Session = Depends(get_db)
):
    """
    Paso 1: El usuario pide recuperar su cuenta.
    Genera un token temporal y simula el envío de correo.
    """
    # 1. Buscar usuario
    user = db.query(User).filter(User.email == request.email).first()
```

```
    # 2. Lógica de Seguridad (Anti-Enumeración)
    # Si el usuario NO existe, no decimos nada. Fingimos que enviamos el email.
    # Esto evita que un atacante sepa qué emails están registrados.
    if not user:
        # Simulamos un retardo para que el tiempo de respuesta sea igual
        return {"msg": "Si el correo existe, recibirás instrucciones en breve."}

    # 3. Generar Token Seguro (Hexadecimal)
    reset_token = secrets.token_urlsafe(32)

    # 4. Guardar en DB (Expira en 15 mins)
    db_reset = PasswordReset(
        user_id=user.id,
        token=reset_token,
        expires_at=datetime.utcnow() + timedelta(minutes=15),
        used=False
    )
    db.add(db_reset)
    db.commit()

    # 5. Enviar Email (en segundo plano)
    background_tasks.add_task(send_email_mock, user.email, reset_token)

    return {"msg": "Si el correo existe, recibirás instrucciones en breve."}


@router.post("/reset-password", status_code=status.HTTP_200_OK)
def execute_reset_password(
    action: ResetPasswordAction,
    db: Session = Depends(get_db)
):
    """
    Paso 2: El usuario hace clic en el link y envía el token + nueva pass.
    """
    # 1. Validar coincidencia de passwords (UX básica)
    if action.new_password != action.confirm_password:
        raise HTTPException(status_code=400, detail="Las contraseñas no coinciden")

    # 2. Buscar el token en DB
    reset_entry = db.query(PasswordReset).filter(PasswordReset.token == action.token).first()

    # 3. Validaciones de Seguridad del Token
    if not reset_entry:
```

```
        raise HTTPException(status_code=400, detail="Token inválido o inexistente")

    if reset_entry.used:
        raise HTTPException(status_code=400, detail="Este token ya fue utilizado")

    if reset_entry.expires_at < datetime.utcnow():
        raise HTTPException(status_code=400, detail="El token ha expirado. Solicita uno nuevo.")

    # 4. Obtener al Usuario dueño del token
    user = db.query(User).filter(User.id == reset_entry.user_id).first()
    if not user:
        raise HTTPException(status_code=404, detail="Usuario no encontrado")

    # --- LA MAGIA DEL NEXO SINÉRGICO ---

    # A. Actualizar Password (Hasheada)
    user.password_hash = get_password_hash(action.new_password)

    # B. INVALIDACIÓN GLOBAL (Segregación de Sesión)
    # Incrementamos la versión. Cualquier token JWT anterior (en el móvil del hacker,
    # en la laptop robada) dejará de funcionar inmediatamente porque su 'ver' será
    # menor a este nuevo número.
    user.token_version += 1

    # C. Marcar token como usado (Quemarlo)
    reset_entry.used = True

    # 5. Guardar Cambios
    db.commit()

    # Opcional: Registrar en Audit Log aquí (Si implementaste la tabla de auditoría)
    # create_audit_log(db, user.id, "PASSWORD_RESET", "Recuperación exitosa via token")

    return {"msg": "Contraseña actualizada correctamente. Inicia sesión con tu nueva
credencial."}
```

Explicación de la Lógica de "Nexo Security"

1. **`secrets.token_urlsafe(32)`**: Usamos la librería `secrets` de Python, diseñada para criptografía. `random` no es suficiente para generar tokens de seguridad.
2. **`background_tasks`**: En FastAPI, esto es vital. Permite responder al usuario "Correo enviado" instantáneamente, mientras el servidor procesa el envío del email en paralelo. Evita que la interfaz se congele.

3. **El "Kill Switch" (`user.token_version += 1`)**:
    - Imagina que un atacante tiene la contraseña vieja y está logueado.
    - El usuario legítimo recupera la cuenta con este endpoint.
    - Al hacer `commit`, la DB actualiza la versión.
    - El siguiente request que haga el atacante fallará en tu Middleware (que programamos antes) porque su token tiene la versión vieja. **Recuperación total.**

Cómo probarlo en tu entorno local

1. Asegúrate de tener un usuario creado (ej. `cientifico@nexo.com`).
2. Envía un POST a `/auth/forgot-password` con ese email.
3. Mira la consola/terminal donde corre Python. Verás el **Link Simulado**.
4. Copia el `token` de ese link.
5. Envía un POST a `/auth/reset-password` con ese token y una nueva contraseña.
6. Intenta loguearte con la contraseña vieja -> **Fallará**.
7. Intenta loguearte con la nueva -> **Éxito**.