**1. Validación y normalización de datos**

* Validar bien emails, nombres, DNI, fechas, etc.
* Normalizar emails (por ejemplo, lower() para evitar duplicados por mayúsculas).
* Validar formato de contraseña (longitud mínima, complejidad).
* Usar Pydantic para validaciones complejas en los schemas (UserCreate, UserUpdate).

**2. Roles y permisos más completos**

* Ya usás un enum de roles, genial.
* Implementar permisos finos (por ejemplo, solo admin puede ver todos usuarios, o cambiar roles).
* Crear un decorator o dependencia para autorización basada en roles para proteger endpoints según permiso.

**3. Recuperación y cambio de contraseña**

* Endpoints para:
  + Solicitar reseteo de contraseña (enviar email con token temporal).
  + Resetear contraseña con token válido.
* Esto implica generar tokens de uso único y con expiración.

**4. Auditoría y logs**

* Registrar actividad importante (logins, cambios de contraseña, creación/edición de usuarios).
* Fecha y hora de última conexión (puede ayudar en seguridad y análisis).

**5. Tests automatizados**

* Unit tests y tests de integración para el CRUD, seguridad, y autorización.
* Que prueben casos felices y manejo de errores.

**6. Documentación**

* Mejorar documentación (ej. con OpenAPI y descripciones en endpoints).
* Documentar claramente el modelo de datos y los flujos de autenticación.

**7. Mejor manejo de errores**

* Manejar errores específicos con mensajes claros y estandarizados.
* Usar respuestas consistentes en toda la API.

**8. Soporte para paginación, filtros y ordenamiento**

* Para el listado de usuarios (GET /users), agregar parámetros para filtrar, ordenar, paginar.
* Esto mejora escalabilidad.

**9. Optimización de seguridad**

* Limitar intentos fallidos de login (rate limiting).
* Usar HTTPS en producción (certificados SSL).
* Guardar secrets en variables de entorno y no en código.
* Revisión periódica del algoritmo de hash y parámetros (bcrypt rounds).

**10. Opcional: Refactorización para ser paquete reusable**

* Organizar como paquete Python independiente, con configuración flexible.
* Separar lógica de negocio, modelos, y rutas.
* Permitir configuración externa (p. ej. conexión a DB, JWT settings).
* Documentar instalación y uso para otros proyectos.