

ISS-03 — Custom User antes de la primera migración

Estableciendo los cimientos de
autenticación en Django

Módulo de Arquitectura Backend

El Objetivo:
¿Qué construiremos?

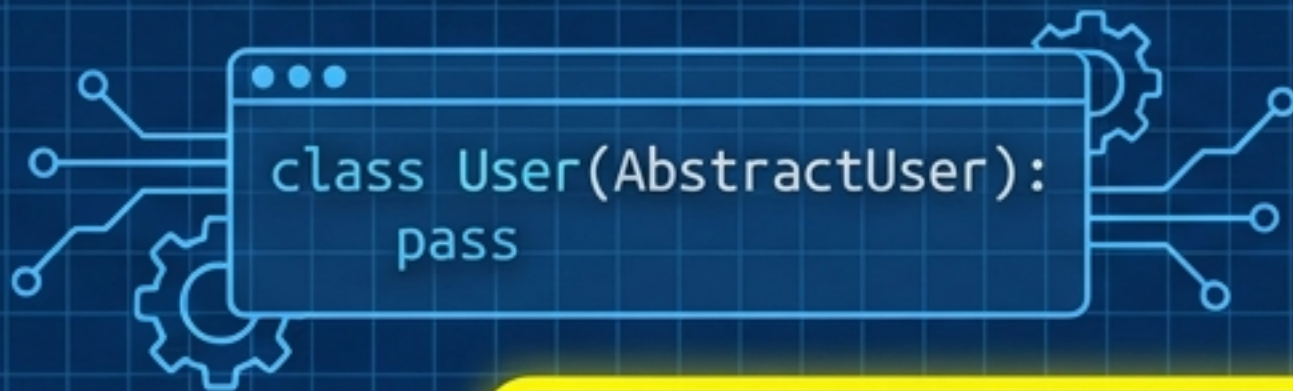
DEFAULT
USER



CUSTOM
SHIELDED
USER

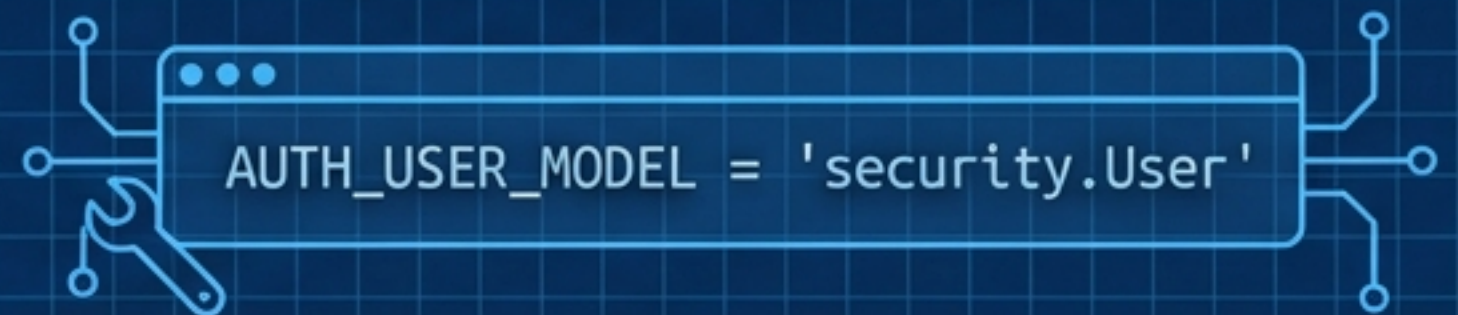
El Modelo

Definición de un modelo de usuario personalizado (security.User) que hereda de AbstractUser.



El Registro

Configuración crítica de AUTH_USER_MODEL en los Settings del proyecto.



REQUISITO INNEGOCIABLE: Todo esto debe realizarse
ANTES de ejecutar la primera migración en la base de datos.

Punto de Partida: Condiciones Previas

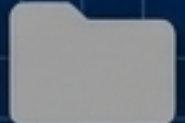
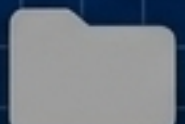
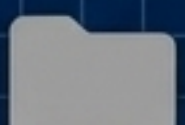


```
[OK] ISS-02 completado con éxito.  
[OK] Entorno virtual (.venv) activo.  
[OK] Apps de negocio (client, product, sale) creadas físicamente.  
[OK] Apps registradas en INSTALLED_APPS.  
  
[OK] Apps registradas en INSTALLED_APPS.
```

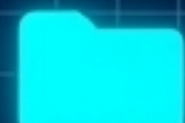
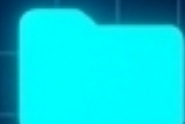
¡ALERTA! Ningún modelo de negocio ha sido escrito aún,
y la base de datos sigue completamente vacía.

Evolución de la Arquitectura

Estado Inicial

-  apps/client/
-  apps/product/
-  apps/sale/

Estado Resultante

-  apps/client/
-  apps/product/
-  apps/sale/
-  apps/common/ [NUEVO PAQUETE]
-  apps/security/ [NUEVA APP]

El Paquete 'common' y el Enumerado de Estado

apps/common/

- Paquete Python.
- NO es una app Django.
- NO tiene apps.py.
- NO se registra en settings.
- NO genera migraciones.
- Almacena lógica compartida.

apps/security/

- App Django completa.
- Tiene apps.py.
- Registrada en settings.
- Genera migraciones.
- Gestiona tablas en la BD.



apps/common/status.py

```
class RecordStatus(models.TextChoices):  
    ACTIVE = 'active', 'Active'  
    INACTIVE = 'inactive', 'Inactive'
```


Instanciación: Creación de la App 'security'

```
(.venv) $ mkdir -p apps/security  
(.venv) $ python manage.py startapp security apps/security
```

Nota Arquitectónica:

El comando startapp exige que el directorio destino ya exista. Se utiliza la bandera -p para asegurar la creación del árbol completo sin errores.

Configuración del Registro (apps.py)

apps/security/apps.py

```
name = 'security'  
+ name = 'apps.security'  
+ label = 'security'
```

Ruta absoluta de importación (dónde vive el código físicamente).

Prefijo interno de Django para las tablas de la base de datos (evita que arrastren el nombre del paquete superior).

Lógica Central: El Gestor Personalizado

apps/security/models.py

```
class StoreUserManager(UserManager):
```

```
    def create_user(self, username, email=None, password=None, **extra_fields):  
        extra_fields.setdefault('status', RecordStatus.INACTIVE)  
        return super().create_user(...)
```

Seguridad por defecto: Los usuarios nacen inactivos si la lógica de negocio no indica lo contrario.

```
    def create_superuser(self, username, email=None, password=None, **extra_fields):  
        extra_fields.setdefault('status', RecordStatus.ACTIVE)  
        extra_fields.setdefault('is_staff', True)  
        extra_fields.setdefault('is_superuser', True)  
        return super().create_superuser(...)
```

Inmunidad administrativa: Fuerza el estado activo. Evita que el método `save()` posterior desactive accidentalmente al superusuario.

Anatomía del Modelo User (Herencia y Campos)

apps/security/models.py

```
class User(AbstractUser):
    email = models.EmailField(unique=True)
    status = models.CharField(
        choices=RecordStatus.choices,
        default=RecordStatus.INACTIVE
    )
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

Herencia: Obtenemos campos base (password, first_name) y la integración con el panel administrativo gratuitamente.

Identidad: El correo se vuelve único, preparando el terreno para autenticación basada en email en lugar de username.

Metadatos y Restricciones (Clase Meta)



apps/security/models.py

```
class Meta:
    db_table = 'users'

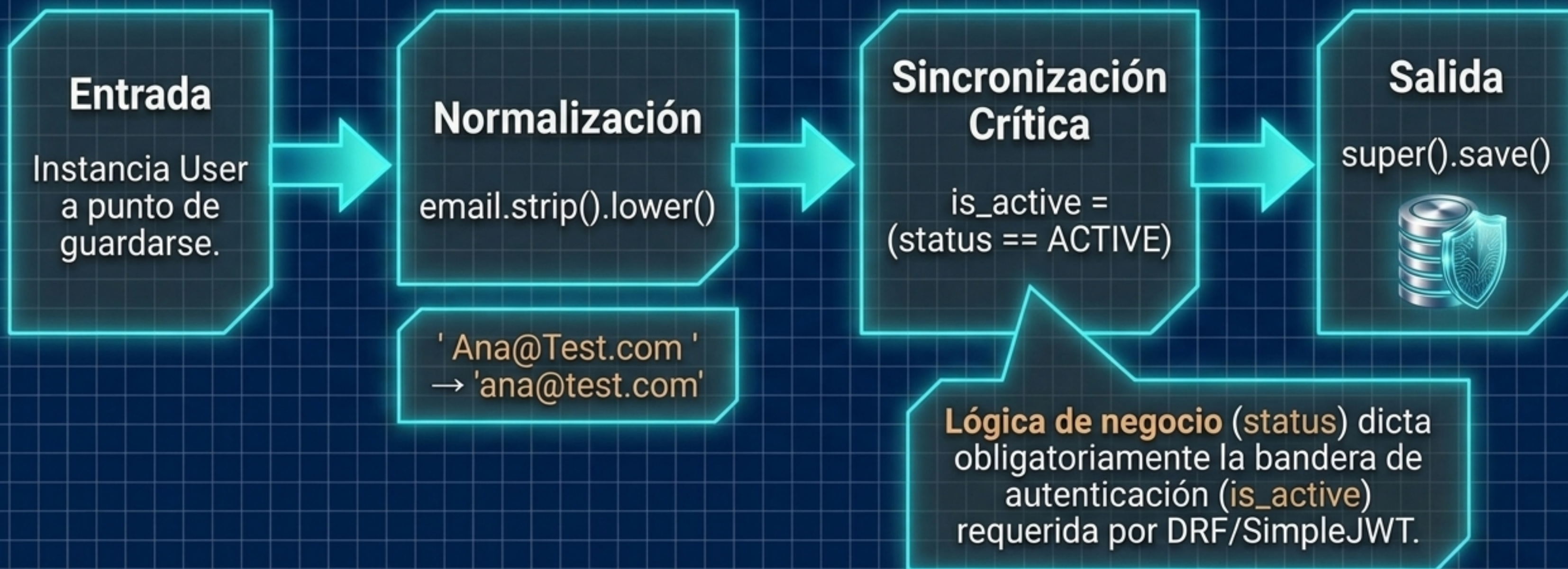
    constraints = [
        models.CheckConstraint(
            condition=models.Q(status__in=RecordStatus.values),
            name='users_status_valid'
        )
    ]
```

Nomenclatura limpia: Evita el nombre autogenerado por defecto (security_user).

Integridad de Base de Datos: Garantiza que ninguna inserción directa en SQL pueda violar los estados permitidos ('active' o 'inactive').



El Hook de Guardado (save())



Conectando el Sistema (Settings)

El Registro (INSTALLED_APPS)

```
INSTALLED_APPS = [  
    ... apps por defecto ...  
    'apps.security.apps.SecurityConfig',  
    'apps.client.apps.ClientConfig',  
    'apps.product.apps.ProductConfig',  
    'apps.sale.apps.SaleConfig',  
]
```

Se inserta **ANTES** de las apps de negocio.

El Enganche Global

```
AUTH_USER_MODEL = 'security.User'
```

Esta **única línea** le dice a todo el framework Django (admin, auth, relaciones foráneas) que ignore el modelo de usuario por defecto y utilice el nuestro.

Pruebas Unitarias de Selección de Motor

Implementación de **DatabaseSelectionTests** en **apps/security/tests.py**.
(Se ubica aquí por ser la primera app real con tests del proyecto).



PostgreSQL

`django.db.backends.postgresql`



MySQL

`django.db.backends.mysql`
Opciones: `charset utf8mb4`



SQL Server

`mssql` (Vía `mssql-django`)



Oracle

Uso de `SERVICE_NAME`
en el DSN.

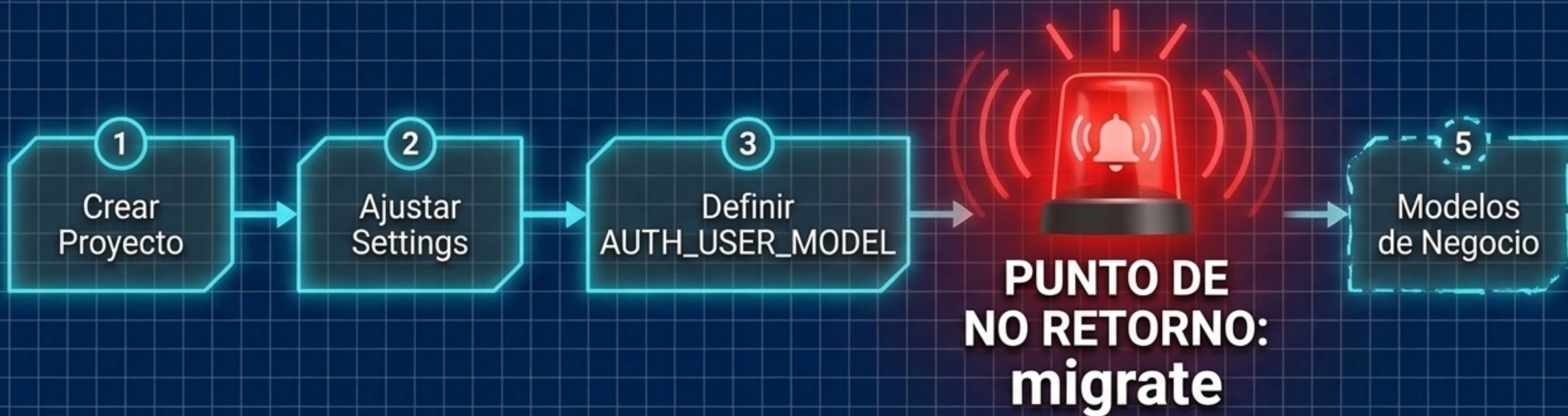
Manejo de Errores: Valida `ImproperlyConfigured` para motores inválidos o variables de entorno faltantes.

Ejecución: La Primera Migración

```
> python manage.py makemigrations security
Migrations for 'security':
  apps/security/migrations/0001_initial.py
    - Create model User
> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, security, sessions
Running migrations:
  Applying security.0001_initial... OK
> python manage.py test apps.security.tests.DatabaseSelectionTests
Ran 6 tests in 0.012s
OK
```

Si se sigue el proceso estrictamente, la migración inicial contendrá **exclusivamente el modelo User**.

Alerta Arquitectónica: ¿Por qué AHORA?



EL DESASTRE

Si se ejecuta **migrate** ANTES de definir **AUTH_USER_MODEL**, Django creará las tablas físicas para su usuario por defecto (auth_user). Cambiar la configuración después de este punto corrompe el esquema, rompe las llaves foráneas y obliga a destruir y recrear la base de datos completa.

Matriz de Decisión: AbstractUser vs. AbstractBaseUser

Criterio	AbstractUser	AbstractBaseUser
Campos base (nombre, email)	✓ Incluidos	✗ Requiere construir desde cero
Gestión de contraseñas (Hash)	✓ Resuelto	✓ Resuelto
Integración con Admin UI	✓ Automática	✗ Requiere código manual complejo
Validadores nativos	✓ Incluidos	✗ Manuales

Veredicto: Elegimos **AbstractUser**. Resuelve la criptografía y la gestión administrativa de forma segura, evitando reinventar sistemas críticos (práctica prohibida en este laboratorio).

Diagnóstico de Errores Frecuentes

WATCH OUT

	Migrar prematuramente	Ejecutar migrate antes de configurar settings. (Requiere purgar la base de datos y reiniciar).
	Desincronizar el estado	Olvidar la línea <code>is_active = (status == ACTIVE)</code> en <code>save()</code> . (El usuario queda bloqueado a nivel JWT).
	Desarmar superusuarios	No sobrescribir <code>create_superuser</code> forzando <code>ACTIVE</code> . (El usuario administrador nace inactivo y bloqueado).
	Ignorar normalización	No aplicar <code>strip().lower()</code> al email. (<code>'Ana@test.com'</code> y <code>'ana@test.com'</code> crearán colisiones o duplicados lógicos).

GATE: Criterios y Evidencias

Criterio de Aceptación (AC)	Verificación	Evidencia
AC-03-01: AUTH_USER_MODEL apunta a security.User	Inspección de settings	config/settings/__init__.py
AC-03-02: Modelo Client no tiene password	Inspección de estructura del modelo	Sin campo password en models.py
AC-03-03: Usuario inactivo no inicia sesión	Prueba de flujo HTTP (Espera 401)	AuthFlowTests PASS
AC-03-04: Clave incorrecta / Hash protegido	Prueba de autenticación	AuthFlowTests PASS

Preparación para la Defensa Oral

¿Por qué el paquete common no se declara en la lista de INSTALLED_APPS?

¿Qué diferencia arquitectónica exacta existe entre los atributos name y label en el archivo apps.py?

¿Por qué conservamos obligatoriamente la columna is_active si ya introdujimos nuestra propia columna status?

¿Cuál es la consecuencia física en la base de datos si cambiamos AUTH_USER_MODEL después de haber ejecutado migraciones previas?