Guia Completa - Proyecto EduPro 360

Historia de Usuario HU-01: Registro de Usuarios con Roles y Permisos

Esta guia documenta paso a paso todo el proceso de desarrollo, desde la concepcion inicial hasta la implementacion completa, incluyendo decisiones tecnicas, problemas encontrados y soluciones aplicadas.

Contexto y Objetivo

Desarrollar un sistema de gestion escolar (EduPro 360) con Django REST Framework.

HU-01: Registro de Usuarios - Permitir el registro de usuarios con roles (super_admin, admin, profesor, estudiante), validacion de correo unico, politicas de seguridad de contrasena, y envio automatico de correo de bienvenida.

Campos requeridos: nombre, apellido, correo (unico), contrasena (validada), rol, estado (activo/inactivo), fecha_creacion (automatica).

Autenticacion: JWT (JSON Web Tokens) con SimpleJWT.

Comunicacion asincrona: Celery + Redis para envio de emails.

Documentacion: Swagger/OpenAPI via drf-spectacular.

Arquitectura y Stack Tecnologico

Backend: Django 5.2.9 + Django REST Framework 3.15.2

Autenticacion: djangorestframework-simplejwt 5.3.1

Base de datos: PostgreSQL (via psycopg2-binary)

Tareas asincronas: Celery 5.4.0 + Redis 5.0.1 (broker y result backend)

Documentacion API: drf-spectacular 0.27.2 (Swagger UI)

CORS: django-cors-headers 4.4.0 (desarrollo permisivo)

Email: SMTP Gmail con App Password (TLS puerto 587)

Entorno: Python 3.13, venv, Windows + VS Code

Decision 1: Modelo de Usuario Personalizado

Por que: Django recomienda usar AbstractUser desde el inicio para evitar migraciones complejas posteriores.

Implementacion: Extender AbstractUser para anadir campos 'rol', 'estado', 'fecha_creacion'.

Configuracion: AUTH_USER_MODEL = 'usuarios.Usuario' en settings.py ANTES de la primera migracion.

Ventaja: Flexibilidad total para anadir campos y metodos personalizados sin afectar el sistema de autenticacion de Django.

Prerrequisitos del Entorno

Sistema operativo: Windows 10/11

IDE: Visual Studio Code con extensiones Python y Pylance

Python 3.13 instalado y accesible desde PATH

PostgreSQL 12+ instalado (con psql o pgAdmin)

Redis instalado (WSL, Docker o Memurai en Windows)

Cuenta Gmail con verificacion en dos pasos activada (para App Password)

Git (opcional, para control de versiones)

1. Crear entorno y proyecto

Crea una carpeta de trabajo y abre terminal en ella.

Genera entorno virtual y actívalo.

```
# En PowerShell:
python -m venv env
# Activar
./env/Scripts/Activate.ps1
# Actualizar pip
python -m pip install --upgrade pip
```

2. Instalar dependencias

Instala Django, Django REST Framework, SimpleJWT, CORS, Psycopg2 (PostgreSQL), Celery, Redis, drf-spectacular.

```
pip install django==5.2.9 djangorestframework drf-spectacular djangorestframework-simplejwt django-cors-headers ps
```

3. Crear proyecto y app

Crea el proyecto y la app principal de usuarios.

```
django-admin startproject edu .
python manage.py startapp applications/usuarios
```

4. Configurar settings.py

Registra apps: 'rest_framework', 'drf_spectacular', 'corsheaders', 'applications.usuarios'.

Configura base de datos PostgreSQL.

Activa DRF y JWT.

Configura CORS.

Añade Celery + Redis y correo SMTP (Gmail con App Password).

Selecciona modelo de usuario personalizado: AUTH_USER_MODEL='usuarios.Usuario'.

```
# Fragmentos clave de edu/settings.py
INSTALLED_APPS = [
    'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes',
    'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles',
    'corsheaders', 'rest_framework', 'rest_framework.authtoken', 'drf_spectacular',
    'applications.usuarios',
]

DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.postgresql',
    'NAME': 'educar', 'USER': 'postgres', 'PASSWORD': '*******', 'HOST': 'localhost', 'PORT': '5432'
  }
}

REST_FRAMEWORK = {
  'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema',
  'DEFAULT_AUTHENTICATION_CLASSES': ['rest_framework_simplejwt.authentication.JWTAuthentication'],
  'DEFAULT_PERMISSION_CLASSES': ['rest_framework.permissions.IsAuthenticated'],
}

from datetime import timedelta
SIMPLE_JWT = {
  'ACCESS_TOKEN_LIFETIME': timedelta(hours=1),
  'REFRESH_TOKEN_LIFETIME': timedelta(days=7),
  'AUTH_HEADER_TYPES': ('Bearer',),
}

CORS_ALLOW_ALL_ORIGINS = True
CORS_ALLOW_CREDENTIALS = True

# Celery + Redis
CELERY_BROKER_URL = 'redis://localhost:6379/0'
CELERY_RESULT_BACKEND = 'redis://localhost:6379/0'
CELERY_TIMEZONE = 'America/Bogota'

# Email (Gmail SMTP)
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'tu_correo@gmail.com'
EMAIL_HOST_PASSWORD = 'tu_app_password'
DEFAULT_FROM_EMAIL = 'Colegio Django <tu_correo@gmail.com>'

# Modelo de usuario propio
AUTH_USER_MODEL = 'usuarios.Usuario'
```

5. Inicializar Celery

Crea edu/celery.py y referencia en edu/__init__.py para autodiscover de tareas.

```
# edu/celery.py
import os
from celery import Celery
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'edu.settings')
```

```
app = Celery('edu')
app.config_from_object('django.conf:settings', namespace='CELERY')
app.autodiscover_tasks()
```

```
# edu/__init__.py
from .celery import app as celery_app
__all__ = ('celery_app',)
```

6. Modelo Usuario (roles, estado, fecha)

Define roles y estado, y fecha de creación automática.

```
# applications/usuarios/models.py
from django.contrib.auth.models import AbstractUser
from django.db import models

class Usuario(AbstractUser):
    ROLES = (
        ('super_admin', 'Super Administrador'),
        ('admin', 'Administrador'),
        ('profesor', 'Profesor'),
        ('estudiante', 'Estudiante'),
    )
    ESTADOS = (
        ('activo', 'Activo'),
        ('inactivo', 'Inactivo'),
    )
    rol = models.CharField(max_length=20, choices=ROLES, default='estudiante')
    estado = models.CharField(max_length=20, choices=ESTADOS, default='activo')
    fecha_creacion = models.DateTimeField(auto_now_add=True, null=True, blank=True)

    class Meta:
        verbose_name = 'Usuario'
        verbose_name_plural = 'Usuarios'

    def __str__(self):
        return f"{self.username} - {self.get_rol_display()}"
```

```
# Migraciones
python manage.py makemigrations usuarios
python manage.py migrate
```

7. Serializers (registro, lectura, login)

Valida email y username únicos, políticas de contraseña, y envía email de bienvenida.

```
# applications/usuarios/api/serializar.py (fragmentos)
from rest_framework import serializers
from django.contrib.auth import get_user_model
import re
Usuario = get_user_model()

class RegistroSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True, min_length=8, style={'input_type': 'password'})
    password_confirm = serializers.CharField(write_only=True, style={'input_type': 'password'})
    class Meta:
        model = Usuario
        fields = ('username','email','first_name','last_name','password','password_confirm','rol','is_active')
```

```
        extra_kwargs = {'is_active': {'required': False}}

    # Validaciones de email, username y fuerza de contraseña...

    def create(self, validated_data):
        from applications.usuarios.tasks import send_welcome_email
        password = validated_data.pop('password')
        if 'is_active' not in validated_data:
            validated_data['is_active'] = True
        usuario = Usuario.objects.create_user(**validated_data, password=password)
        send_welcome_email.delay(user_email=usuario.email, username=usuario.username, first_name=usuario.first_nam
        return usuario

class UsuarioSerializer(serializers.ModelSerializer):
    rol_display = serializers.CharField(source='get_rol_display', read_only=True)
    class Meta:
        model = Usuario
        fields = ('id','username','email','first_name','last_name','rol','rol_display','is_active','is_staff','is_

class LoginSerializer(serializers.Serializer):
    username = serializers.CharField()
    password = serializers.CharField(write_only=True, style={'input_type': 'password'})
```

8. ViewSet + Rutas

ViewSet maneja login (JWT), registro (celery email), perfil (me) y CRUD.

Se expone vía router en /api/usuarios/.

```
# applications/usuarios/api/router.py
from rest_framework.routers import DefaultRouter
from .view import UsuarioViewSet
router = DefaultRouter()
router.register(r'usuarios', UsuarioViewSet, basename='usuarios')
urlpatterns = router.urls
```

9. Tarea Celery: envío de bienvenida

Usa Django send_mail con HTML y añade usuario + contraseña (solo desarrollo).

```
# applications/usuarios/tasks.py (fragmentos)
from celery import shared_task
from django.core.mail import send_mail
from django.conf import settings

@shared_task
def send_welcome_email(user_email, username, first_name, password):
    subject = 'Bienvenido al Sistema de Gestion Escolar!'
    message = f"Hola {first_name}\n\nUsuario: {username}\nContraseña: {password}"
    html_message = f'''
    <html><body>
    <h2 style='color:#4CAF50'>Bienvenido!</h2>
    <p>Hola <strong>{first_name}</strong></p>
    <p><strong>Usuario:</strong> {username}</p>
    <p><strong>Contraseña:</strong> {password}</p>
    </body></html>
    '''
```

```
    send_mail(subject, message, settings.DEFAULT_FROM_EMAIL, [user_email], html_message=html_message)
```

10. URLs y Swagger

Incluye router de usuarios y drf-spectacular para documentación.

```
# edu/urls.py (fragmento)
from django.urls import path, include
from drf_spectacular.views import SpectacularAPIView, SpectacularSwaggerView

urlpatterns = [
  path('api/schema/', SpectacularAPIView.as_view(), name='schema'),
  path('api/docs/', SpectacularSwaggerView.as_view(url_name='schema')),
  path('api/', include('applications.usuarios.api.router')),
]
```

Problemas Encontrados y Soluciones

PROBLEMA 1: Error 'no existe la columna usuarios_usuario.estado' al iniciar servidor.

CAUSA: Campos 'estado' y 'fecha_creacion' anadidos al modelo pero migraciones no aplicadas.

SOLUCION: python manage.py makemigrations usuarios && python manage.py migrate

PROBLEMA 2: Email no enviado, error 535 Authentication Failed (Gmail).

CAUSA: Uso de contrasena normal en lugar de App Password.

SOLUCION: Generar App Password en Cuenta Google > Seguridad > Verificacion en dos pasos > Contrasenas de aplicaciones. Actualizar EMAIL_HOST_PASSWORD en settings.py.

PROBLEMA 3: Correos no llegan a dominios corporativos (soy.sena.edu.co, lambdaanalytics.co).

CAUSA: Filtros antispam o politicas de dominio bloquean SMTP externo.

SOLUCION: Revisar carpeta spam/cuarentena del destinatario. Para produccion, usar servicio transaccional (SendGrid, Mailgun) o solicitar whitelist de tu servidor SMTP.

PROBLEMA 4: ModuleNotFoundError para celery, drf-spectacular, simplejwt.

CAUSA: Paquetes no instalados en el entorno virtual activo.

SOLUCION: Activar venv y ejecutar pip install celery redis drf-spectacular djangorestframework-simplejwt.

PROBLEMA 5: Celery worker falla en Windows con 'not enough values to unpack'.

CAUSA: Pool por defecto (prefork) no compatible con Windows.

SOLUCION: Usar pool=solo: celery -A edu worker -l info --pool=solo

Evolucion del Codigo y Decisiones Tecnicas

FASE 1 - Modelo Usuario:

Decision: Extender AbstractUser (no AbstractBaseUser) para mantener compatibilidad con admin y auth de Django.

Anadir choices ROLES y ESTADOS con valores legibles.

Implementar __str__ para mostrar username y rol en el admin.

FASE 2 - Serializadores y Validaciones:

Decision: Separar RegistroSerializer (escritura con validacion) de UsuarioSerializer (lectura).

Validaciones implementadas: email unico, username unico, contrasena min 8 caracteres con mayuscula/minuscula/numero.

Confirmacion de contrasena con campo password_confirm (no guardado en BD).

FASE 3 - Autenticacion JWT:

Decision inicial: DRF Token Authentication (problema: no funciona bien con Swagger Bearer auth).

Migracion a SimpleJWT: tokens con expiracion configurable, refresh tokens, mejor integracion con Swagger.

Configuracion: ACCESS_TOKEN_LIFETIME=1h, REFRESH_TOKEN_LIFETIME=7d, esquema Bearer en drf-spectacular.

FASE 4 - Tareas Asincronas:

Decision: Celery + Redis para envio de emails (evitar bloqueo de request HTTP).

Broker: Redis en localhost:6379 (facil de instalar, rapido, sin persistencia necesaria para emails).

Task send_welcome_email: acepta user_email, username, first_name, password; envia HTML + texto plano.

Cambio posterior: Anadir password al email (solo desarrollo) para facilitar pruebas del cliente.

FASE 5 - Campos Adicionales (HU-01):

Anadir 'estado' (activo/inactivo) y 'fecha_creacion' (auto_now_add=True) al modelo Usuario.

Crear migracion 0002_usuario_estado_usuario_fecha_creacion y aplicar.

Resultado: Base de datos actualizada sin perdida de datos existentes.

Ejecucion en Desarrollo (Paso a Paso)

1. Activar entorno virtual:

./.venv/Scripts/Activate.ps1

2. Iniciar Redis (verificar con redis-cli ping):

- WSL: sudo service redis-server start

- Docker: docker run -d -p 6379:6379 redis

- Memurai (Windows): iniciar desde servicios

3. Iniciar Celery Worker (terminal separada):

celery -A edu worker -l info --pool=solo

4. Iniciar servidor Django (terminal separada):

python manage.py runserver

5. Acceder a Swagger UI:

http://127.0.0.1:8000/api/docs/

Pruebas Funcionales Realizadas

TEST 1 - Registro de usuario:

POST /api/usuarios/registro/ con body: {username, email, first_name, last_name, password, password_confirm, rol}

Resultado esperado: 201 Created, usuario en BD, tarea Celery encolada, email recibido en Gmail.

TEST 2 - Login con JWT:

POST /api/usuarios/login/ con {username, password}

Resultado esperado: 200 OK con tokens {access, refresh, usuario}.

TEST 3 - Perfil autenticado:

GET /api/usuarios/me/ con header Authorization: Bearer

Resultado esperado: 200 OK con datos del usuario autenticado.

TEST 4 - Validaciones de contrasena:

Intentar registro con contrasena sin mayuscula: Rechazado con mensaje claro.

Intentar registro con contrasena <8 caracteres: Rechazado.

Intentar registro con password != password_confirm: Rechazado.

TEST 5 - Email unico:

Registrar dos usuarios con mismo email: Segundo rechazado con error 'Este correo electronico ya esta registrado'.

RESULTADO: Todas las pruebas pasaron exitosamente.

Configuracion de PostgreSQL (Detallado)

1. Instalar PostgreSQL 12+ desde https://www.postgresql.org/download/windows/

2. Durante instalacion, establecer contrasena para usuario 'postgres'.

3. Abrir psql o pgAdmin y crear base de datos:

psql -U postgres -h localhost -p 5432

CREATE DATABASE educar;

\q

4. Configurar en edu/settings.py:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'educar',

'USER': 'postgres',

'PASSWORD': 'tu_contrasena_postgres',

'HOST': 'localhost',

'PORT': '5432',

}

}

5. Probar conexion: python manage.py dbshell (debe abrir psql conectado a 'educar').

Configuracion de Gmail App Password (Detallado)

REQUISITO: Verificacion en dos pasos activada en tu cuenta Google.

PASOS:

1. Ir a https://myaccount.google.com/security

2. En 'Verificacion en dos pasos', hacer clic en 'Contrasenas de aplicaciones'.

3. Seleccionar aplicacion: 'Correo', dispositivo: 'Windows' (o 'Otro').

4. Generar: Google muestra un codigo de 16 caracteres (ej: abcd efgh ijkl mnop).

5. Copiar ese codigo (sin espacios) y pegar en settings.py:

EMAIL_HOST_PASSWORD = 'abcdefghijklmnop'

6. Configurar tambien:

EMAIL_HOST_USER = 'tu_correo@gmail.com'

DEFAULT_FROM_EMAIL = 'Nombre Remitente '

NOTA: No uses tu contrasena normal de Gmail; solo funciona el App Password.

Entregabilidad de Correo (Corporativo)

OBSERVACION: Emails llegan correctamente a Gmail pero no a dominios corporativos/educativos.

CAUSA PROBABLE: Filtros antispam, politicas SPF/DKIM, o cuarentena del dominio receptor.

SOLUCIONES:

1. Revisar carpeta spam/cuarentena del destinatario.

2. Solicitar al administrador del dominio receptor que agregue tu IP/servidor SMTP a whitelist.

3. Usar proveedor transaccional de email (SendGrid, Mailgun, Amazon SES) con SPF/DKIM configurado.

4. En desarrollo, probar con cuentas personales (Gmail, Outlook) para confirmar funcionalidad.

RECOMENDACION PRODUCCION: Migrar a servicio de email transaccional para mayor deliverability.

Resultado Esperado y Checklist Final

Al completar esta guia, tu aplicacion debe:

1. Permitir registro de usuarios con validacion de email unico y contrasena robusta.

2. Enviar correo de bienvenida asincrono via Celery con credenciales del usuario.

3. Autenticar usuarios con JWT y devolver tokens access/refresh.

4. Exponer endpoints RESTful documentados en Swagger UI.

5. Persistir datos en PostgreSQL con modelo Usuario personalizado (roles, estado, fecha).

6. Ejecutar tareas en background sin bloquear requests HTTP.

CHECKLIST:

[ ] Entorno virtual activado y dependencias instaladas (requirements.txt).

[ ] PostgreSQL corriendo y base de datos 'educar' creada.

[ ] Redis corriendo (redis-cli ping responde PONG).

[ ] Migraciones aplicadas (python manage.py migrate sin errores).

[ ] Celery worker ejecutandose (terminal activa con pool=solo).

[ ] Servidor Django corriendo en http://127.0.0.1:8000/.

[ ] Swagger accesible en http://127.0.0.1:8000/api/docs/.

[ ] Registro de usuario funcional y email recibido en Gmail.

[ ] Login devuelve tokens JWT validos.

[ ] Endpoint /api/usuarios/me/ retorna datos del usuario autenticado.

Fin de la guía – EduPro 360

ANEXO - Codigo Completo de Archivos Clave

A continuacion se incluyen los listados completos de todos los archivos criticos del proyecto para facilitar la replicacion exacta.

1. edu/settings.py (Configuracion completa del proyecto)

No se pudo leer edu\edu\settings.py: [Errno 2] No such file or directory: 'C:\\Users\\EQUIPO\\Documents\\LAMDA\\DJANGO\\UDEMY\\m_colegio\\colegio_Dj\\edu\\edu\\edu\\settings.py'

2. edu/urls.py (Rutas principales y Swagger)

No se pudo leer edu\edu\urls.py: [Errno 2] No such file or directory: 'C:\\Users\\EQUIPO\\Documents\\LAMDA\\DJANGO\\UDEMY\\m_colegio\\colegio_Dj\\edu\\edu\\edu\\urls.py'

3. edu/celery.py (Inicializacion de Celery)

No se pudo leer edu\edu\celery.py: [Errno 2] No such file or directory: 'C:\\Users\\EQUIPO\\Documents\\LAMDA\\DJANGO\\UDEMY\\m_colegio\\colegio_Dj\\edu\\edu\\edu\\celery.py'

4. edu/__init__.py (Import de Celery)

No se pudo leer edu\edu__init__.py: [Errno 2] No such file or directory: 'C:\\Users\\EQUIPO\\Documents\\LAMDA\\DJANGO\\UDEMY\\m_colegio\\colegio_Dj\\edu\\edu\\edu__init__.py'

5. applications/usuarios/models.py (Modelo Usuario personalizado)

No se pudo leer edu\applications\usuarios\models.py: [Errno 2] No such file or directory: 'C:\\Users\\EQUIPO\\Documents\\LAMDA\\DJANGO\\UDEMY\\m_colegio\\colegio_Dj\\edu\\edu\\applications\\usuarios\\models.py'

6. applications/usuarios/api/serializar.py (Serializadores con validaciones)

No se pudo leer edu\applications\usuarios\api\serializar.py: [Errno 2] No such file or directory: 'C:\\Users\\EQUIPO\\Documents\\LAMDA\\DJANGO\\UDEMY\\m_colegio\\colegio_Dj\\edu\\edu\\applications\\usuarios\\api\\serializar.py'

7. applications/usuarios/api/view.py (ViewSet con endpoints)

No se pudo leer edu\applications\usuarios\api\view.py: [Errno 2] No such file or directory: 'C:\\Users\\EQUIPO\\Documents\\LAMDA\\DJANGO\\UDEMY\\m_colegio\\colegio_Dj\\edu\\edu\\applications\\usuarios\\api\\view.py'

8. applications/usuarios/api/router.py (Router DRF)

No se pudo leer edu\applications\usuarios\api\router.py: [Errno 2] No such file or directory: 'C:\\Users\\EQUIPO\\Documents\\LAMDA\\DJANGO\\UDEMY\\m_colegio\\colegio_Dj\\edu\\edu\\applications\\usuarios\\api\\router.py'

9. applications/usuarios/tasks.py (Tarea Celery de email)

No se pudo leer edu\applications\usuarios\tasks.py: [Errno 2] No such file or directory: 'C:\\Users\\EQUIPO\\Documents\\LAMDA\\DJANGO\\UDEMY\\m_colegio\\colegio_Dj\\edu\\edu\\applications\\usuarios\\tasks.py'

Fin de la Guia Completa - EduPro 360 / HU-01

Esta guia documenta TODO el proceso de desarrollo desde la concepcion inicial hasta la implementacion completa y funcional del sistema de registro de usuarios con roles, autenticacion JWT, envio asincrono de emails y documentacion API.

Version: 1.0 (22 de diciembre de 2025)

Para soporte o dudas, revisar el codigo fuente completo incluido en el anexo.