**Guía: Desplegar tu programa en Django (proyecto CDPJ\_1)**

A continuación tienes una guía completa **desde mkdir CDPJ\_1** hasta una app Django mínima que permite: subir 3 archivos Excel (máx 5MB cada uno), ejecutar la lógica de validación (tu script refactorizado), y descargar los archivos generados: validos.xlsx, rechazados.xlsx, y reporte\_validacion.pdf.

**Resumen de la estructura final**

CDPJ\_1/

├─ venv/ (virtualenv opcional)

├─ requirements.txt

├─ manage.py

├─ cdpj\_project/

│ ├─ settings.py

│ ├─ urls.py

│ └─ ...

└─ validator/ # app

├─ models.py

├─ forms.py

├─ views.py

├─ urls.py

├─ templates/validator/upload.html

├─ validators/validacion\_report.py # tu lógica, refactorizada como función

└─ migrations/

**1) Crear proyecto y entorno**

# en tu terminal

mkdir CDPJ\_1

cd CDPJ\_1

python -m venv venv # opcional pero recomendado

# activar venv

# Windows: venv\Scripts\activate

# Linux/macOS: source venv/bin/activate

# instala dependencias (ver requirements abajo)

pip install -r requirements.txt

# crear proyecto y app

django-admin startproject cdpj\_project .

python manage.py startapp validator

**Nota:** incluye requirements.txt con al menos:

django>=4.2

pandas

openpyxl

reportlab

matplotlib

numpy

**2) Ajustes en cdpj\_project/settings.py**

Añade/asegura las siguientes líneas:

# settings.py (fragmentos)

INSTALLED\_APPS += [

'validator',

]

# Media (archivos subidos y generados)

import os

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

# Ajusta tamaño máximo de upload si quieres (opcional)

DATA\_UPLOAD\_MAX\_MEMORY\_SIZE = 10 \* 1024 \* 1024 # 10MB request

FILE\_UPLOAD\_MAX\_MEMORY\_SIZE = 6 \* 1024 \* 1024 # 6MB file

**3) Modelo (opcional)**

Puedes no guardar en DB si sólo quieres procesar en disco. Si quieres registro, usa models.py así:

# validator/models.py

from django.db import models

class UploadSet(models.Model):

creado = models.DateTimeField(auto\_now\_add=True)

bruto = models.FileField(upload\_to='uploads/', null=True, blank=True)

suscriptores = models.FileField(upload\_to='uploads/', null=True, blank=True)

main\_pier = models.FileField(upload\_to='uploads/', null=True, blank=True)

processed = models.BooleanField(default=False)

def \_\_str\_\_(self):

return f"UploadSet {self.id} - {self.creado}"

Si no quieres persistir, omite el modelo y procesa archivos temporales en MEDIA\_ROOT.

**4) Formulario para subir archivos**

# validator/forms.py

from django import forms

class UploadFilesForm(forms.Form):

bruto = forms.FileField(label='Documento a filtrar (bruto)', required=True)

main\_pier = forms.FileField(label='Documento MAIN PIER', required=True)

suscriptores = forms.FileField(label='Documento SUSCRIPTORES PIER', required=True)

def clean(self):

cleaned = super().clean()

for fkey in ['bruto', 'main\_pier', 'suscriptores']:

f = cleaned.get(fkey)

if f:

if f.size > 5 \* 1024 \* 1024:

raise forms.ValidationError(f"El archivo {fkey} supera 5MB")

return cleaned

**5) Refactorizar tu script (validacion\_report.py) para usarlo desde Django**

* Mueve tu archivo validacion\_report.py dentro de validator/validators/validacion\_report.py.
* Extrae la lógica main() en una función llamada run\_validation(bruto\_path, detalles\_path, sus\_path, out\_dir) que devuelva rutas a los archivos generados.

Ejemplo de validator/validators/validacion\_report.py (adaptado, exporta run\_validation):

# validator/validators/validacion\_report.py

# --- pega tu código aquí ---, pero:

# 1) reemplaza constantes WORKDIR y FILE\_\* por parámetros

# 2) extrae una función run\_validation(...)

from pathlib import Path

# (importa el resto de librerías como en tu script)

def run\_validation(path\_bruto, path\_detalles, path\_sus, out\_dir=None):

"""

path\_\*: rutas (str o Path) a los archivos subidos

out\_dir: carpeta donde guardar salidas (si None -> carpeta contenedora de path\_bruto)

Retorna: dict con keys 'validos\_xlsx','rechazados\_xlsx','validos\_csv','rechazados\_csv','pdf'

"""

path\_bruto = Path(path\_bruto)

path\_detalles = Path(path\_detalles)

path\_sus = Path(path\_sus)

if out\_dir is None:

out\_dir = path\_bruto.parent

else:

out\_dir = Path(out\_dir)

out\_dir.mkdir(parents=True, exist\_ok=True)

# define outputs dentro de out\_dir (reusa nombres de tu script)

OUT\_VALID\_XLSX = out\_dir / 'validos.xlsx'

OUT\_RECH\_XLSX = out\_dir / 'rechazados.xlsx'

OUT\_VALID\_CSV = out\_dir / 'validos.csv'

OUT\_RECH\_CSV = out\_dir / 'rechazados.csv'

OUT\_PDF = out\_dir / 'reporte\_validacion.pdf'

# --- aquí copia la lógica de tu main, pero usando las variables locales ---

# recuerda: no uses sys.exit(); levanta excepciones en su lugar si hay errores.

# Al final, retorna rutas:

return {

'validos\_xlsx': str(OUT\_VALID\_XLSX),

'rechazados\_xlsx': str(OUT\_RECH\_XLSX),

'validos\_csv': str(OUT\_VALID\_CSV),

'rechazados\_csv': str(OUT\_RECH\_CSV),

'pdf': str(OUT\_PDF),

}

Importante: reemplaza print() y sys.exit() por excepciones o logging para que Django no termine.

**6) View: recibir archivos, guardarlos, ejecutar validación y entregar links**

# validator/views.py

from django.shortcuts import render

from django.conf import settings

from django.http import HttpResponse, HttpResponseRedirect

from django.urls import reverse

from .forms import UploadFilesForm

from pathlib import Path

import uuid

from .validators.validacion\_report import run\_validation

def upload\_view(request):

context = {}

if request.method == 'POST':

form = UploadFilesForm(request.POST, request.FILES)

if form.is\_valid():

# Carpeta única por proceso

run\_id = uuid.uuid4().hex

out\_dir = Path(settings.MEDIA\_ROOT) / 'processing' / run\_id

out\_dir.mkdir(parents=True, exist\_ok=True)

# Guardar archivos

bruto\_path = out\_dir / request.FILES['bruto'].name

with open(bruto\_path, 'wb+') as dest:

for chunk in request.FILES['bruto'].chunks():

dest.write(chunk)

main\_path = out\_dir / request.FILES['main\_pier'].name

with open(main\_path, 'wb+') as dest:

for chunk in request.FILES['main\_pier'].chunks():

dest.write(chunk)

sus\_path = out\_dir / request.FILES['suscriptores'].name

with open(sus\_path, 'wb+') as dest:

for chunk in request.FILES['suscriptores'].chunks():

dest.write(chunk)

# Ejecutar lógica (sincronamente)

try:

outputs = run\_validation(bruto\_path, main\_path, sus\_path, out\_dir=out\_dir)

except Exception as e:

context['error'] = str(e)

context['form'] = form

return render(request, 'validator/upload.html', context)

# Preparar URLs para descarga

media\_url = settings.MEDIA\_URL.rstrip('/')

base\_web = f"{media\_url}/processing/{run\_id}"

# los archivos se guardan con nombres fijos, así que podemos construir rutas

context['download\_links'] = {

'validos\_xlsx': f"{base\_web}/validos.xlsx",

'rechazados\_xlsx': f"{base\_web}/rechazados.xlsx",

'validos\_csv': f"{base\_web}/validos.csv",

'rechazados\_csv': f"{base\_web}/rechazados.csv",

'pdf': f"{base\_web}/reporte\_validacion.pdf",

}

context['success'] = True

return render(request, 'validator/upload.html', context)

else:

form = UploadFilesForm()

context['form'] = form

return render(request, 'validator/upload.html', context)

**7) URLs**

# validator/urls.py

from django.urls import path

from . import views

urlpatterns = [

path('', views.upload\_view, name='validator\_upload'),

]

# cdpj\_project/urls.py (añadir)

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('', include('validator.urls')),

]

# y al final del archivo:

if settings.DEBUG:

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**8) Template (un solo template upload.html)**

<!-- validator/templates/validator/upload.html -->

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Validación Excel - Upload</title>

</head>

<body>

<h1>Validación - Subir archivos</h1>

{% if error %}

<div style="color: red">Error: {{ error }}</div>

{% endif %}

{% if success %}

<div style="color: green">Proceso completado. Descarga los archivos:</div>

<ul>

<li><a href="{{ download\_links.validos\_xlsx }}">validos.xlsx</a></li>

<li><a href="{{ download\_links.rechazados\_xlsx }}">rechazados.xlsx</a></li>

<li><a href="{{ download\_links.pdf }}">reporte\_validacion.pdf</a></li>

</ul>

<hr/>

{% endif %}

<form method="post" enctype="multipart/form-data">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Ejecutar validación</button>

</form>

<p>Notas: Archivos máximo 5MB. En desarrollo los archivos están en /media/processing/.</p>

</body>

</html>

**9) Migraciones y correr servidor (dev)**

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

Visita: http://127.0.0.1:8000/ y verás el template de subida.

**10) Cosas importantes / Consideraciones de producción**

* **No ejecutar procesamientos muy pesados en requests** en producción: considera usar una cola (Celery / RQ) si el trabajo tarda mucho.
* **Permisos y limpieza**: borra carpetas processing/<run\_id> antiguas si no las necesitas. Implementa un job cron.
* **Almacenamiento**: usar S3 o almacenamiento persistente para archivos en deploy real.
* **Seguridad**: valida extensiones y contenido (xls/xlsx). Escanea por macros si hay riesgo.
* **Tamaño de memoria**: pandas puede consumir RAM con archivos grandes. Considera chunking.

**11) Debugging rápido**

* Si find\_column falla, imprime df.columns temporales para ver nombres exactos.
* Si ves PermissionError al abrir archivos, revisa permisos de MEDIA\_ROOT.
* Para probar localmente, sube archivos pequeños que reproduzcan casos.

**12) ¿Qué más puedo hacer por ti?**

* Puedo:
  + Refactorizar tu validacion\_report.py directamente para que exporte run\_validation() (pega el archivo tal cual y lo convierto).
  + Preparar los archivos completos models.py, views.py y settings listos para copiar/pegar.
  + Añadir la funcionalidad para mostrar previsualización de rechazados dentro del template.

¡Listo! Si quieres que genere ahora los archivos de código listos para pegar (por ejemplo validator/views.py, validator/forms.py, validator/validators/validacion\_report.py refactorizado), dime y los creo aquí mismo.