**📦 Nueva App: presupuestos**

**1️⃣ Crear la app**

python manage.py startapp presupuestos

En settings.py:

INSTALLED\_APPS = [

...

"rest\_framework",

"presupuestos",

]

**2️⃣ Modelo**

**presupuestos/models.py**

from django.db import models

class Presupuesto(models.Model):

producto = models.CharField(max\_length=200)

email = models.EmailField()

mensaje = models.TextField(blank=True)

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

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

return f"Presupuesto para {self.producto} ({self.email})"

Migraciones:

python manage.py makemigrations presupuestos

python manage.py migrate

**3️⃣ Serializer**

**presupuestos/api/serializers.py**

from rest\_framework import serializers

from presupuestos.models import Presupuesto

class PresupuestoSerializer(serializers.ModelSerializer):

class Meta:

model = Presupuesto

fields = ["id", "producto", "email", "mensaje", "creado\_en"]

**4️⃣ Envío a Telegram**

Para mandar el presupuesto a un **grupo o canal**, necesitás un **bot token** y un **chat\_id**.  
Configuralo en settings.py:

# --- Telegram ---

TELEGRAM\_BOT\_TOKEN = "TU\_BOT\_TOKEN"

TELEGRAM\_CHAT\_ID = "TU\_CHAT\_ID" # puede ser grupo o canal

Helper: **presupuestos/utils.py**

import requests

from django.conf import settings

def enviar\_a\_telegram(texto):

token = settings.TELEGRAM\_BOT\_TOKEN

chat\_id = settings.TELEGRAM\_CHAT\_ID

url = f"https://api.telegram.org/bot{token}/sendMessage"

payload = {"chat\_id": chat\_id, "text": texto, "parse\_mode": "HTML"}

requests.post(url, data=payload)

**5️⃣ View de la API**

**presupuestos/api/views.py**

from rest\_framework import generics

from presupuestos.models import Presupuesto

from .serializers import PresupuestoSerializer

from presupuestos.utils import enviar\_a\_telegram

class PresupuestoCreate(generics.CreateAPIView):

queryset = Presupuesto.objects.all()

serializer\_class = PresupuestoSerializer

def perform\_create(self, serializer):

instance = serializer.save()

# Enviar mensaje a Telegram

mensaje = (

f"📢 <b>Nuevo presupuesto recibido</b>\n\n"

f"📦 Producto: {instance.producto}\n"

f"📧 Email: {instance.email}\n"

f"📝 Mensaje: {instance.mensaje}\n"

f"⏰ Fecha: {instance.creado\_en.strftime('%d-%m-%Y %H:%M')}"

)

enviar\_a\_telegram(mensaje)

**6️⃣ URLs**

**presupuestos/api/urls.py**

from django.urls import path

from .views import PresupuestoCreate

urlpatterns = [

path("create/", PresupuestoCreate.as\_view(), name="presupuesto-create"),

]

En urls.py principal:

from django.urls import path, include

urlpatterns = [

...

path("api/presupuestos/", include("presupuestos.api.urls")),

]

**7️⃣ Probar en Swagger**

Levantar servidor:

python manage.py runserver

Ir a 👉 http://127.0.0.1:8000/swagger/

Endpoint disponible:

POST /api/presupuestos/create/

Ejemplo de request:

{

"producto": "Celular Samsung Galaxy",

"email": "cliente@test.com",

"mensaje": "Quiero presupuesto por 5 unidades con envío."

}

➡️ Al crear el presupuesto:

* Se guarda en la BD.
* Se envía el aviso directo al **Telegram** configurado.

**8️⃣ Listado de presupuestos (solo staff/admin)**

📄 **presupuestos/api/views.py**

from rest\_framework import generics, permissions

from presupuestos.models import Presupuesto

from .serializers import PresupuestoSerializer

class IsStaffUser(permissions.BasePermission):

"""Permiso personalizado: solo usuarios staff"""

def has\_permission(self, request, view):

return bool(request.user and request.user.is\_staff)

class PresupuestoList(generics.ListAPIView):

queryset = Presupuesto.objects.all().order\_by("-creado\_en")

serializer\_class = PresupuestoSerializer

permission\_classes = [permissions.IsAuthenticated, IsStaffUser]

📄 **presupuestos/api/urls.py**

from django.urls import path

from .views import PresupuestoCreate, PresupuestoList

urlpatterns = [

path("create/", PresupuestoCreate.as\_view(), name="presupuesto-create"),

path("list/", PresupuestoList.as\_view(), name="presupuesto-list"),

]

🔒 **Protección con autenticación**

Como ya está DRF, basta con habilitar login por **SessionAuth o TokenAuth**.  
En settings.py:

REST\_FRAMEWORK = {

"DEFAULT\_AUTHENTICATION\_CLASSES": [

"rest\_framework.authentication.SessionAuthentication",

"rest\_framework.authentication.BasicAuthentication",

],

"DEFAULT\_PERMISSION\_CLASSES": [

"rest\_framework.permissions.IsAuthenticated",

],

}

Con esto:

* /api/presupuestos/create/ sigue abierto para cualquiera (clientes).
* /api/presupuestos/list/ requiere login con usuario staff (lo verificamos en IsStaffUser).

📊 **En Swagger**

Al entrar a /swagger/:

* Se verá también el endpoint GET /api/presupuestos/list/.
* Al probarlo, pedirá autenticación.
* Solo los usuarios staff podrán ver los presupuestos en el listado.

=============================================================================

* **Django → Telegram**: cuando se crea un presupuesto, llega el aviso al bot.
* **Telegram → Django**: podés mandar comandos (/listar, /presupuesto <id>, /usuarios) y el bot responde.
* **Botones inline**: desde Telegram podés abrir directo la ficha en el admin de Django.
* **Flag en el front**: si el servidor recibió algo desde Telegram, lo muestra en la plantilla.

**🛠️ Funcionalidades posibles desde Telegram**

1. **Listar presupuestos recientes (solo admin/staff)**
   * El bot responde en Telegram con un listado de los últimos 5 presupuestos.
   * Ejemplo:
   * 📦 Celular Samsung Galaxy - cliente@test.com
   * 📦 Notebook Lenovo - ana@mail.com
   * Incluso podés enviar botones inline con links directos a /admin/presupuestos/presupuesto/<id>/change/.
2. **Detalle de un presupuesto por ID**
   * El usuario en Telegram manda:
   * /presupuesto 3
   * El bot busca el presupuesto en la DB y responde con toda la info.
3. **Responder presupuesto desde Telegram**
   * Podrías agregar un campo respondido = BooleanField(default=False) en el modelo.
   * Con un comando /responder 3 el bot marca el presupuesto como respondido en la BD.
4. **Notificaciones más ricas**
   * Cuando llega un presupuesto nuevo, además del mensaje plano, podés mandar **botones interactivos** en Telegram (con InlineKeyboardMarkup).
   * Ejemplo:
   * 📢 Nuevo presupuesto
   * Producto: Celular Samsung Galaxy
   * Email: cliente@test.com
   * Mensaje: Quiero 5 unidades
   * ⏰ Fecha: 30-09-2025 14:33

Y debajo, un botón:  
👉 [Ver en Admin](http://127.0.0.1:8000/admin/presupuestos/presupuesto/3/change/)

**🔗 Cómo integrarlo en Django**

**1. Crear vista webhook**

📄 presupuestos/views.py

import json

from django.http import JsonResponse

from django.views.decorators.csrf import csrf\_exempt

from presupuestos.models import Presupuesto

@csrf\_exempt

def telegram\_webhook(request):

if request.method == "POST":

data = json.loads(request.body.decode("utf-8"))

message = data.get("message", {})

chat\_id = message.get("chat", {}).get("id")

text = message.get("text", "")

# Comando para listar

if text.startswith("/listar"):

presupuestos = Presupuesto.objects.order\_by("-creado\_en")[:5]

respuesta = "\n".join(

[f"📦 {p.producto} - {p.email}" for p in presupuestos]

) or "No hay presupuestos."

enviar\_a\_telegram(respuesta, chat\_id)

# Comando para detalle

elif text.startswith("/presupuesto"):

try:

\_, pid = text.split()

p = Presupuesto.objects.get(id=int(pid))

respuesta = (

f"📦 Producto: {p.producto}\n"

f"📧 Email: {p.email}\n"

f"📝 {p.mensaje}\n"

f"⏰ {p.creado\_en.strftime('%d-%m-%Y %H:%M')}"

)

except Exception:

respuesta = "❌ ID inválido o presupuesto no encontrado."

enviar\_a\_telegram(respuesta, chat\_id)

else:

enviar\_a\_telegram("Comandos: /listar, /presupuesto <id>", chat\_id)

return JsonResponse({"ok": True})

return JsonResponse({"error": "Método no permitido"}, status=405)

**2. Reusar helper para responder**

📄 presupuestos/utils.py

def enviar\_a\_telegram(texto, chat\_id=None):

token = settings.TELEGRAM\_BOT\_TOKEN

if not chat\_id:

chat\_id = settings.TELEGRAM\_CHAT\_ID

url = f"https://api.telegram.org/bot{token}/sendMessage"

payload = {"chat\_id": chat\_id, "text": texto, "parse\_mode": "HTML"}

requests.post(url, data=payload)

**3. URL para Telegram**

📄 presupuestos/urls.py

from django.urls import path

from .views import telegram\_webhook

urlpatterns = [

path("telegram/webhook/", telegram\_webhook, name="telegram-webhook"),

]

**4. Configurar webhook en Telegram**

Ejecutás (una sola vez):

curl -X POST "https://api.telegram.org/bot<YOUR\_TOKEN>/setWebhook" \

-d "url=https://TU\_DOMINIO/api/presupuestos/telegram/webhook/"

🔒 **Nota de seguridad**:  
Conviene filtrar los chat\_id permitidos (ej. solo admin) para que cualquiera no pueda espiar presupuestos.

L**istar los usuarios registrados desde Telegram** dentro de tu mismo proyecto Django, se puede hacer igual que con los presupuestos, aprovechando el webhook.

**🛠️ Listado de usuarios registrados (solo admins en Telegram)**

1. **Obtener usuarios de Django**  
   Django ya tiene el modelo User (en django.contrib.auth.models.User).  
   Podés consultarlo directamente y devolver los últimos registrados, o todos.

**📄 presupuestos/views.py**

Agregamos el comando /usuarios en el webhook:

from django.contrib.auth.models import User

@csrf\_exempt

def telegram\_webhook(request):

if request.method == "POST":

data = json.loads(request.body.decode("utf-8"))

message = data.get("message", {})

chat\_id = message.get("chat", {}).get("id")

text = message.get("text", "")

# --- Listado de presupuestos ---

if text.startswith("/listar"):

presupuestos = Presupuesto.objects.order\_by("-creado\_en")[:5]

respuesta = "\n".join(

[f"📦 {p.producto} - {p.email}" for p in presupuestos]

) or "No hay presupuestos."

enviar\_a\_telegram(respuesta, chat\_id)

# --- Detalle de presupuesto ---

elif text.startswith("/presupuesto"):

try:

\_, pid = text.split()

p = Presupuesto.objects.get(id=int(pid))

respuesta = (

f"📦 Producto: {p.producto}\n"

f"📧 Email: {p.email}\n"

f"📝 {p.mensaje}\n"

f"⏰ {p.creado\_en.strftime('%d-%m-%Y %H:%M')}"

)

except Exception:

respuesta = "❌ ID inválido o presupuesto no encontrado."

enviar\_a\_telegram(respuesta, chat\_id)

# --- Listado de usuarios ---

elif text.startswith("/usuarios"):

usuarios = User.objects.order\_by("-date\_joined")[:5]

respuesta = "\n".join(

[f"👤 {u.username} ({u.email or 'sin email'})" for u in usuarios]

) or "No hay usuarios registrados."

enviar\_a\_telegram(respuesta, chat\_id)

else:

enviar\_a\_telegram("Comandos: /listar, /presupuesto <id>, /usuarios", chat\_id)

return JsonResponse({"ok": True})

return JsonResponse({"error": "Método no permitido"}, status=405)

**🔒 Seguridad**

* Lo ideal es **validar el chat\_id** antes de devolver datos sensibles (usuarios o presupuestos).
* Podés definir en settings.py una lista de TELEGRAM\_ADMINS = [12345678] y comparar:

if chat\_id not in settings.TELEGRAM\_ADMINS:

enviar\_a\_telegram("⛔ No autorizado.", chat\_id)

return JsonResponse({"ok": False})

**📲 Uso en Telegram**

* /usuarios → devuelve los últimos 5 usuarios registrados.
* Ejemplo de respuesta:
* 👤 admin (admin@site.com)
* 👤 juan (juan@test.com)
* 👤 maria (maria@gmail.com)

1️⃣ **Botón en Telegram con link al admin de Django**  
2️⃣ **Indicador en un template de presupuestos cuando el servidor recibe mensajes desde Telegram**

**1️⃣ Botones inline en Telegram (abrir admin Django)**

Telegram soporta **InlineKeyboardMarkup** → botones clicables debajo del mensaje.  
Lo adaptamos al comando /usuarios.

📄 presupuestos/utils.py

import requests

from django.conf import settings

def enviar\_a\_telegram(texto, chat\_id=None, buttons=None):

token = settings.TELEGRAM\_BOT\_TOKEN

if not chat\_id:

chat\_id = settings.TELEGRAM\_CHAT\_ID

url = f"https://api.telegram.org/bot{token}/sendMessage"

payload = {

"chat\_id": chat\_id,

"text": texto,

"parse\_mode": "HTML",

}

# Si vienen botones, los agregamos como markup

if buttons:

payload["reply\_markup"] = {

"inline\_keyboard": buttons

}

requests.post(url, json=payload)

📄 presupuestos/views.py → comando /usuarios ahora con botón:

elif text.startswith("/usuarios"):

usuarios = User.objects.order\_by("-date\_joined")[:3]

if not usuarios:

enviar\_a\_telegram("No hay usuarios registrados.", chat\_id)

else:

for u in usuarios:

texto = f"👤 {u.username} ({u.email or 'sin email'})"

url\_admin = f"{settings.SITE\_URL}/admin/auth/user/{u.id}/change/"

buttons = [[{"text": "🔗 Ver en Admin", "url": url\_admin}]]

enviar\_a\_telegram(texto, chat\_id, buttons=buttons)

👉 settings.SITE\_URL debe ser tu dominio/base del proyecto (ej. http://127.0.0.1:8000).

Ahora cuando escribís /usuarios en Telegram, el bot devuelve una lista con cada usuario y un botón que abre el admin directamente en el navegador.

**2️⃣ Flag en el front cuando llegan mensajes desde Telegram**

Podemos guardar en la base de datos cada vez que se recibe algo desde Telegram.

📄 presupuestos/models.py

class TelegramLog(models.Model):

chat\_id = models.CharField(max\_length=50)

mensaje = models.TextField()

recibido\_en = models.DateTimeField(auto\_now\_add=True)

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

return f"Msg {self.chat\_id} - {self.recibido\_en}"

📄 presupuestos/views.py → dentro del webhook:

from presupuestos.models import TelegramLog

@csrf\_exempt

def telegram\_webhook(request):

if request.method == "POST":

data = json.loads(request.body.decode("utf-8"))

message = data.get("message", {})

chat\_id = str(message.get("chat", {}).get("id"))

text = message.get("text", "")

# Guardamos log en la BD

TelegramLog.objects.create(chat\_id=chat\_id, mensaje=text)

...

**Mostrar flag en template**

📄 presupuestos/views.py (una vista de ejemplo para la lista de presupuestos):

from django.shortcuts import render

from presupuestos.models import Presupuesto, TelegramLog

def lista\_presupuestos(request):

presupuestos = Presupuesto.objects.all().order\_by("-creado\_en")

hay\_telegram = TelegramLog.objects.exists()

return render(request, "presupuestos/lista.html", {

"presupuestos": presupuestos,

"hay\_telegram": hay\_telegram,

})

📄 templates/presupuestos/lista.html

{% extends "base.html" %}

{% block content %}

<h2>Listado de Presupuestos</h2>

{% if hay\_telegram %}

<p style="color:green">✅ El servidor recibió mensajes desde Telegram.</p>

{% else %}

<p style="color:red">⚠️ Aún no llegaron mensajes desde Telegram.</p>

{% endif %}

<ul>

{% for p in presupuestos %}

<li>{{ p.producto }} - {{ p.email }}</li>

{% endfor %}

</ul>

{% endblock %}

**🚀 Resumen**

* /usuarios en Telegram devuelve usuarios con **botón directo al admin**.
* Cada mensaje recibido por el webhook se guarda en TelegramLog.
* En el front (lista.html) tenés un flag que indica si el servidor ya recibió mensajes desde Telegram.

=============================================================================

¡Genial! 🚀 Vamos a mejorar el **flag del front** para que no sea solo un ✅/⚠️, sino que también muestre:

* 📩 el último mensaje recibido,
* 👤 el chat\_id (o username si lo capturamos),
* ⏰ la hora en que llegó.

**1️⃣ Guardar más info en el log**

📄 presupuestos/models.py

class TelegramLog(models.Model):

chat\_id = models.CharField(max\_length=50)

username = models.CharField(max\_length=100, blank=True, null=True)

mensaje = models.TextField()

recibido\_en = models.DateTimeField(auto\_now\_add=True)

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

return f"Msg de {self.username or self.chat\_id} - {self.recibido\_en}"

**2️⃣ Capturar username en el webhook**

📄 presupuestos/views.py

from presupuestos.models import TelegramLog

@csrf\_exempt

def telegram\_webhook(request):

if request.method == "POST":

data = json.loads(request.body.decode("utf-8"))

message = data.get("message", {})

chat = message.get("chat", {})

chat\_id = str(chat.get("id"))

username = chat.get("username") or chat.get("first\_name") or "Desconocido"

text = message.get("text", "")

# Guardamos log en la BD

TelegramLog.objects.create(

chat\_id=chat\_id,

username=username,

mensaje=text,

)

# ... resto de la lógica de comandos ...

return JsonResponse({"ok": True})

return JsonResponse({"error": "Método no permitido"}, status=405)

**3️⃣ Pasar el último log al template**

📄 presupuestos/views.py

def lista\_presupuestos(request):

presupuestos = Presupuesto.objects.all().order\_by("-creado\_en")

ultimo\_log = TelegramLog.objects.order\_by("-recibido\_en").first()

return render(request, "presupuestos/lista.html", {

"presupuestos": presupuestos,

"ultimo\_log": ultimo\_log,

})

**4️⃣ Mostrar en el template**

📄 templates/presupuestos/lista.html

{% extends "base.html" %}

{% block content %}

<h2>Listado de Presupuestos</h2>

{% if ultimo\_log %}

<p style="color:green">

✅ El servidor recibió mensajes desde Telegram.

</p>

<div style="border:1px solid #ccc; padding:10px; border-radius:5px; background:#f8f8f8;">

<p><strong>👤 Usuario:</strong> {{ ultimo\_log.username }} (ID: {{ ultimo\_log.chat\_id }})</p>

<p><strong>📩 Mensaje:</strong> {{ ultimo\_log.mensaje }}</p>

<p><strong>⏰ Recibido:</strong> {{ ultimo\_log.recibido\_en|date:"d-m-Y H:i" }}</p>

</div>

{% else %}

<p style="color:red">⚠️ Aún no llegaron mensajes desde Telegram.</p>

{% endif %}

<hr>

<ul>

{% for p in presupuestos %}

<li>{{ p.producto }} - {{ p.email }}</li>

{% endfor %}

</ul>

{% endblock %}

**🎉 Resultado**

* Si alguien manda un mensaje al bot en Telegram, se guarda en TelegramLog.
* En tu **template de presupuestos**, aparece un recuadro con la info del último mensaje.
* Si todavía no llegó nada, sigue mostrando el aviso ⚠️.

=============================================================================

Sumamos un **endpoint REST** para consultar el último mensaje recibido de Telegram. Así podés usarlo desde AJAX en el front o incluso integrarlo en Swagger.

**1️⃣ Serializer para el log de Telegram**

📄 presupuestos/api/serializers.py

from rest\_framework import serializers

from presupuestos.models import Presupuesto, TelegramLog

class PresupuestoSerializer(serializers.ModelSerializer):

class Meta:

model = Presupuesto

fields = ["id", "producto", "email", "mensaje", "creado\_en"]

class TelegramLogSerializer(serializers.ModelSerializer):

class Meta:

model = TelegramLog

fields = ["id", "chat\_id", "username", "mensaje", "recibido\_en"]

**2️⃣ View para obtener el último mensaje**

📄 presupuestos/api/views.py

from rest\_framework.response import Response

from rest\_framework.views import APIView

from presupuestos.models import TelegramLog

from .serializers import TelegramLogSerializer

class UltimoTelegramLog(APIView):

def get(self, request):

ultimo = TelegramLog.objects.order\_by("-recibido\_en").first()

if not ultimo:

return Response({"detail": "No hay mensajes registrados aún."}, status=404)

return Response(TelegramLogSerializer(ultimo).data)

**3️⃣ URL para el endpoint**

📄 presupuestos/api/urls.py

from django.urls import path

from .views import PresupuestoCreate, PresupuestoList, UltimoTelegramLog

urlpatterns = [

path("create/", PresupuestoCreate.as\_view(), name="presupuesto-create"),

path("list/", PresupuestoList.as\_view(), name="presupuesto-list"),

path("ultimo-telegram/", UltimoTelegramLog.as\_view(), name="ultimo-telegram"),

]

**4️⃣ Uso en Swagger / AJAX**

* En Swagger te va a aparecer:
* GET /api/presupuestos/ultimo-telegram/

Devuelve algo como:

{

"id": 7,

"chat\_id": "123456789",

"username": "juan",

"mensaje": "/listar",

"recibido\_en": "2025-09-30T14:15:22Z"

}

* En el front, podés hacer AJAX:

async function actualizarTelegram() {

const resp = await fetch("/api/presupuestos/ultimo-telegram/");

if (resp.ok) {

const data = await resp.json();

document.getElementById("ultimo-telegram").innerHTML =

`📩 ${data.username || data.chat\_id}: ${data.mensaje} <br> ⏰ ${data.recibido\_en}`;

}

}

setInterval(actualizarTelegram, 5000); // refrescar cada 5s

Y en tu template lista.html solo agregás:

<div id="ultimo-telegram"></div>

Con esto tenés:

* El **flag en el template**.
* El **último mensaje guardado** mostrado en el front.
* Un **endpoint REST** que podés usar en Swagger, Postman o en AJAX para refrescar datos en vivo.