**1) Preparación general**

* Crea 3 apps (opcional, para que quede ordenado):

**python manage.py startapp presence # registro / usuarios conectados**

**python manage.py startapp simple\_chat # chat por polling (sin channels)**

**python manage.py startapp quotes # enviar presupuesto PDF**

* Añade las apps a settings.py en INSTALLED\_APPS:

INSTALLED\_APPS += [

"presence",

"simple\_chat",

"quotes",

]

* Revisa que tengas django.contrib.sessions, django.contrib.auth y django.contrib.messages en INSTALLED\_APPS (vienen por defecto).

**2) Tiempo de sesión (30 minutos) — dónde y cómo**

Colocá estas líneas en settings.py (reemplaza si ya existen):

# settings.py (colocar en la parte de sessions / configuración general)

SESSION\_COOKIE\_AGE = 30 \* 60 # 30 minutos (en segundos)

SESSION\_SAVE\_EVERY\_REQUEST = True # cada request renueva el tiempo

SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE = False

**Notas / colocación**:

* Pones esto directamente en settings.py.
* SESSION\_SAVE\_EVERY\_REQUEST = True asegura que la sesión se renueve con actividad (mide *inactividad* real).
* Con esto ya cerrará la sesión automáticamente al no haber requests por 30 minutos.

**Middleware para logout forzado y redirección a aviso**

Si querés que el usuario sea expulsado y redirigido a una página session\_expired (más control), agrega un middleware:

**presence/middleware.py crearlo no existe**

from django.utils import timezone

from django.shortcuts import redirect

from django.contrib.auth import logout

class AutoLogoutMiddleware:

def \_\_init\_\_(self, get\_response):

self.get\_response = get\_response

def \_\_call\_\_(self, request):

if request.user.is\_authenticated:

now\_ts = timezone.now().timestamp()

last = request.session.get("last\_activity", now\_ts)

# 30 minutos = 1800 segundos

if now\_ts - last > 1800:

logout(request)

# después de logout, opcionalmente redirigir a página informativa

return redirect("session\_expired")

request.session["last\_activity"] = now\_ts

return self.get\_response(request)

**Agregar en settings.py (después de AuthenticationMiddleware):**

MIDDLEWARE = [

# ...

"django.contrib.auth.middleware.AuthenticationMiddleware",

"presence.middleware.AutoLogoutMiddleware",

# ...

]

**Crea una vista/template session\_expired si vas a redirigir.**

**Archivo:** **presence/views.py**

from django.shortcuts import render

def session\_expired(request):

return render(request, "presence/session\_expired.html")

**Archivo: presence/urls.py crearlo no existe**

from django.urls import path

from . import views

urlpatterns = [

path("session-expired/", views.session\_expired, name="session\_expired"),

]

**En el urls.py principal agrega:**

from django.urls import include, path

urlpatterns = [

# ...

path("", include("presence.urls")),

]

**4️⃣ Template de aviso**

Crea el archivo:  
**presence/templates/session\_expired.html**

Contenido básico de ejemplo:

{% extends "base.html" %}

{% block content %}

<h2>Sesión expirada</h2>

<p>Tu sesión se cerró por inactividad. <a href="{% url 'login' %}">Iniciar sesión de nuevo</a>.</p>

{% endblock %}

**3) Registro / presencia de usuarios (quiénes están “en línea”)**

**Objetivo:** llevar un last\_seen por usuario y una vista que muestre usuarios activos (ej.: última actividad en 5 minutos).

**presence/models.py**

from django.db import models

from django.conf import settings

class UserActivity(models.Model):

user = models.OneToOneField(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE)

last\_seen = models.DateTimeField(auto\_now=True)

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

return f"{self.user} - {self.last\_seen}"

**Middleware para actualizar last\_seen**  
presence/middleware.py (puede compartirse con la del AutoLogout o hacerse separado):

from django.utils import timezone

from .models import UserActivity

class UpdateLastSeenMiddleware:

def \_\_init\_\_(self, get\_response):

self.get\_response = get\_response

def \_\_call\_\_(self, request):

if request.user.is\_authenticated:

UserActivity.objects.update\_or\_create(user=request.user, defaults={"last\_seen": timezone.now()})

return self.get\_response(request)

**Agregar en settings.py (luego de AuthenticationMiddleware):**

**MIDDLEWARE += ["presence.middleware.UpdateLastSeenMiddleware"]**

**Vista para mostrar usuarios activos**  
**presence/views.py**

from django.shortcuts import render

from django.utils import timezone

from datetime import timedelta

from .models import UserActivity

def online\_users(request):

cutoff = timezone.now() - timedelta(minutes=5) # considerar "activo" si vió hace 5 min

active = UserActivity.objects.filter(last\_seen\_\_gte=cutoff).select\_related("user")

return render(request, "presence/online\_users.html", {"active": active})

**presence/urls.py**

from django.urls import path

from .views import online\_users

app\_name = "presence"

urlpatterns = [

path("online/", online\_users, name="online-users"),

]

Template presence/templates/presence/online\_users.html

{% extends "base.html" %}

{% block content %}

<h2>Usuarios conectados</h2>

<ul>

{% for ua in active %}

<li>{{ ua.user.username }} — {{ ua.last\_seen|timesince }} ago</li>

{% empty %}

<li>No hay usuarios conectados.</li>

{% endfor %}

</ul>

{% endblock %}

**Migrations( siempre que modificamos los models se usan los siguientes comandos)**

**python manage.py makemigrations presence**

**python manage.py migrate**

**4) Chat (sin channels) — AJAX polling (simple, ideal para clase) Websocket**

**Idea:** guardar mensajes en DB; cliente hace polling cada X segundos para obtener nuevos mensajes.

**simple\_chat/models.py**

from django.db import models

from django.conf import settings

class ChatMessage(models.Model):

user = models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.SET\_NULL, null=True, blank=True)

text = models.TextField()

created\_at = models.DateTimeField(auto\_now\_add=True)

class Meta:

ordering = ["created\_at"]

**simple\_chat/views.py**

from django.shortcuts import render

from django.http import JsonResponse

from django.views.decorators.http import require\_POST

from django.contrib.auth.decorators import login\_required

from .models import ChatMessage

def chat\_view(request):

return render(request, "simple\_chat/chat.html")

def messages\_api(request):

# opcional: soportar ?after\_id=nnn para solo devolver nuevos

after = request.GET.get("after\_id")

qs = ChatMessage.objects.all()

if after:

qs = qs.filter(id\_\_gt=int(after))

msgs = [{"id": m.id, "user": m.user.username if m.user else "Anon", "text": m.text, "created\_at": m.created\_at.isoformat()} for m in qs]

return JsonResponse({"messages": msgs})

@require\_POST

def post\_message\_api(request):

# Usar autenticación si querés solo logueados; aquí permitimos anon también

user = request.user if request.user.is\_authenticated else None

text = request.POST.get("text","").strip()

if not text:

return JsonResponse({"error": "empty"}, status=400)

m = ChatMessage.objects.create(user=user, text=text)

return JsonResponse({"id": m.id, "created\_at": m.created\_at.isoformat()})

**simple\_chat/urls.py**

from django.urls import path

from . import views

app\_name = "simple\_chat"

urlpatterns = [

path("", views.chat\_view, name="chat"),

path("api/messages/", views.messages\_api, name="messages-api"),

path("api/post/", views.post\_message\_api, name="post-api"),

]

**Template simple\_chat/templates/ chat.html**

{% extends "base.html" %}

{% block content %}

<h2>Chat del Mercadito</h2>

<div id="chat-box" style="height: 300px; overflow:auto; border:1px solid #ddd; padding:10px;"></div>

<form id="chat-form">

{% csrf\_token %}

<textarea id="chat-text" rows="2" style="width:100%"></textarea>

<button type="submit" class="btn-primary">Enviar</button>

</form>

<script>

let lastId = 0;

const box = document.getElementById("chat-box");

function renderMessage(m){

const el = document.createElement("div");

el.innerHTML = `<strong>${m.user}</strong>: ${m.text} <small style="color:#999">${new Date(m.created\_at).toLocaleTimeString()}</small>`;

box.appendChild(el);

box.scrollTop = box.scrollHeight;

}

async function poll(){

const res = await fetch("{% url 'simple\_chat:messages-api' %}?after\_id=" + lastId);

const data = await res.json();

data.messages.forEach(m => {

renderMessage(m);

lastId = Math.max(lastId, m.id);

});

}

setInterval(poll, 3000); // cada 3s

poll();

document.getElementById("chat-form").addEventListener("submit", async (e)=>{

e.preventDefault();

const text = document.getElementById("chat-text").value;

if (!text.trim()) return;

const form = new FormData();

form.append("text", text);

form.append("csrfmiddlewaretoken", document.querySelector('[name=csrfmiddlewaretoken]').value);

const resp = await fetch("{% url 'simple\_chat:post-api' %}", {

method: "POST",

body: form

});

if (resp.ok){

document.getElementById("chat-text").value = "";

poll(); // traer los nuevos

}

});

</script>

{% endblock %}

**Notas:**

* **Polling es simple y funciona bien para demostraciones.**
* **Para producción, usar WebSockets/Pusher si necesitás escala y latencia baja.**

**Migrations**

**python manage.py makemigrations simple\_chat**

**python manage.py migrate**

**5) Enviar presupuesto (PDF + email) — app quotes**

**pip install pypdf -> sirve pero hay que armar el pdf con codigo**

**Flujo: el usuario solicita “Enviar presupuesto” para un producto → se genera un PDF con los datos → se envía por correo (al comprador y/o al vendedor).**

Instalá la librería:

pip install xhtml2pdf

(xhtml2pdf convierte HTML a PDF; es fácil para plantillas. Si querés mejor CSS, usar WeasyPrint.)

**quotes/models.py (opcional para registro de presupuestos):**

from django.db import models

from django.conf import settings

from market.models import Product

class QuoteRequest(models.Model):

product = models.ForeignKey(Product, on\_delete=models.CASCADE)

requester = models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.SET\_NULL, null=True, blank=True)

email = models.EmailField()

message = models.TextField(blank=True)

created\_at = models.DateTimeField(auto\_now\_add=True)

**quotes/views.py**

from django.template.loader import render\_to\_string

from django.core.mail import EmailMessage

from django.conf import settings

from django.http import HttpResponseRedirect

from django.shortcuts import get\_object\_or\_404, render

from xhtml2pdf import pisa

import io

from .models import QuoteRequest

from market.models import Product

def request\_quote(request, product\_id):

product = get\_object\_or\_404(Product, pk=product\_id)

if request.method == "POST":

email = request.POST.get("email")

message = request.POST.get("message", "")

qr = QuoteRequest.objects.create(product=product, requester=request.user if request.user.is\_authenticated else None, email=email, message=message)

# render PDF from template

html = render\_to\_string("quotes/quote\_pdf.html", {"product": product, "request": qr})

result = io.BytesIO()

pisa\_status = pisa.CreatePDF(src=html, dest=result)

if not pisa\_status.err:

pdf = result.getvalue()

mail = EmailMessage(

subject=f"Presupuesto para {product.title}",

body="Adjuntamos el presupuesto solicitado.",

from\_email=settings.DEFAULT\_FROM\_EMAIL,

to=[email],

)

mail.attach(f"presupuesto\_{qr.id}.pdf", pdf, "application/pdf")

mail.send()

# redirigir con mensaje

return HttpResponseRedirect("/gracias-presupuesto/")

return render(request, "quotes/request\_quote.html", {"product": product})

**quotes/urls.py**

from django.urls import path

from .views import request\_quote

app\_name = "quotes"

urlpatterns = [

path("solicitar/<int:product\_id>/", request\_quote, name="request-quote"),

]

**Templates:**

* **quotes/templates/request\_quote.html — formulario:**

{% extends "base.html" %}

{% block content %}

<h2>Solicitar presupuesto para {{ product.title }}</h2>

<form method="post">

{% csrf\_token %}

<label>Email</label>

<input type="email" name="email" required>

<label>Mensaje (opcional)</label>

<textarea name="message"></textarea>

<button class="btn-primary" type="submit">Enviar presupuesto</button>

</form>

{% endblock %}

* **quotes/templates/ quote\_pdf.html — plantilla que se convertirá en PDF:**

<!doctype html>

<html>

<head><meta charset="utf-8"><style>

body { font-family: sans-serif; }

.header { text-align:center; margin-bottom:20px; }

.product { border-top:1px solid #ddd; padding-top:10px; }

</style></head>

<body>

<div class="header">

<h1>Presupuesto</h1>

<p>Producto: {{ product.title }}</p>

</div>

<div class="product">

<p><strong>Precio sugerido:</strong> ${{ product.price }}</p>

<p><strong>Marca:</strong> {{ product.marca }}</p>

<p><strong>Descripción:</strong> {{ product.description }}</p>

</div>

<hr>

<p>Solicitado por: {{ request.email }} el {{ request.created\_at }}</p>

<p>Mensaje: {{ request.message }}</p>

</body>

</html>

**Configurar email en settings.py (ejemplo con Gmail SMTP para pruebas — NO usar credenciales reales en repo):**

EMAIL\_BACKEND = "django.core.mail.backends.smtp.EmailBackend"

EMAIL\_HOST = "smtp.gmail.com"

EMAIL\_PORT = 587

EMAIL\_USE\_TLS = True

EMAIL\_HOST\_USER = os.getenv("EMAIL\_HOST\_USER")

EMAIL\_HOST\_PASSWORD = os.getenv("EMAIL\_HOST\_PASSWORD")

DEFAULT\_FROM\_EMAIL = os.getenv("DEFAULT\_FROM\_EMAIL", EMAIL\_HOST\_USER)

**Migrations**

python manage.py makemigrations quotes

python manage.py migrate

**Probar**: entra a /quotes/solicitar/<product\_id>/, completa formulario y comprueba que llega el mail con adjunto PDF.

**6) Rutas principales (añadir al urls.py del proyecto)**

from django.urls import include, path

urlpatterns = [

# ...

path("presence/", include("presence.urls", namespace="presence")),

path("chat/", include("simple\_chat.urls", namespace="simple\_chat")),

path("quotes/", include("quotes.urls", namespace="quotes")),

]

**7) Permisos, seguridad y consejos para la clase**

* **CSRF**: todos los POST en AJAX deben mandar csrfmiddlewaretoken (mi ejemplo del chat lo incluye).
* **Escala**: polling cada 3 s está bien para demo; para producción usar WebSockets.
* **Correo**: en desarrollo usá EMAIL\_BACKEND = 'django.core.mail.backends.console.EmailBackend' para depurar (mira los mails en la consola).
* **PDF**: xhtml2pdf tiene limitaciones CSS; para layouts complejos usar WeasyPrint.
* **No comités de claves**: usa .env o secretos en el host (no subir API keys al repo).
* **Documentación**: crea un README con pasos de instalación y variables de entorno que deben configurar tus alumnos.

**8) Checklist rápido para poner en marcha (comandos)**

# instalar dependencias

pip install xhtml2pdf

# crear apps (si aún no lo hiciste)

python manage.py startapp presence

python manage.py startapp simple\_chat

python manage.py startapp quotes

# registros en settings.py: INSTALLED\_APPS y MIDDLEWARE

# crear migraciones y aplicarlas

python manage.py makemigrations

python manage.py migrate

# crear superuser

python manage.py createsuperuser

# ejecutar servidor

python manage.py runserver

**=======================================================================**

**1️⃣ Swagger listo por app**

**settings.py**

Instalá y agregá las librerías:

pip install drf-yasg

# settings.py

INSTALLED\_APPS = [

...

'rest\_framework',

'drf\_yasg',

]

**urls.py (en la carpeta principal del proyecto)**

from django.contrib import admin

from django.urls import path, include

from rest\_framework import permissions

from drf\_yasg.views import get\_schema\_view

from drf\_yasg import openapi

schema\_view = get\_schema\_view(

openapi.Info(

title="Mi API",

default\_version="v1",

description="Documentación de la API con Swagger",

contact=openapi.Contact(email="soporte@midominio.com"),

),

public=True,

permission\_classes=[permissions.AllowAny],

)

urlpatterns = [

path('admin/', admin.site.urls),

path('api/', include('mi\_app.urls')), # incluís aquí las urls de cada app

path('swagger/', schema\_view.with\_ui('swagger', cache\_timeout=0), name='schema-swagger-ui'),

path('redoc/', schema\_view.with\_ui('redoc', cache\_timeout=0), name='schema-redoc'),

]

Ahora **cada app** solo necesita tener sus urls.py con los endpoints. Swagger los detecta automáticamente.

**2️⃣ Cierre de sesión por inactividad (30 min)**

Agregá en settings.py:

# Tiempo de expiración de sesión: 30 minutos de inactividad

SESSION\_COOKIE\_AGE = 30 \* 60 # 30 minutos en segundos

SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE = True

SESSION\_SAVE\_EVERY\_REQUEST = False # la sesión se renueva solo si hay actividad

Con esto:

* Si un usuario pasa **30 min sin actividad**, la sesión expira automáticamente.
* Si cierra el navegador, la sesión también se elimina.

**3️⃣ Extra opcional: middleware para avisar**

Si querés asegurarte de que se renueve solo cuando haya actividad “real” (ej: peticiones AJAX o vistas), podés activar:

SESSION\_SAVE\_EVERY\_REQUEST = True

Esto extiende la sesión en cada request del usuario que llegue al servidor.

**Resumen de dónde pegar cada snippet**

| **Archivo** | **Snippet** |
| --- | --- |
| settings.py | INSTALLED\_APPS (drf\_yasg), opciones de sesión. |
| urls.py (proyecto) | schema\_view y rutas swagger/ y redoc/. |
| urls.py de cada app | Solo las rutas de la app, Swagger las recoge solo. |

Con estos cambios tenés:

* **Swagger/Redoc** funcionando para **todas las apps** sin repetir código.
* **Expiración de sesión** automática a los 30 min de inactividad en todo el proyecto.