**App Django completa** (market\_ai) que integra **los 3 módulos IA**

🔸Sugeridor de precios

🔸chatbot asistente

🔸Recomendador por embeddings usando **Gemini / Google Gen AI**.

Incluyo: instalación, configuración, código de la app (models, views, forms, urls), templates y señales para generar embeddings cuando se publica/edita un producto. También dejo ejemplos de prompts y cómo probarlo.

Antes de arrancar: usar la **Google Gen AI SDK (google-genai)** para llamar a Gemini (puede usarse vía API key o vía Vertex AI en Google Cloud). Documentación y guía de instalación en los enlaces oficiales que usé para armar esto: instalación pip install google-genai y quickstart. [PyPI](https://pypi.org/project/google-genai/?utm_source=chatgpt.com)[Google AI for Developers](https://ai.google.dev/gemini-api/docs/quickstart?utm_source=chatgpt.com)

Para embeddings y la API de Gemini/Vertex AI consulté las páginas oficiales de embeddings y generación de contenido. [Google Cloud](https://cloud.google.com/vertex-ai/generative-ai/docs/embeddings/get-text-embeddings?utm_source=chatgpt.com)[Google AI for Developers](https://ai.google.dev/gemini-api/docs/text-generation?utm_source=chatgpt.com)

**1) Requisitos e instalación (rápido)**

Desde tu virtualenv:

pip install google-genai

Configurar credenciales (dos opciones):

* **Gemini Developer API / AI Studio (API key)** — exporta la variable GEMINI\_API\_KEY o GOOGLE\_API\_KEY. Ejemplo:

export GEMINI\_API\_KEY="tu\_api\_key\_aqui"

* **Vertex AI (Google Cloud)** — si vas a usar Vertex AI en un proyecto de Google Cloud: configura ADC o variables env:

export GOOGLE\_GENAI\_USE\_VERTEXAI=true

export GOOGLE\_CLOUD\_PROJECT="tu-proyecto"

export GOOGLE\_CLOUD\_LOCATION="us-central1" # o la región que uses

# y autenticar con gcloud auth application-default login o service account

(La SDK detecta automáticamente la forma correcta; docs oficiales con ejemplos de cliente y opciones en la SDK). [Google Cloud](https://cloud.google.com/vertex-ai/generative-ai/docs/sdks/overview?utm_source=chatgpt.com)[Google AI for Developers](https://ai.google.dev/gemini-api/docs/quickstart?utm_source=chatgpt.com)

**2) Crear la app Django**

Desde la raíz del proyecto:

python manage.py startapp market\_ai

Agregar 'market\_ai' a INSTALLED\_APPS en settings.py.

Si aún no lo hiciste, agrega MEDIA\_ROOT y MEDIA\_URL y en urls.py principal sirve static/media en desarrollo.

**3) Estructura de archivos que voy a proponer**

myclase/

market/ # tu app con Product (existente)

market\_ai/ # nueva app IA creada en el punto anterios

\_\_init\_\_.py

admin.py

apps.py

forms.py

gemini\_client.py

models.py

signals.py

templates/

price\_suggest.html

ai\_chat.html

recommendations.html

urls.py

views.py

templates/

base.html

settings.py

urls.py

**4) Código (archivos clave)**

**IMPORTANTE**: adapta nombres de templates, rutas y CSS a tu proyecto. Yo doy una base funcional.

**market\_ai/gemini\_client.py — wrapper para llamar a Gemini (google-genai)**

# market\_ai/gemini\_client.py

import os

import logging

from google import genai

from google.genai import types

logger = logging.getLogger(\_\_name\_\_)

def get\_client():

"""

Crea un cliente genai según si hay API key o se usa Vertex AI.

"""

api\_key = os.environ.get("GEMINI\_API\_KEY") or os.environ.get("GOOGLE\_API\_KEY")

use\_vertex = os.environ.get("GOOGLE\_GENAI\_USE\_VERTEXAI", "false").lower() in ("1","true","yes")

if use\_vertex:

project = os.environ.get("GOOGLE\_CLOUD\_PROJECT")

location = os.environ.get("GOOGLE\_CLOUD\_LOCATION", "us-central1")

client = genai.Client(vertexai=True, project=project, location=location)

else:

if not api\_key:

raise ValueError("GEMINI\_API\_KEY no encontrada en variables de entorno")

client = genai.Client(api\_key=api\_key)

return client

\_client = None

def client():

global \_client

if \_client is None:

\_client = get\_client()

return \_client

# Generación de texto corregida

def generate\_text(prompt, model="gemini-1.5-flash", max\_output\_tokens=300):

c = client()

try:

# Configuración correcta para la nueva API

config = types.GenerateContentConfig(

max\_output\_tokens=max\_output\_tokens,

temperature=0.7

)

# Llamada corregida

response = c.models.generate\_content(

model=model,

contents=prompt,

config=config

)

# Extraer el texto de la respuesta

if hasattr(response, 'text'):

return response.text

elif hasattr(response, 'candidates') and response.candidates:

return response.candidates[0].content.parts[0].text

else:

return str(response)

except Exception as e:

logger.exception("Error al llamar a Gemini generate\_content")

return f"Error al generar texto: {e}"

# Generar embedding corregido

def embed\_text(text, model="text-embedding-004"):

c = client()

try:

response = c.models.embed\_content(

model=model,

contents=text

)

if hasattr(response, 'embeddings'):

embeddings = response.embeddings

if isinstance(embeddings, list) and len(embeddings) > 0:

return embeddings[0].values if hasattr(embeddings[0], 'values') else embeddings[0]

return embeddings

return response

except Exception as e:

logger.exception("Error al generar embedding")

return None

(la función generate\_content y embed\_content vienen de la SDK google-genai según docs). [Google AI for Developers](https://ai.google.dev/gemini-api/docs/text-generation?utm_source=chatgpt.com)[datastax.com](https://www.datastax.com/blog/how-to-create-vector-embeddings-in-python?utm_source=chatgpt.com)

**market\_ai/models.py — para cachear embeddings (opcional pero recomendado)**

# market\_ai/models.py

from django.db import models

class ProductEmbedding(models.Model):

# enlaza con tu modelo Product (ajusta el import si hace falta)

from django.conf import settings

# usa string para evitar import circular

product = models.OneToOneField('market.Product', on\_delete=models.CASCADE, related\_name="embedding",)

model = models.CharField(max\_length=100, default="gemini-embedding-001")

vector = models.JSONField() # guardamos lista de floats

updated\_at = models.DateTimeField(auto\_now=True)

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

return f"Embedding {self.product\_id} ({self.model})"

**market\_ai/signals.py — generar embedding al guardar producto**

# market\_ai/signals.py

from django.db.models.signals import post\_save

from django.dispatch import receiver

from market.models import Product

from .models import ProductEmbedding

from .gemini\_client import embed\_text

@receiver(post\_save, sender=Product)

def compute\_product\_embedding(sender, instance, created, \*\*kwargs):

# crea o actualiza embedding (puede tardar, tenelo en cuenta)

text = f"{instance.title}. {instance.description or ''}. Marca: {instance.marca or ''}"

emb = embed\_text(text)

if emb:

obj, \_ = ProductEmbedding.objects.update\_or\_create(

product=instance,

defaults={"vector": emb}

)

Nota: llamar al API en sincronía en post\_save puede retrasar la respuesta al usuario. Para producción conviene usar un task queue (Celery / RQ) y procesar embebidos en background. Aquí lo dejo simple para que lo entiendas y funcione localmente.

**market\_ai/forms.py**

# market\_ai/forms.py

from django import forms

from market.models import Product

class PriceSuggestForm(forms.Form):

title = forms.CharField(max\_length=200)

description = forms.CharField(widget=forms.Textarea, required=False)

marca = forms.CharField(max\_length=100, required=False)

current\_price = forms.DecimalField(max\_digits=12, decimal\_places=2, required=False)

class ChatForm(forms.Form):

message = forms.CharField(widget=forms.Textarea(attrs={"rows":2}), label="Tu mensaje")

**market\_ai/views.py — vistas principales**

# market\_ai/views.py

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

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

from .forms import PriceSuggestForm, ChatForm

from .gemini\_client import generate\_text, embed\_text

from market.models import Product

from .models import ProductEmbedding

import math

def price\_suggest(request):

sugerencia = None

prompt\_example = (

"Eres un asistente que sugiere un precio justo para un producto en un marketplace argentino. "

"Recibís título, descripción y marca. Devuelve solo un número entero (sin moneda) y una breve razón."

)

if request.method == "POST":

form = PriceSuggestForm(request.POST)

if form.is\_valid():

data = form.cleaned\_data

prompt = (

f"{prompt\_example}\n\n"

f"Titulo: {data['title']}\n"

f"Descripcion: {data['description']}\n"

f"Marca: {data['marca']}\n"

f"Precio actual (si hay): {data.get('current\_price')}\n\n"

"Devuelve: Precio sugerido y una explicación corta."

)

respuesta = generate\_text(prompt)

sugerencia = respuesta

else:

form = PriceSuggestForm()

return render(request, "market\_ai/price\_suggest.html", {"form": form, "sugerencia": sugerencia})

def ai\_chat(request):

# simple chat basado en session (no persistente)

if "ai\_chat\_history" not in request.session:

request.session["ai\_chat\_history"] = []

history = request.session["ai\_chat\_history"]

if request.method == "POST":

form = ChatForm(request.POST)

if form.is\_valid():

user\_msg = form.cleaned\_data["message"]

# armamos prompt con historia corta

system = "Sos un asistente amablemente orientado a ayudar en un marketplace (publicar, comprar, trueque). Responde en español."

accumulated = system + "\n\n"

for turn in history[-6:]:

accumulated += f"Usuario: {turn['user']}\nAsistente: {turn['ai']}\n"

accumulated += f"Usuario: {user\_msg}\nAsistente: "

ai\_resp = generate\_text(accumulated)

# guardamos en history

history.append({"user": user\_msg, "ai": ai\_resp})

request.session["ai\_chat\_history"] = history

request.session.modified = True

else:

form = ChatForm()

return render(request, "market\_ai/ai\_chat.html", {"form": form, "history": history})

def recommend\_similar(request, pk):

# recomienda productos similares por embeddings

producto = get\_object\_or\_404(Product, pk=pk, active=True)

try:

# intentamos usar embeddings guardados

target = producto.embedding.vector

except Exception:

# si no hay embedding guardado, generamos en vuelo

text = f"{producto.title}. {producto.description or ''}"

target = embed\_text(text)

# calculo de similitud simple con todos los product embeddings

candidates = ProductEmbedding.objects.exclude(product=producto)

results = []

import numpy as np

tvec = np.array(target, dtype=float)

for c in candidates:

vec = np.array(c.vector, dtype=float)

# coseno

cos = float(np.dot(tvec, vec) / (np.linalg.norm(tvec) \* np.linalg.norm(vec)))

results.append((c.product, cos))

results.sort(key=lambda x: x[1], reverse=True)

top = [p for p,score in results[:6]]

return render(request, "market\_ai/recommendations.html", {"product": producto, "recommended": top})

En recommend\_similar uso ProductEmbedding (cacheada) para rapidez; si no existe, genero embedding en vuelo. Para un dataset real te conviene usar un **vector DB** (Pinecone, Milvus, etc.) pero para clase esto alcanza.

**market\_ai/urls.py**

# market\_ai/urls.py

from django.urls import path

from . import views

app\_name = "market\_ai"

urlpatterns = [

path("price-suggest/", views.price\_suggest, name="price-suggest"),

path("chat/", views.ai\_chat, name="ai-chat"),

path("recommend/<int:pk>/", views.recommend\_similar, name="recommend-similar"),

]

En tu urls.py de myclase el principal proyecto añade:

path("ai/", include("market\_ai.urls", namespace="market\_ai")),

**Templates (básicos)**

**templates/ price\_suggest.html**

{% extends "base.html" %}

{% block content %}

<h2>Sugeridor de precio (IA)</h2>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

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

</form>

{% if sugerencia %}

<div class="ai-result">

<h3>Resultado de Gemini</h3>

<pre>{{ sugerencia }}</pre>

</div>

{% endif %}

{% endblock %}

**templates/market\_ai/ai\_chat.html**

{% extends "base.html" %}

{% block content %}

<h2>Chat asistente del Mercadito (IA)</h2>

<div class="chat-box">

{% for turn in history %}

<div class="chat-user"><strong>Tú:</strong> {{ turn.user }}</div>

<div class="chat-ai"><strong>Asistente:</strong> {{ turn.ai }}</div>

<hr>

{% endfor %}

</div>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

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

</form>

{% endblock %}

**templates/recommendations.html**

{% extends "base.html" %}

{% block content %}

<h2>Productos similares a "{{ product.title }}"</h2>

<div class="card-grid">

{% for p in recommended %}

<div class="card">

<h3>{{ p.title }}</h3>

<p>{{ p.description|truncatewords:12 }}</p>

<p><strong>${{ p.price }}</strong></p>

<a href="{% url 'market:product-detail' p.pk %}" class="btn-primary">Ver</a>

</div>

{% empty %}

<p>No se encontraron recomendaciones.</p>

{% endfor %}

</div>

{% endblock %}

Ajustá los links (por ejemplo market:product-detail) según tus urls en market.

**5) Integración con tu modelo Product**

Si aún no añadiste image y stock, te recomiendo aplicar los cambios que conversamos antes (stock, image, is\_available). Si lo hacés, recuerda ejecutar:

python manage.py makemigrations

python manage.py migrate

Luego crea un superuser y entra a admin para probar.

**6) Registrar señales y admin**

En market\_ai/apps.py añadir ready para registrar señales:

# market\_ai/apps.py

from django.apps import AppConfig

class MarketAiConfig(AppConfig):

default\_auto\_field = 'django.db.models.BigAutoField'

name = 'market\_ai'

def ready(self):

import market\_ai.signals # registra el signal

En market\_ai/admin.py registra ProductEmbedding:

from django.contrib import admin

from .models import ProductEmbedding

admin.site.register(ProductEmbedding)

**7) Pruebas rápidas y consejos**

1. **Probar local**: asegura GEMINI\_API\_KEY configurada y ejecuta la view ai/price-suggest/.
2. **Logs**: si la llamada a Gemini falla, mira runserver logs (mi wrapper captura excepciones).
3. **Límites/costos**: Gemini/Vertex AI tiene políticas de uso y costos — revisá consola y cuota antes de hacer muchas llamadas. [Google AI for Developers](https://ai.google.dev/gemini-api/docs/quickstart?utm_source=chatgpt.com)
4. **Producción**: evita llamadas sincronas desde signals; mejor usar Celery/RQ para calcular embeddings.
5. **Seguridad**: NO comitees la API key en Git. Usá variables de entorno o Secret Manager.

**8) Fuentes / documentación que usé (útiles para profundizar)**

* Quickstart y librería Python google-genai (instalación y ejemplos). [Google AI for Developers](https://ai.google.dev/gemini-api/docs/quickstart?utm_source=chatgpt.com)[PyPI](https://pypi.org/project/google-genai/?utm_source=chatgpt.com)
* Ejemplos de generación de texto (generate\_content) con Gemini. [Google AI for Developers](https://ai.google.dev/gemini-api/docs/text-generation?utm_source=chatgpt.com)
* Documentación oficial de embeddings / gemini-embedding-001 (cómo generar y usar embeddings). [Google Cloud](https://cloud.google.com/vertex-ai/generative-ai/docs/embeddings/get-text-embeddings?utm_source=chatgpt.com)