**0) Requisitos previos (local)**

* Tener repo en **GitHub** (o GitLab/Bitbucket).
* requirements.txt actualizado (pip freeze > requirements.txt).
* Archivos de tu proyecto subidos (incluye manage.py, carpeta del proyecto con settings.py, wsgi.py, apps).
* Añade estas dependencias mínimas al requirements.txt:
* Django
* gunicorn
* psycopg2-binary
* dj-database-url
* whitenoise[brotli]
* mercadopago # si usás SDK

Ajustá versiones según tu entorno.

**1) Cambios en settings.py (producción-friendly)**

Añadí (o modificá) estas partes en tu settings.py:

import os

import dj\_database\_url

from pathlib import Path

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

# DEBUG cuando no esté en Render (en Render setearás env var)

DEBUG = 'RENDER' not in os.environ

# ALLOWED\_HOSTS: usa el host que Render provee en RENDER\_EXTERNAL\_HOSTNAME

RENDER\_EXTERNAL\_HOSTNAME = os.environ.get("RENDER\_EXTERNAL\_HOSTNAME")

if RENDER\_EXTERNAL\_HOSTNAME:

ALLOWED\_HOSTS = [RENDER\_EXTERNAL\_HOSTNAME]

else:

ALLOWED\_HOSTS = ["127.0.0.1", "localhost"]

# DATABASES via DATABASE\_URL (Render te dará la URL)

DATABASES = {

"default": dj\_database\_url.config(default=os.environ.get("DATABASE\_URL"), conn\_max\_age=600)

}

# Static files — WhiteNoise

STATIC\_URL = "/static/"

if not DEBUG:

STATIC\_ROOT = os.path.join(BASE\_DIR, "staticfiles")

STATICFILES\_STORAGE = "whitenoise.storage.CompressedManifestStaticFilesStorage"

MIDDLEWARE = [

"django.middleware.security.SecurityMiddleware",

"whitenoise.middleware.WhiteNoiseMiddleware", # <- justo después de SecurityMiddleware

# ...

]

* **Por qué**: usar DATABASE\_URL y WhiteNoise es la práctica recomendada en Render. [Render+1](https://render.com/docs/deploy-django)

**2) Static + media (uploads)**

* **Static (CSS/JS/ico):** usaremos WhiteNoise + collectstatic (lo copiaremos a staticfiles en el build). Esto hace que tu app sirva los static. [Render+1](https://render.com/docs/deploy-django)
* **Media (imágenes subidas por usuarios / avatars / productos):** **no** uses el disco local en producción (Render permite discos persistentes pero no es lo ideal para escalado). Lo recomendado es usar **S3 (django-storages)** u otro almacenamiento externo para media. Si estás empezando y aceptás limitaciones, podés montar un **Persistent Disk** en Render, pero para producción real te sugiero S3. [testdriven.io+1](https://testdriven.io/blog/django-render/?utm_source=chatgpt.com)

**3) build.sh (script de build)**

Crea en la raíz build.sh y commitéalo:

#!/usr/bin/env bash

set -o errexit

# instalar dependencias (Render hace pip install automáticamente si lo especificás en build)

pip install -r requirements.txt

# migraciones

python manage.py migrate --noinput

# collectstatic

python manage.py collectstatic --noinput

Hazlo ejecutable:

chmod +x build.sh

Render sugiere usar un build script para migraciones/collectstatic. [Render](https://render.com/docs/deploy-django)

**4) start command (cómo arranca la app)**

En la configuración del Web Service en Render pon el **Start Command** (reemplazá myclase por tu proyecto):

gunicorn myclase.wsgi:application --bind 0.0.0.0:$PORT --workers $WEB\_CONCURRENCY

Y en Environment podés definir WEB\_CONCURRENCY=2 o 3 según plan.

Alternativa que muestra Render (ASGI) si usas uvicorn/ASGI; lo normal para Django es la línea con wsgi.

**5) render.yaml (opcional, automático)**

Si preferís declarar infra en repo, un ejemplo mínimo (reemplazá nombres):

databases:

- name: myclase-db

plan: free

services:

- type: web

name: myclase-web

env: python

plan: free

repo: https://github.com/tu-usuario/tu-repo

branch: main

buildCommand: './build.sh'

startCommand: 'gunicorn myclase.wsgi:application --bind 0.0.0.0:$PORT --workers $WEB\_CONCURRENCY'

envVars:

- key: DATABASE\_URL

fromDatabase:

name: myclase-db

property: connectionString

- key: SECRET\_KEY

generateValue: true

- key: WEB\_CONCURRENCY

value: "2"

Render provee esta opción (render.yaml) para infra-as-code. [Render](https://render.com/docs/deploy-django)

**6) Pasos en el Dashboard de Render (manualmente)**

1. **Crear la DB**: Dashboard → **New** → **Postgres** → crear (guardá la DATABASE\_URL). [Render](https://render.com/docs/postgresql-creating-connecting?utm_source=chatgpt.com)
2. **Nuevo Web Service**: New → Web Service → conectar a tu repo GitHub → elegir branch.
   * **Build Command**: ./build.sh
   * **Start Command**: gunicorn myclase.wsgi:application --bind 0.0.0.0:$PORT --workers $WEB\_CONCURRENCY
   * **Environment**: Python 3 (podés elegir versión)
3. **Environment Variables**: en la pestaña *Environment* agrega:
   * DATABASE\_URL → (si no lo pusiste por render.yaml) la URL de la DB creada.
   * SECRET\_KEY → generá una segura (Render puede generarla).
   * DEBUG → False (o elimina para que DEBUG = 'RENDER' not in os.environ funcione).
   * MERCADOPAGO\_ACCESS\_TOKEN → tu token de MP.
   * PUBLIC\_KEY o MERCADOPAGO\_PUBLIC\_KEY → si usás en JS.
   * WEB\_CONCURRENCY → 2 (opcional).  
     Render tiene un panel para variables y grupos de env vars. [Render](https://render.com/docs/configure-environment-variables?utm_source=chatgpt.com)
4. **Deploy**: Guardá y deployá. Mirá los logs en la página del servicio (build y runtime logs).

**7) Después del primer deploy**

* Abrí **Render Shell** (Dashboard → tu servicio → Shell) y creá superuser:
* python manage.py createsuperuser
* Si algo falla en build (por ejemplo collectstatic con ManifestStaticFilesStorage), podés:
  + Ejecutar python manage.py collectstatic localmente para detectar rutas faltantes, o
  + Usar STATICFILES\_STORAGE='whitenoise.storage.CompressedStaticFilesStorage' temporalmente para debug.  
    Render Docs y la comunidad tienen notas sobre problemas comunes con collectstatic. [Render+1](https://render.com/docs/deploy-django)

**8) Manejo de media uploads (avatars/product images)**

* **Recomendado (producción):** usar **AWS S3 + django-storages** (configurar DEFAULT\_FILE\_STORAGE). Hay tutoriales claros para esto. [testdriven.io](https://testdriven.io/blog/storing-django-static-and-media-files-on-amazon-s3/?utm_source=chatgpt.com)
* **Temporal / pequeño proyecto**: podés usar un **Render Persistent Disk** y montar MEDIA\_ROOT ahí, pero **tené en cuenta** que no escala horizontalmente igual que S3. (Útil para pruebas). [Render+1](https://community.render.com/t/configure-django-to-use-render-disk/778?utm_source=chatgpt.com)

**9) Integración Mercado Pago**

* **Variable de entorno**: MERCADOPAGO\_ACCESS\_TOKEN en Render.
* En tu vista ya usás settings.MERCADOPAGO\_ACCESS\_TOKEN — perfecto. Solo asegurate de poner esa env var en Render antes del deploy (o en render.yaml).
* Si usás la SDK en server-side, no exponer tokens. Para el JS, usá la public key (env var) que pones en plantilla.

**10) Checklist final antes de deploy**

* requirements.txt actualizado.
* build.sh en la raíz y ejecutable.
* settings.py adaptado (dj-database-url + whitenoise + ALLOWED\_HOSTS).
* STATIC\_ROOT definido y collectstatic probado localmente.
* Repo subido a GitHub.
* Crear Postgres en Render y apuntar DATABASE\_URL.
* Añadir SECRET\_KEY, MERCADOPAGO\_ACCESS\_TOKEN y PUBLIC\_KEY en Environment.
* Start command configurado: gunicorn myclase.wsgi:application --bind 0.0.0.0:$PORT --workers $WEB\_CONCURRENCY.
* Comprobar logs de build y runtime en Render. [Render+1](https://render.com/docs/deploy-django)

**Enlaces útiles (de lectura rápida)**

* Deploy Django en Render (guía oficial). [Render](https://render.com/docs/deploy-django)
* WhiteNoise (configurar staticfiles). [whitenoise.readthedocs.io](https://whitenoise.readthedocs.io/en/stable/django.html?utm_source=chatgpt.com)
* Crear Postgres en Render. [Render](https://render.com/docs/postgresql-creating-connecting?utm_source=chatgpt.com)
* Variables de entorno en Render. [Render](https://render.com/docs/configure-environment-variables?utm_source=chatgpt.com)
* Recomendación sobre media files / S3 (por qué usar S3). [testdriven.io+1](https://testdriven.io/blog/django-render/?utm_source=chatgpt.com)