# **Estandarización de recetas**

NOMBRE: Cristobal Estay Pozo

CARRERA: ingeniería informática

ASIGNATURA: Programación Back End

PROFESOR: Javier Ahumada Cortes

Indice

[**Estandarización de recetas** 1](#_Toc210519130)

[1 Introducción 3](#_Toc210519131)

[2 Conexión a BD y configuración 4](#_Toc210519132)

[2.1 receta/settings.py (fragmento con .env + DATABASES) 4](#_Toc210519133)

[2.2 .env (archivo en la raíz, junto a manage.py) 6](#_Toc210519134)

[3 Migraciones (aplicación en BD) 7](#_Toc210519135)

[4 Modelos 8](#_Toc210519136)

[4.1 recetas/models.py — Organización / Zonas / Dispositivos 8](#_Toc210519137)

[4.2 recetas/models.py — Reglas y relación through con umbrales 9](#_Toc210519138)

[4.3 recetas/models.py — Organización / Zonas / Dispositivos 10](#_Toc210519139)

[4.4 recetas/models.py — Series: Measurement 11](#_Toc210519140)

[5 Admin básico (registro + columnas) 12](#_Toc210519141)

[5.1 Category y Receta 12](#_Toc210519142)

[5.2 AlertRule y ProductAlertRule (con min/max): 13](#_Toc210519143)

[5.3 Organization / Zone / Device: 13](#_Toc210519144)

[5.4 Measurement (series con date\_hierarchy): 14](#_Toc210519145)

[6 Semillas 14](#_Toc210519146)

[7 Git — Rama de avance 17](#_Toc210519147)

[8 README del repositorio 17](#_Toc210519148)

[9 Conclusion 18](#_Toc210519149)

# Introducción

El presente documento expone el avance del proyecto **Estandarización de Recetas** sobre el framework **Django**. El objetivo de este hito es dejar el sistema **navegable desde el Django Admin**, demostrando la correcta **conexión a base de datos**, la aplicación de **migraciones**, la carga de **catálogo inicial** (semillas) y la habilitación de un **admin básico** con columnas, filtros, búsquedas y ordenamientos útiles, de acuerdo con la pauta de evaluación.

Para este avance se adoptó **SQLite** como motor de datos en ambiente local y se parametrizó la configuración mediante variables definidas en un archivo **.env**, lo que permite intercambiar el motor (p. ej., MySQL) sin modificar código. A nivel de dominio, se modelaron las entidades requeridas por la pauta (o sus equivalencias): **Category**, **Product** (mapeado a Receta), **AlertRule**, **ProductAlertRule** (relación *through* con umbrales mínimo/máximo), así como **Organization**, **Zone**, **Device** y la serie temporal **Measurement** para evidenciar date\_hierarchy.

La inicialización de datos se resolvió con un **management command** (seed\_catalog\_es) que carga categorías, reglas de alerta y una organización con sus zonas y dispositivos; además, vincula recetas con reglas mediante los umbrales correspondientes y registra mediciones de ejemplo. Finalmente, se documenta el **control de versiones** en una rama dedicada (**u2-c2-admin-basico**) y se incluyen **capturas de consola y del Admin** como evidencia del correcto funcionamiento.

**Alcance del documento**

* Configuración de BD vía .env y migraciones aplicadas.
* Modelado de catálogo y operacionales (Category, Receta≡Product, AlertRule, ProductAlertRule, Organization, Zone, Device, Measurement).
* Semillas de datos mediante comando gestionado.
* Admin básico con list\_display, search\_fields, list\_filter, ordering, list\_select\_related y date\_hierarchy.
* Evidencias de ejecución y rama de Git utilizada para el avance

# Conexión a BD y configuración

El proyecto utiliza **SQLite** para desarrollo. La configuración de base de datos se parametriza mediante **variables de entorno** definidas en un archivo .env, cumpliendo la pauta solicitada.

## receta/settings.py (fragmento con .env + DATABASES)

"""

Django settings for receta project.

Generated by 'django-admin startproject' using Django 5.2.6.

"""

from pathlib import Path

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

SECRET\_KEY = 'django-insecure-2ti@ik2e+gug@fj-#-+23e#ks+dn&y6@=\*@h\_x#\*w5w57a75!%'

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'recetas',

    'accounts.apps.AccountsConfig',  # <-- agregado para login/registro

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'receta.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [BASE\_DIR / "templates"],  # <-- agregado: carpeta global de templates

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

WSGI\_APPLICATION = 'receta.wsgi.application'

# Database

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': BASE\_DIR / 'db.sqlite3',

    }

}

# Password validation

AUTH\_PASSWORD\_VALIDATORS = [

    {

        'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

    },

]

# Internationalization

LANGUAGE\_CODE = 'es'   # <-- cambiado de 'en-us' a 'es' para español

TIME\_ZONE = 'America/Santiago'  # <-- cambiado de 'UTC' a Chile

USE\_I18N = True

USE\_TZ = True

# Static files

STATIC\_URL = 'static/'

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

# Configuración de login/logout  <-- agregado

LOGIN\_URL = "login"

LOGIN\_REDIRECT\_URL = "post\_login"

LOGOUT\_REDIRECT\_URL = "login"

**Descripción:**

* load\_dotenv(...) habilita la lectura de variables externas.
* DB\_ENGINE permite seleccionar motor sin modificar código.
* El bloque if/else define DATABASES según el motor (por defecto, SQLite).

## .env (archivo en la raíz, junto a manage.py)

DJANGO\_DEBUG=True

DB\_ENGINE=sqlite

DB\_NAME=db.sqlite3

**Descripción:**  
El archivo .env centraliza parámetros de conexión; de esta forma, la configuración no queda embebida en el código fuente.

# Migraciones (aplicación en BD)

**Comandos utilizados:**

python manage.py makemigrations

python manage.py migrate

**Descripción:**

* makemigrations genera archivos de migración a partir de los modelos declarados.
* migrate aplica dichas migraciones sobre la base de datos (creación y actualización de tablas).

A screenshot of a computer

AI-generated content may be incorrect.

# Modelos

En este proyecto, **Product ≡ Receta**. Se agregan los modelos y relaciones requeridas manteniendo compatibilidad con los modelos existentes.

## recetas/models.py — Organización / Zonas / Dispositivos

from django.db import models

class Category(models.Model):

    name = models.CharField("Nombre", max\_length=120, unique=True)

    class Meta:

        verbose\_name = "Categoría"

        verbose\_name\_plural = "Categorías"

        ordering = ("name",)

    def \_\_str\_\_(self):

        return self.name

# -------------------------

# Receta

# -------------------------

class Receta(TimeStampedSoftDelete):

    nombre = models.CharField(max\_length=150)

    instrucciones = models.TextField(blank=True)

    estandarizada = models.BooleanField(default=False)

    usuario = models.ForeignKey(Usuario, on\_delete=models.SET\_NULL, null=True, blank=True, related\_name="recetas")

    estado = models.CharField(max\_length=10, choices=ESTADO, default="ACTIVO")

    category = models.ForeignKey(

        Category,                 # puedes usar 'Category' en string si prefieres

        on\_delete=models.PROTECT, # evita borrar categorías en uso

        related\_name="recetas",

        null=True, blank=True,    # para no romper tus datos existentes

        verbose\_name="Categoría",

    )

    class Meta:

        verbose\_name = "Receta"

        verbose\_name\_plural = "Recetas"

        ordering = ("nombre",)

    def \_\_str\_\_(self):

        return self.nombre

    def clean(self):

        # Si está estandarizada, debe tener instrucciones

        if self.estandarizada and not (self.instrucciones or "").strip():

            raise ValidationError("Una receta estandarizada debe tener instrucciones.")

**Descripción:**  
Se implementa **AlertRule** (con severidad) y la relación **through** ProductAlertRule para registrar los umbrales min/max por producto, tal como exige la pauta.

## recetas/models.py — Reglas y relación through con umbrales

SEVERITY\_CHOICES = [

    ("low", "Baja"),

    ("med", "Media"),

    ("high", "Alta"),

]

class AlertRule(models.Model):

    name = models.CharField(max\_length=120)

    severity = models.CharField(max\_length=10, choices=SEVERITY\_CHOICES, default="med")

    class Meta:

        verbose\_name = "Regla de alerta"

        verbose\_name\_plural = "Reglas de alerta"

    def \_\_str\_\_(self):

        return f"{self.name} [{self.get\_severity\_display()}]"

class ProductAlertRule(models.Model):

    """

    Relación through: Receta (product) ↔ AlertRule con min/max.

    Importante para cumplir pauta: mostrar min/max en admin.

    """

    # Usamos string 'Receta' para referenciar tu modelo ya creado

    product = models.ForeignKey("Receta", on\_delete=models.CASCADE, related\_name="product\_alert\_rules")

    alert\_rule = models.ForeignKey(AlertRule, on\_delete=models.CASCADE, related\_name="product\_alert\_rules")

    min\_threshold = models.FloatField(null=True, blank=True)

    max\_threshold = models.FloatField(null=True, blank=True)

    class Meta:

        unique\_together = [("product", "alert\_rule")]

        verbose\_name = "Regla por producto"

        verbose\_name\_plural = "Reglas por producto"

    def \_\_str\_\_(self):

        return f"{self.product} → {self.alert\_rule} (min:{self.min\_threshold}, max:{self.max\_threshold})"

**Descripción:**  
Se implementa **AlertRule** (con severidad) y la relación **through** ProductAlertRule para registrar los umbrales min/max por producto, tal como exige la pauta.

## recetas/models.py — Organización / Zonas / Dispositivos

class Organization(models.Model):

    name = models.CharField(max\_length=120, unique=True)

    class Meta:

        verbose\_name = "Organización"

        verbose\_name\_plural = "Organizaciones"

    def \_\_str\_\_(self):

        return self.name

class Zone(models.Model):

    organization = models.ForeignKey(Organization, on\_delete=models.CASCADE, related\_name="zones")

    name = models.CharField(max\_length=120)

    class Meta:

        unique\_together = [("organization", "name")]

        verbose\_name = "Zona"

        verbose\_name\_plural = "Zonas"

    def \_\_str\_\_(self):

        return f"{self.organization} / {self.name}"

class Device(models.Model):

    zone = models.ForeignKey(Zone, on\_delete=models.CASCADE, related\_name="devices")

    name = models.CharField(max\_length=120)

    serial = models.CharField(max\_length=80, unique=True)

    class Meta:

        verbose\_name = "Dispositivo"

        verbose\_name\_plural = "Dispositivos"

    def \_\_str\_\_(self):

        return f"{self.name} ({self.serial})"

**Descripción:**  
Se modela la jerarquía **Organization → Zone → Device** para clasificar activos por cliente/área/equipo.

## recetas/models.py — Series: Measurement

class Measurement(models.Model):

    device = models.ForeignKey(Device, on\_delete=models.CASCADE, related\_name="measurements")

    product = models.ForeignKey("Receta", on\_delete=models.PROTECT, related\_name="measurements")

    alert\_rule = models.ForeignKey(AlertRule, on\_delete=models.PROTECT, related\_name="measurements")

    value = models.FloatField()

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

    class Meta:

        ordering = ["-created\_at"]

        verbose\_name = "Medición"

        verbose\_name\_plural = "Mediciones"

    def \_\_str\_\_(self):

        return f"{self.device} {self.alert\_rule}: {self.value} @ {self.created\_at:%Y-%m-%d %H:%M}"

**Descripción:**  
Measurement persiste series de mediciones por dispositivo/producto/regla, con columna temporal created\_at requerida para date\_hierarchy en el Admin

# Admin básico (registro + columnas)

A screenshot of a computer

AI-generated content may be incorrect.

## Category y Receta

@admin.register(Category)

class CategoryAdmin(admin.ModelAdmin):

    list\_display = ("id", "name")

    search\_fields = ("name",)

    ordering = ("name",)

@admin.register(Receta)

class RecetaAdmin(admin.ModelAdmin):

    # mostramos categoría para cumplir pauta de catálogo

    list\_display = ("id", "nombre", "category", "usuario", "estandarizada", "estado", "created\_at")

    list\_select\_related = ("usuario", "category")

    search\_fields = ("nombre", "usuario\_\_nombre")

    list\_filter = ("category", "estandarizada", "estado", "usuario")

    ordering = ("-created\_at",)

    date\_hierarchy = "created\_at"

## AlertRule y ProductAlertRule (con min/max):

@admin.register(AlertRule)

class AlertRuleAdmin(admin.ModelAdmin):

    list\_display = ("id", "name", "severity")

    list\_filter = ("severity",)

    search\_fields = ("name",)

    ordering = ("name",)

@admin.register(ProductAlertRule)

class ProductAlertRuleAdmin(admin.ModelAdmin):

    list\_display = ("id", "product", "alert\_rule", "min\_threshold", "max\_threshold")

    list\_select\_related = ("product", "alert\_rule")

    list\_filter = ("alert\_rule\_\_severity",)

    search\_fields = (

        "product\_\_nombre",     # si tu Receta usa 'name', cambia a "product\_\_name"

        "alert\_rule\_\_name",

    )

    ordering = ("product\_\_nombre",)      # o ("product\_\_name",) según tu campo

## Organization / Zone / Device:

@admin.register(Organization)

class OrganizationAdmin(admin.ModelAdmin):

    list\_display = ("id", "name")

    search\_fields = ("name",)

    ordering = ("name",)

@admin.register(Zone)

class ZoneAdmin(admin.ModelAdmin):

    list\_display = ("id", "name", "organization")

    list\_select\_related = ("organization",)

    list\_filter = ("organization",)

    search\_fields = ("name", "organization\_\_name")

    ordering = ("organization\_\_name", "name")

@admin.register(Device)

class DeviceAdmin(admin.ModelAdmin):

    list\_display = ("id", "name", "serial", "zone", "organization")

    list\_select\_related = ("zone", "zone\_\_organization")

    list\_filter = ("zone\_\_organization", "zone")

    search\_fields = ("name", "serial", "zone\_\_name", "zone\_\_organization\_\_name")

    ordering = ("zone\_\_organization\_\_name", "zone\_\_name", "name")

    def organization(self, obj):

        return obj.zone.organization

    organization.short\_description = "Organization"

## Measurement (series con date\_hierarchy):

@admin.register(Measurement)

class MeasurementAdmin(admin.ModelAdmin):

    list\_display = ("id", "created\_at", "device", "product", "alert\_rule", "value")

    list\_select\_related = ("device", "product", "alert\_rule")

    date\_hierarchy = "created\_at"  # requisito de la pauta

    list\_filter = ("alert\_rule\_\_severity", "device\_\_zone\_\_organization")

    search\_fields = (

        "device\_\_serial",

        "product\_\_nombre",     # si tu Receta usa 'name', cambia a "product\_\_name"

        "alert\_rule\_\_name",

    )

    ordering = ("-created\_at",)

# Semillas

Se implementó un **management command** idempotente que carga el catálogo, crea entidades por organización y vincula productos con reglas mediante umbrales; además, registra mediciones de ejemplo.

# recetas/management/commands/seed\_catalog\_es.py

from django.core.management.base import BaseCommand

from django.db import transaction

from recetas.models import (

    Category, AlertRule, ProductAlertRule,

    Organization, Zone, Device, Measurement,

    Receta,

)

class Command(BaseCommand):

    help = "Carga catálogo inicial (categorías, reglas, organización) y vincula Receta ↔ AlertRule con umbrales."

    @transaction.atomic

    def handle(self, \*args, \*\*kwargs):

        self.stdout.write("\n== Cargando Catálogo ==\n")

        # --- Categorías

        cat\_bebida, \_ = Category.objects.get\_or\_create(name="Bebida")

        cat\_snack,  \_ = Category.objects.get\_or\_create(name="Snack")

        self.stdout.write(self.style.SUCCESS(f"Categorías OK: {cat\_bebida}, {cat\_snack}"))

        # --- AlertRules

        a\_temp, \_ = AlertRule.objects.get\_or\_create(name="Temperatura", defaults={"severity": "high"})

        a\_ph,   \_ = AlertRule.objects.get\_or\_create(name="pH",           defaults={"severity": "med"})

        self.stdout.write(self.style.SUCCESS(f"AlertRules OK: {a\_temp}, {a\_ph}"))

        # --- Organización / Zonas / Dispositivos

        org, \_ = Organization.objects.get\_or\_create(name="Planta Central")

        z1, \_ = Zone.objects.get\_or\_create(organization=org, name="Cocina")

        z2, \_ = Zone.objects.get\_or\_create(organization=org, name="Envasado")

        d1, \_ = Device.objects.get\_or\_create(zone=z1, serial="EQ-001", defaults={"name": "Horno 1"})

        d2, \_ = Device.objects.get\_or\_create(zone=z2, serial="EQ-002", defaults={"name": "Llenadora 2"})

        self.stdout.write(self.style.SUCCESS(f"Org/Zone/Device OK: {org}, {z1}, {z2}, {d1}, {d2}"))

        # --- Busca recetas por nombre (ajusta si usas otros)

        nombres\_recetas = ["Jugo Naranja", "Jugo Manzana", "Barra Cereal"]

        recetas = list(Receta.objects.filter(nombre\_\_in=nombres\_recetas))

        # Seteamos categoría por conveniencia si no tienen

        for r in recetas:

            if "Jugo" in r.nombre and getattr(r, "category\_id", None) is None:

                r.category = cat\_bebida

                r.save(update\_fields=["category"])

            elif "Barra" in r.nombre and getattr(r, "category\_id", None) is None:

                r.category = cat\_snack

                r.save(update\_fields=["category"])

        if not recetas:

            self.stdout.write(self.style.WARNING(

                "No se encontraron Recetas con los nombres esperados. "

                "Crea algunas en /admin/ o cambia la lista 'nombres\_recetas' en este comando."

            ))

        else:

            # --- Vincula Receta ↔ AlertRule con min/max (through)

            for r in recetas:

                if "Jugo" in r.nombre:

                    ProductAlertRule.objects.update\_or\_create(

                        product=r, alert\_rule=a\_temp,

                        defaults={"min\_threshold": 2, "max\_threshold": 5}

                    )

                    ProductAlertRule.objects.update\_or\_create(

                        product=r, alert\_rule=a\_ph,

                        defaults={"min\_threshold": 3.4, "max\_threshold": 4.2}

                    )

                elif "Barra" in r.nombre:

                    ProductAlertRule.objects.update\_or\_create(

                        product=r, alert\_rule=a\_temp,

                        defaults={"min\_threshold": 15, "max\_threshold": 22}

                    )

            self.stdout.write(self.style.SUCCESS("Vínculos Receta ↔ AlertRule OK (con min/max)."))

            # --- Measurements de ejemplo

            if d1 and recetas:

                m1, \_ = Measurement.objects.get\_or\_create(

                    device=d1, product=recetas[0], alert\_rule=a\_temp, value=3.2

                )

                m2, \_ = Measurement.objects.get\_or\_create(

                    device=d1, product=recetas[0], alert\_rule=a\_ph, value=3.8

                )

                self.stdout.write(self.style.SUCCESS(f"Measurements OK: {m1.id}, {m2.id}"))

        self.stdout.write(self.style.SUCCESS("\nCatálogo inicial cargado.\n"))

**Comando de ejecución:**

python manage.py seed\_catalog\_es

**Descripción:**  
El comando crea categorías y reglas, estructura organización/zonas/dispositivos, vincula recetas con reglas asignando umbrales mínimos y máximos, y registra mediciones de ejemplo. Puede ejecutarse múltiples veces sin duplicar datos.

A computer screen with white text and green text

AI-generated content may be incorrect.

# Git — Rama de avance

**Rama utilizada:**

u2-c2-admin-basico

**Comandos ejecutados:**

git checkout -b u2-c2-admin-basico

git add .

git commit -m "U2: BD (.env) + seeds + admin básico"

git push origin u2-c2-admin-basico

**Descripción:**  
Se crea una rama específica para el avance, dejando trazabilidad clara de los cambios requeridos por la pauta.

# README del repositorio

**Base de datos:**

SQLite.

**Ejecución:**

pip install -r requirements.txt

python manage.py migrate

python manage.py runserver

**Semillas (catálogo):**

python manage.py seed\_catalog\_es

**Administrador:**

python manage.py createsuperuser

# Conclusion

El avance desarrollado deja el proyecto navegable y administrable desde Django Admin, cumpliendo con los hitos definidos en la pauta. Se demostró la conexión a base de datos parametrizada vía .env, la correcta aplicación de migraciones, y la carga del catálogo inicial mediante un management command idempotente. Asimismo, se configuró un admin básico con columnas, búsquedas, filtros, orden por defecto, optimización con list\_select\_related y date\_hierarchy para series temporales, lo que facilita la inspección y gestión de datos. Finalmente, el trabajo quedó versionado en la rama dedicada u2-c2-admin-basico, aportando trazabilidad y buenas prácticas de control de versiones.

El resultado es una base sólida y coherente con el dominio de Estandarización de Recetas, donde el “producto” se mapea a Receta y se integran correctamente Category, AlertRule, ProductAlertRule (con umbrales), Organization, Zone, Device y Measurement. Con esta estructura, el equipo dispone de un punto estable para evolucionar la solución hacia flujos operativos y vistas específicas de usuario.

Próximos pasos sugeridos

-Habilitar formularios y vistas de negocio fuera del Admin (CRUDs y paneles por rol).

-Implementar reglas de permisos/roles y segmentación por organización/zona.

-Automatizar la carga de semillas en despliegues y preparar fixtures de prueba.

-Incorporar validaciones de dominio (p. ej., consistencia de umbrales) y tests automatizados.

-Considerar el paso a MySQL en ambientes superiores, manteniendo la configuración por .env.