

---

## Estandarización de recetas

---

NOMBRE: Cristobal Estay Pozo  
CARRERA: ingeniería informática  
ASIGNATURA: Programación Back End  
PROFESOR: Javier Ahumada Cortes

## Índice

<b>Estandarización de recetas</b> .....	<b>1</b>
1    Introducción .....	3
2    Conexión a BD y configuración .....	4
2.1    receta/settings.py (fragmento con .env + DATABASES).....	4
2.2    .env (archivo en la raíz, junto a manage.py) .....	6
3    Migraciones (aplicación en BD) .....	7
4    Modelos .....	8
4.1    recetas/models.py — Organización / Zonas / Dispositivos .....	8
4.2    recetas/models.py — Reglas y relación through con umbrales.....	9
4.3    recetas/models.py — Organización / Zonas / Dispositivos .....	10
4.4    recetas/models.py — Series: Measurement .....	11
5    Admin básico (registro + columnas).....	12
5.1    Category y Receta .....	12
5.2    AlertRule y ProductAlertRule (con min/max): .....	13
5.3    Organization / Zone / Device: .....	13
5.4    Measurement (series con date_hierarchy):.....	14
6    Semillas .....	14
7    Git — Rama de avance .....	17
8    README del repositorio .....	17
9    Conclusion .....	18

## 1 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

## 2 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.

### 2.1 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).

## 2.2 .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.

### 3 Migraciones (aplicación en BD)

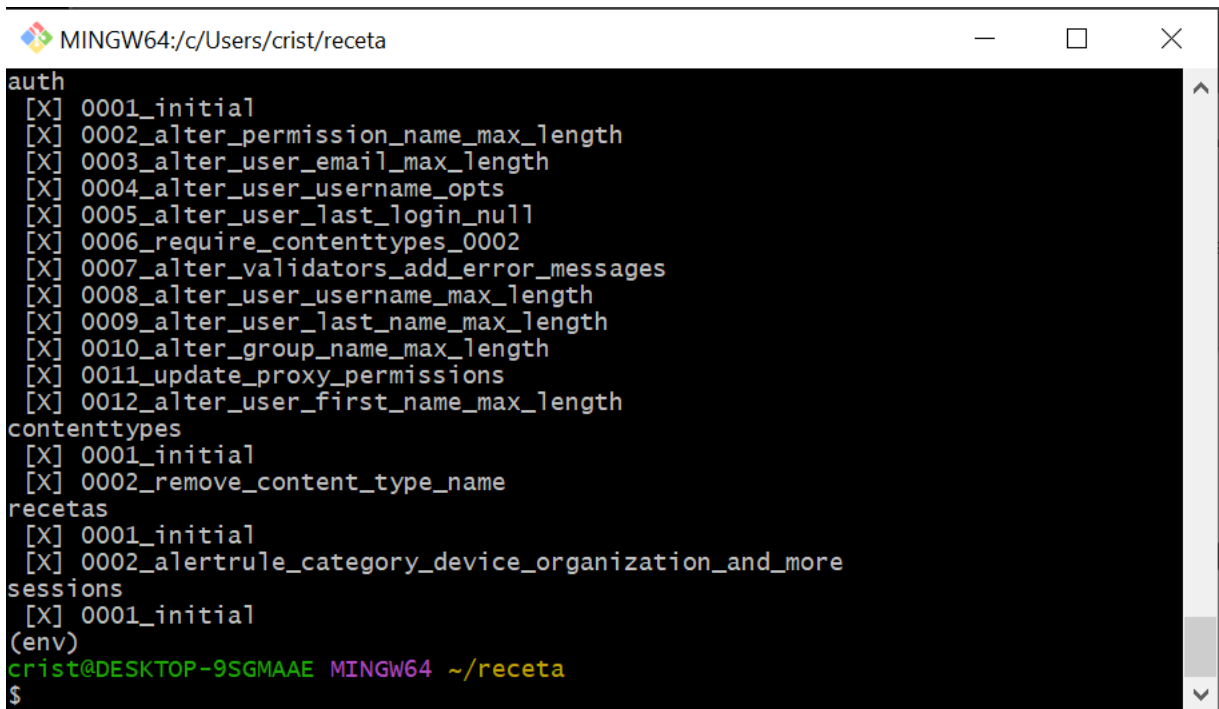
#### Comandos utilizados:

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).



```
MINGW64:/c/Users/crist/receta
auth
[X] 0001_initial
[X] 0002_alter_permission_name_max_length
[X] 0003_alter_user_email_max_length
[X] 0004_alter_user_username_opts
[X] 0005_alter_user_last_login_null
[X] 0006_require_contenttypes_0002
[X] 0007_alter_validators_add_error_messages
[X] 0008_alter_user_username_max_length
[X] 0009_alter_user_last_name_max_length
[X] 0010_alter_group_name_max_length
[X] 0011_update_proxy_permissions
[X] 0012_alter_user_first_name_max_length
contenttypes
[X] 0001_initial
[X] 0002_remove_content_type_name
recetas
[X] 0001_initial
[X] 0002_alertrule_category_device_organization_and_more
sessions
[X] 0001_initial
(env) crist@DESKTOP-9SGMAAE MINGW64 ~/receta
$
```

## 4 Modelos

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

### 4.1 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,
        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.

## 4.2 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.

## 4.3 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.

## 4.4 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

## 5 Admin básico (registro + columnas)

AUTENTICACIÓN Y AUTORIZACIÓN	
Grupos	+ Añadir  Modificar
Usuarios	+ Añadir  Modificar
RECETAS	
Categorías	+ Añadir  Modificar
Dispositivos	+ Añadir  Modificar
Equipos	+ Añadir  Modificar
Ingredientes	+ Añadir  Modificar
Ingredientes de Receta	+ Añadir  Modificar
Mediciones	+ Añadir  Modificar
Organizaciones	+ Añadir  Modificar
Producciones	+ Añadir  Modificar
Prueba calidades	+ Añadir  Modificar
Recetas	+ Añadir  Modificar
Reglas de alerta	+ Añadir  Modificar
Reglas por producto	+ Añadir  Modificar
Usuarios	+ Añadir  Modificar

Acciones recientes

Mis acciones

- + barra cereal  
Receta
- + jugo  
Receta
- + Jugo Naranja  
Receta

### 5.1 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"
```

## 5.2 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
```

## 5.3 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"
```

## 5.4 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",)
```

## 6 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}"))

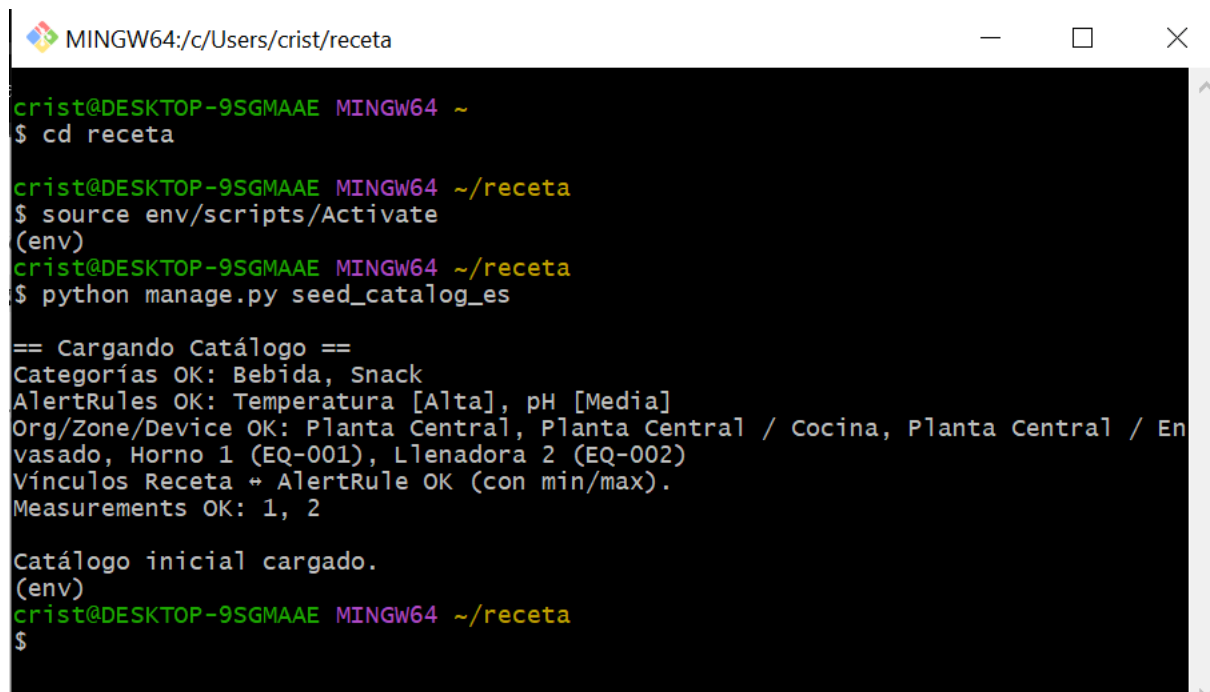
self.stdout.write(self.style.SUCCESS("\nCatálogo inicial cargado.\n"))
```

### 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.



```
MINGW64:/c/Users/crist/receta

crist@DESKTOP-9SGMAAE MINGW64 ~
$ cd receta

crist@DESKTOP-9SGMAAE MINGW64 ~/receta
$ source env/scripts/Activate
(env)
crist@DESKTOP-9SGMAAE MINGW64 ~/receta
$ python manage.py seed_catalog_es

== Cargando Catálogo ==
Categorías OK: Bebida, Snack
AlertRules OK: Temperatura [Alta], pH [Media]
Org/Zone/Device OK: Planta Central, Planta Central / Cocina, Planta Central / En
vasado, Horno 1 (EQ-001), Llenadora 2 (EQ-002)
Vínculos Receta ↔ AlertRule OK (con min/max).
Measurements OK: 1, 2

Catálogo inicial cargado.
(env)
crist@DESKTOP-9SGMAAE MINGW64 ~/receta
$
```



## 7 Git — Rama de avance

### Rama utilizada:

Repositorio: <https://github.com/stay0710/estandarizacion-recetas>

Rama de avance: 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.

## 8 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
```

## 9 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.

