Guía Paso a Paso: Crear un Sistema de Importación de Productos con Django, Bootstrap 5, Grappelli y CRUD Completo

Esta guía está diseñada para una **sesión práctica de 4 horas** donde los estudiantes aprenderán a construir una aplicación web completa en Django con funcionalidades CRUD para **Productos** y **Embarques**. Se cubrirán **modelos**, **vistas basadas en clases (CBV)**, **formularios**, **plantillas**, **admin**, **consultas optimizadas**, **relaciones entre modelos** y **integración de Bootstrap**.

□ Distribución del Tiempo (4 horas ~ 240 minutos)

- **Introducción y Configuración (30 min)**
- **Creación de Modelos y Migraciones (45 min)**
- **Configuración del Admin (30 min)**
- **Creación de Vistas, Formularios y URLs (60 min)**
- **Creación de Plantillas Base y Listas (60 min)**
- **Cierre y Pruebas (15 min)**

□□ 1. Introducción y Configuración (30 min)

a. Presentación del MVP (5 min)

- **Objetivo:** Crear una aplicación web que gestione productos, categorías y embarques de importación con operaciones CRUD completas.
- **Herramientas:** Django, Python, Bootstrap 5, Grappelli, Pillow.
- **Resultado esperado:** Una página web con menús para navegar entre Productos y Embarques, vistas para listar, crear, editar y eliminar cada uno.

b. Preparar el Entorno (10 min)

Abre tu terminal o consola.

1. **Crear un entorno virtual** (opcional pero recomendado):

```
python -m venv venv
```

2. **Activar el entorno virtual:**
   - En Windows:

```
venv\Scripts\activate
```

- En macOS/Linux:

```
source venv/bin/activate
```

3. **Instalar Django y dependencias:**

```
pip install Django django-grappelli pillow
```

c. Crear el Proyecto Django (15 min)

1. **Crear el proyecto principal en el directorio actual (el punto al final es importante):**

```
django-admin startproject sistema_importacion .
```

2. **Crear la aplicación `productos`:**

```
python manage.py startapp productos
```

□□ 2. Configurar `settings.py` y `urls.py` (5 min)

a. Editar `sistema_importacion/settings.py`

Abre el archivo `sistema_importacion/settings.py` con tu editor de texto.

1. **Agregar `'grappelli'` y `'productos'` a `INSTALLED_APPS`. `grappelli` debe ir ANTES de `'django.contrib.admin'`:**

```
INSTALLED_APPS = [
    'grappelli', # <-- Asegúrate de que esté aquí
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'productos', # <-- Agrega tu app aquí
]
```

2. **Configurar `MEDIA` para manejar archivos subidos:**

```
import os # Asegúrate de tener esta línea arriba del archivo

# ... resto de settings ...
```

```
# Media files (Images, etc.)
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

b. Editar `sistema_importacion/urls.py`

Abre `sistema_importacion/urls.py`.

1. **Agregar rutas para servir archivos MEDIA en modo DEBUG (importante para desarrollo) y las URLs de nuestras apps:**

```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings # Importar settings
from django.conf.urls.static import static # Importar static

urlpatterns = [
    path('grappelli/', include('grappelli.urls')), # URL de Grappelli
    path('admin/', admin.site.urls),
    path('', include('productos.urls')), # Ruta para nuestra app productos
]

# Sirve archivos media en modo DEBUG
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

2. **Crear el archivo `productos/urls.py` (vacío por ahora):**

```
touch productos/urls.py
```

□□ 3. Crear Modelos y Aplicar Migraciones (45 min)

a. Definir Modelos en `productos/models.py` (20 min)

Abre `productos/models.py`.

Explicación: Los modelos son clases Python que definen la estructura de la base de datos. Cada modelo es una tabla, cada campo es una columna.

```
from django.db import models

class Categoria(models.Model):
    """
    Modelo para representar una categoría de productos.
    Por ejemplo: 'Electrónica', 'Ropa', 'Alimentos'.
    """
    nombre = models.CharField(max_length=100) # Campo de texto corto
    descripcion = models.TextField(blank=True, null=True) # Campo de texto largo, opcional

    def __str__(self):
        """Devuelve el nombre cuando se imprime el objeto."""
```

```
        return self.nombre

class Producto(models.Model):
    """
    Modelo para representar un producto importado.
    """
    nombre = models.CharField(max_length=200) # Nombre del producto
    precio = models.DecimalField(max_digits=10, decimal_places=2) # Precio con 2 decimales
    stock = models.IntegerField(default=0) # Cantidad disponible
    imagen = models.ImageField(upload_to='productos/', blank=True, null=True) # Imagen opcional
    categoria = models.ForeignKey(Categoria, on_delete=models.CASCADE) # Relación con Categoría

    def __str__(self):
        return self.nombre

class Embarque(models.Model):
    """
    Modelo para representar un embarque de importación.
    """
    codigo_rastreo = models.CharField(max_length=100, unique=True) # Código único
    fecha_llegada = models.DateField() # Fecha de llegada
    productos = models.ManyToManyField(Producto, related_name='embarques') # Relación muchos a m

    def __str__(self):
        return f"Embarque {self.codigo_rastreo}"
```

b. Crear y Aplicar Migraciones (15 min)

Las migraciones son archivos que Django usa para modificar la base de datos según tus modelos.

1. **Crear migraciones:**

```
python manage.py makemigrations productos
```

Deberías ver algo como: `Migrations for 'productos': ...`

2. **Aplicar migraciones a la base de datos:**

```
python manage.py migrate
```

Verás mensajes indicando que las tablas se han creado.

c. Explicar Relaciones (10 min)

- **`ForeignKey` (uno a muchos):** Un `Producto` pertenece a una `Categoria`, pero una `Categoria` puede tener muchos `Producto`.
- **`ManyToManyField` (muchos a muchos):** Un `Embarque` puede contener muchos `Producto`, y un `Producto` puede venir en muchos `Embarque`.
- **`related_name`:** Permite acceder desde `Producto` a sus `Embarque` usando `producto.embarques.all()`.

□□ 4. Configurar el Panel de Administración (30 min)

a. Configurar `productos/admin.py` (20 min)

Abre `productos/admin.py`.

Explicación: El admin de Django es una interfaz web para gestionar los datos de tu aplicación. Grappelli mejora su apariencia.

```
# productos/admin.py

from django.contrib import admin
from .models import Categoria, Producto, Embarque

class EmbarqueAdmin(admin.ModelAdmin):
    filter_horizontal = ('productos',)

admin.site.register(Categoria)
admin.site.register(Producto)
admin.site.register(Embarque, EmbarqueAdmin)
```

- `filter_horizontal` mejora la interfaz para seleccionar productos en un embarque.

b. Crear Superusuario y Probar (10 min)

1. **Crear un superusuario:**

```
python manage.py createsuperuser
```

Sigue las instrucciones para crear un nombre de usuario, correo y contraseña.

2. **Correr el servidor:**

```
python manage.py runserver
```

3. **Abrir el navegador y visitar:** `http://127.0.0.1:8000/admin/`
4. **Iniciar sesión con las credenciales del superusuario.**
5. **Probar el admin:** Agrega algunas categorías, productos y embarques para poblar la base de datos.

□□□ 5. Crear Vistas, Formularios y URLs (60 min)

a. Crear Formularios en `productos/forms.py` (20 min)

Los formularios nos permiten crear/editar objetos de manera segura.

```
# productos/forms.py
```

```
from django import forms
from .models import Producto, Embarque

class ProductoForm(forms.ModelForm):
    class Meta:
        model = Producto
        fields = ['nombre', 'precio', 'stock', 'categoria', 'imagen']
        widgets = {
            'nombre': forms.TextInput(attrs={'class': 'form-control'}),
            'precio': forms.NumberInput(attrs={'class': 'form-control'}),
            'stock': forms.NumberInput(attrs={'class': 'form-control'}),
            'categoria': forms.Select(attrs={'class': 'form-select'}),
            'imagen': forms.ClearableFileInput(attrs={'class': 'form-control'}),
        }

class EmbarqueForm(forms.ModelForm):
    class Meta:
        model = Embarque
        fields = ['codigo_rastreo', 'fecha_llegada', 'productos']
        widgets = {
            'codigo_rastreo': forms.TextInput(attrs={'class': 'form-control'}),
            'fecha_llegada': forms.DateInput(attrs={'class': 'form-control', 'type': 'date'}),
            'productos': forms.SelectMultiple(attrs={'class': 'form-select', 'size': '10'}),
        }
```

b. Crear Vistas en `productos/views.py` (25 min)

Usaremos **Vistas Basadas en Clases (CBV)** para simplificar el CRUD.

```
# productos/views.py

from django.shortcuts import render
from django.urls import reverse_lazy
from django.views.generic import ListView, DetailView, CreateView, UpdateView, DeleteView
from .models import Producto, Embarque
from .forms import ProductoForm, EmbarqueForm

# --- Vistas de Productos ---
class ProductoListView(ListView):
    model = Producto
    template_name = 'productos/producto_lista.html'
    context_object_name = 'productos'

class ProductoDetailView(DetailView):
    model = Producto
    template_name = 'productos/producto_detalle.html'

class ProductoCreateView(CreateView):
    model = Producto
    form_class = ProductoForm
    template_name = 'productos/producto_form.html'
    success_url = reverse_lazy('productos:producto_lista')

class ProductoUpdateView(UpdateView):
    model = Producto
    form_class = ProductoForm
    template_name = 'productos/producto_form.html'
    success_url = reverse_lazy('productos:producto_lista')
```

```
class ProductoDeleteView(DeleteView):
    model = Producto
    template_name = 'productos/producto_confirm_delete.html'
    success_url = reverse_lazy('productos:producto_lista')

# --- Vistas de Embarques ---
class EmbarqueListView(ListView):
    model = Embarque
    template_name = 'productos/embarque_lista.html'
    context_object_name = 'embarques'
    # Optimizamos para cargar los productos relacionados
    queryset = Embarque.objects.all().prefetch_related('productos')

class EmbarqueDetailView(DetailView):
    model = Embarque
    template_name = 'productos/embarque_detalle.html'

class EmbarqueCreateView(CreateView):
    model = Embarque
    form_class = EmbarqueForm
    template_name = 'productos/embarque_form.html'
    success_url = reverse_lazy('productos:embarque_lista')

class EmbarqueUpdateView(UpdateView):
    model = Embarque
    form_class = EmbarqueForm
    template_name = 'productos/embarque_form.html'
    success_url = reverse_lazy('productos:embarque_lista')

class EmbarqueDeleteView(DeleteView):
    model = Embarque
    template_name = 'productos/embarque_confirm_delete.html'
    success_url = reverse_lazy('productos:embarque_lista')

# Vista antigua (opcional, ya no usada si se usan CBV)
def lista_productos(request):
    productos = Producto.objects.all().prefetch_related('embarques')
    context = {
        'productos': productos
    }
    return render(request, 'productos/lista.html', context)
```

c. Actualizar `productos/urls.py` (15 min)

Abre `productos/urls.py` y define todas las rutas para productos y embarques.

```
# productos/urls.py

from django.urls import path
from . import views

app_name = 'productos'

urlpatterns = [
    # URLs de Productos
    path('', views.ProductoListView.as_view(), name='producto_lista'), # Redirige a la lista de
    path('productos/', views.ProductoListView.as_view(), name='producto_lista'),
    path('producto/nuevo/', views.ProductoCreateView.as_view(), name='producto_crear'),
    path('producto/<int:pk>/', views.ProductoDetailView.as_view(), name='producto_detalle'),
    path('producto/<int:pk>/editar/', views.ProductoUpdateView.as_view(), name='producto_editar'
```

```
    path('producto/<int:pk>/eliminar/', views.ProductoDeleteView.as_view(), name='producto_elimi

    # URLs de Embarques
    path('embarques/', views.EmbarqueListView.as_view(), name='embarque_lista'),
    path('embarque/nuevo/', views.EmbarqueCreateView.as_view(), name='embarque_crear'),
    path('embarque/<int:pk>/', views.EmbarqueDetailView.as_view(), name='embarque_detalle'),
    path('embarque/<int:pk>/editar/', views.EmbarqueUpdateView.as_view(), name='embarque_editar'
    path('embarque/<int:pk>/eliminar/', views.EmbarqueDeleteView.as_view(), name='embarque_elimi
]
```

□□ 6. Crear Plantillas Base y Listas (60 min)

a. Crear Directorios y Archivos de Plantillas (5 min)

Dentro de la carpeta `productos`, crea la siguiente estructura:

```
productos/
  └── templates/
      └── productos/
          ├── base.html
          ├── producto_lista.html
          ├── producto_form.html
          ├── producto_detalle.html
          ├── producto_confirm_delete.html
          ├── embarque_lista.html
          ├── embarque_form.html
          ├── embarque_detalle.html
          └── embarque_confirm_delete.html
```

Puedes hacerlo con comandos:

```
mkdir -p productos/templates/productos
touch productos/templates/productos/base.html
touch productos/templates/productos/producto_lista.html
touch productos/templates/productos/producto_form.html
touch productos/templates/productos/producto_detalle.html
touch productos/templates/productos/producto_confirm_delete.html
touch productos/templates/productos/embarque_lista.html
touch productos/templates/productos/embarque_form.html
touch productos/templates/productos/embarque_detalle.html
touch productos/templates/productos/embarque_confirm_delete.html
```

b. Crear `productos/templates/productos/base.html` (25 min)

Esta es la plantilla base con el menú para **Productos** y **Embarques**.

```
<!doctype html>
<html lang="es">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Sistema de Importación</title>
```

```
    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css  " rel="
    <style>
      body {
        font-size: .875rem;
      }
      .navbar {
        background-color: #343a40;
      }
    </style>
</head>
<body>

<header class="navbar navbar-dark sticky-top bg-dark flex-md-nowrap p-0 shadow">
    <a class="navbar-brand col-md-3 col-lg-2 me-0 px-3" href="{% url 'productos:producto_lista'

    <button class="navbar-toggler position-absolute d-md-none collapsed" type="button" data-bs-t
        <span class="navbar-toggler-icon"></span>
    </button>

    <div class="navbar-nav w-100">
        <div class="d-flex justify-content-end">
            <a class="nav-link px-3" href="{% url 'productos:producto_lista' %}">Productos</a>
            <a class="nav-link px-3" href="{% url 'productos:embarque_lista' %}">Embarques</a>
            <a class="nav-link px-3" href="/admin/">Admin</a>
        </div>
    </div>
</header>

<div class="container-fluid">
    <div class="row">
        <main class="col-md-12 ms-sm-auto col-lg-12 px-md-4">
            <div class="d-flex justify-content-between flex-wrap flex-md-nowrap align-items-cent
                <h1 class="h2">{% block page_title %}{% endblock %}</h1>
                {% if request.resolver_match.view_name == 'productos:producto_lista' %}
                <div class="btn-toolbar mb-2 mb-md-0">
                    <a href="{% url 'productos:producto_crear' %}" class="btn btn-sm btn-outline
                        Añadir Nuevo Producto
                    </a>
                </div>
                {% endif %}
                {% if request.resolver_match.view_name == 'productos:embarque_lista' %}
                <div class="btn-toolbar mb-2 mb-md-0">
                    <a href="{% url 'productos:embarque_crear' %}" class="btn btn-sm btn-outline
                        Añadir Nuevo Embarque
                    </a>
                </div>
                {% endif %}
            </div>
            {% block content %}
            {% endblock %}
        </main>
    </div>
</div>

<!-- Bootstrap Bundle with Popper -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js  " int
</body>
</html>
```

Explicación:

- `{% block title %}`, `{% block page_title %}`, `{% block content %}` son **bloques** que pueden ser sobrescritos por otras plantillas que extiendan este archivo.
- `{% url 'productos:...' %}` genera la URL para cada vista usando el namespace `productos`.
- El menú ahora incluye enlaces a `producto_lista` y `embarque_lista`.
- El botón "Añadir Nuevo" cambia según la vista actual.

c. Crear `productos/templates/productos/producto_lista.html` (10 min)

```
{% extends 'productos/base.html' %}

{% block page_title %}Productos{% endblock %}
{% block content %}
<div class="row">
  {% for producto in productos %}
    <div class="col-md-4 col-lg-3 mb-4">
      <div class="card h-100">
        {% if producto.imagen %}
          <img src="{{ producto.imagen.url }}" class="card-img-top" alt="{{ producto.nombre }}">
        {% else %}
          <div class="d-flex align-items-center justify-content-center bg-light" style="height:
            <span class="text-muted">Sin imagen</span>
          </div>
        {% endif %}
        <div class="card-body">
          <h5 class="card-title">{{ producto.nombre }}</h5>
          <h6 class="card-subtitle mb-2 text-muted">{{ producto.categoria.nombre }}</h6>
          <p class="card-text">Precio: ${{ producto.precio }}</p>
          <p class="card-text">Stock: {{ producto.stock }}</p>
          <div class="mt-2">
            <small class="text-muted">Embarques:</small><br>
            {% for embarque in producto.embarques.all %}
              <span class="badge bg-secondary">{{ embarque.codigo_rastreo }}</span>
            {% empty %}
              <span class="badge bg-warning text-dark">No asociado</span>
            {% endfor %}
          </div>
          <div class="mt-2">
            <a href="{% url 'productos:producto_detalle' producto.pk %}" class="btn btn-sm btn-o
            <a href="{% url 'productos:producto_editar' producto.pk %}" class="btn btn-sm btn-ou
            <a href="{% url 'productos:producto_eliminar' producto.pk %}" class="btn btn-sm btn-
          </div>
        </div>
      </div>
    </div>
  {% empty %}
    <p class="text-center">No hay productos registrados aún.</p>
  {% endfor %}
</div>
{% endblock %}
```

d. Crear `productos/templates/productos/embarque_lista.html` (10 min)

```
{% extends 'productos/base.html' %}

{% block page_title %}Embarques{% endblock %}
{% block content %}
<div class="table-responsive">
    <table class="table table-striped table-sm">
```

```
        <thead>
            <tr>
                <th>Código de Rastreo</th>
                <th>Fecha de Llegada</th>
                <th>Productos</th>
                <th>Acciones</th>
            </tr>
        </thead>
        <tbody>
            {% for embarque in embarques %}
            <tr>
                <td>{{ embarque.codigo_rastreo }}</td>
                <td>{{ embarque.fecha_llegada }}</td>
                <td>
                    {% for producto in embarque.productos.all %}
                        <span class="badge bg-primary">{{ producto.nombre }}</span>
                    {% endfor %}
                </td>
                <td>
                    <a href="{% url 'productos:embarque_detalle' embarque.pk %}" class="btn btn-
                    <a href="{% url 'productos:embarque_editar' embarque.pk %}" class="btn btn-s
                    <a href="{% url 'productos:embarque_eliminar' embarque.pk %}" class="btn btn
                </td>
            </tr>
            {% empty %}
            <tr>
                <td colspan="4" class="text-center">No hay embarques registrados aún.</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</div>
{% endblock %}
```

e. Crear `productos/templates/productos/producto_form.html` (5 min)

```
{% extends 'productos/base.html' %}

{% block page_title %}{% if object %}Editar Producto{% else %}Nuevo Producto{% endif %}{% endblo
{% block content %}
    <h2>{% if object %}Editar Producto{% else %}Nuevo Producto{% endif %}</h2>
    <form method="post" enctype="multipart/form-data">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="btn btn-primary">{% if object %}Actualizar{% else %}Crear{%
        <a href="{% url 'productos:producto_lista' %}" class="btn btn-secondary">Cancelar</a>
    </form>
{% endblock %}
```

f. Crear `productos/templates/productos/embarque_form.html` (5 min)

```
{% extends 'productos/base.html' %}

{% block page_title %}{% if object %}Editar Embarque{% else %}Nuevo Embarque{% endif %}{% endblo
{% block content %}
    <h2>{% if object %}Editar Embarque{% else %}Nuevo Embarque{% endif %}</h2>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
```

```
        <button type="submit" class="btn btn-primary">{% if object %}Actualizar{% else %}Crear{%
        <a href="{% url 'productos:embarque_lista' %}" class="btn btn-secondary">Cancelar</a>
    </form>
{% endblock %}
```

□ 7. Cierre y Pruebas (15 min)

a. Correr el Servidor (5 min)

Asegúrate de que el servidor sigue corriendo:

```
python manage.py runserver
```

b. Probar la Aplicación (10 min)

1. Visita `http://127.0.0.1:8000/`. Debería redirigirte a la lista de productos.
2. Observa el menú superior: ahora tiene enlaces para **Productos** y **Embarques**.
3. Prueba:
    - Navegar entre las páginas de Productos y Embarques.
    - Crear nuevos productos y embarques.
    - Editar y eliminar objetos.
    - Ver los detalles de cada uno.
4. Asegúrate de que la interfaz sea intuitiva y funcional.

c. Discusión y Retroalimentación (5 min)

- ¿Qué partes fueron más claras?
- ¿Qué fue más difícil de entender?
- ¿Cómo se relacionan los modelos con las vistas y los templates?

□□ Instrucciones Finales

Para resumir, estos son los comandos principales que usaste:

- `django-admin startproject nombre_proyecto .`
- `python manage.py startapp nombre_app`
- `pip install ...`
- `python manage.py makemigrations nombre_app`
- `python manage.py migrate`
- `python manage.py createsuperuser`
- `python manage.py runserver`