



PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

1. IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Técnico en programación de Software
- Código del Programa de Formación: 233104-V2
- Nombre del Proyecto Formativo (si aplica): DESARROLLO DE APLICACIONES DE SOFTWARE PARA EL SECTOR EMPRESARIAL
- Fase del Proyecto (si aplica): Ejecución
- Actividad de Proyecto Formativo (si aplica): Desarrollar el software de acuerdo con el diseño y a la programación establecida
- Competencia: 220501096 DESARROLLAR LA SOLUCIÓN DE SOFTWARE DE ACUERDO CON EL DISEÑO Y LA PROGRAMACIÓN ESTABLECIDA
- Resultados de Aprendizaje: Rap 3. CODIFICAR EL SOFTWARE EMPLEANDO EL LENGUAJE DE PROGRAMACIÓN SEGÚN REQUERIMIENTO Y DISEÑO
- Duración de la Guía de Aprendizaje (horas): 72 horas (24 sincrónicas + 48 asincrónicas)

2. PRESENTACIÓN

Hasta ahora tu aplicación maneja productos y categorías: una categoría puede tener muchos productos, pero cada producto pertenece a una sola categoría. Esa es una relación uno a muchos, y la resolviste con ForeignKey en la guía de Migraciones y Relaciones.

Pero piensa en cómo funciona un carrito de compras en Mercado Libre, Amazon o Rappi: un pedido puede contener varios productos distintos (audífonos, un cargador y una mochila, por ejemplo), y al mismo tiempo, un mismo producto (esos audífonos) puede aparecer en muchos pedidos diferentes, hechos por distintos clientes. Aquí ya no es “uno a muchos”, sino muchos a muchos: muchos pedidos se relacionan con muchos productos.



En esta guía vas a aprender a modelar y construir esa relación: crearás un modelo Pedido, y usando un modelo intermedio podrás registrar qué productos (y en qué cantidad) tiene cada pedido. Al final, tu aplicación podrá crear pedidos, agregarles productos, calcular el total a pagar y mostrar el detalle completo de cada pedido.

En esta guía vas a:

Entender qué es una relación muchos a muchos y por qué se necesita un modelo intermedio cuando hay información extra (como la cantidad).

Crear los modelos Pedido y PedidoDetalle, y sus migraciones.

Construir una vista para crear pedidos y agregarles productos.

Mostrar el detalle de un pedido: productos, cantidades, subtotales y total.

Listar todos los pedidos y permitir eliminar un producto de un pedido.

Al finalizar esta guía, tu aplicación de productos habrá dado un salto enorme: ya no solo mostrará un catálogo, sino que permitirá simular pedidos completos, como una mini tienda en línea. ¡Vamos a construirlo!

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

- **Descripción de la(s) Actividad(es)**

3.1 Actividades de reflexión inicial:

Descripción de la actividad:

Piensa en un pedido real de comida a domicilio: un solo pedido puede tener hamburguesa, papas y gaseosa. ¿Cómo crees que la app guarda esa información? ¿Un producto puede estar en varios pedidos distintos al mismo tiempo? Dibuja o describe cómo imaginas la relación entre Pedido y Producto en una base de datos.

Ambiente requerido: Ambiente de trabajo individual o en parejas con acceso a computador con Python 3.x instalado, entorno virtual activo, proyecto Django configurado y conexión a internet para consultar documentación.



Estrategias o técnicas didácticas activas: Lluvia de ideas; discusión guiada; preguntas orientadoras; trabajo en parejas.

Materiales de formación Computador con Python 3, Django y las librerías requeridas instaladas; editor de código (VS Code o similar); navegador web; terminal o consola del sistema.

Material de apoyo: Documentación oficial de Django (<https://docs.djangoproject.com/>); Documentación de Python (<https://docs.python.org/3/>); guías y videos del instructor; repositorio del proyecto en GitHub.

Duración de la actividad: 1 hora.

3.2 Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje:

Descripción de la actividad:

Repaso: relación uno a muchos (ForeignKey)

En la guía de Migraciones y Relaciones, cada Producto tenía un campo categoria de tipo ForeignKey. Esto significa: una categoría puede tener muchos productos, pero cada producto apunta a UNA sola categoría.

¿Por qué un pedido y sus productos son “muchos a muchos”?

Un pedido puede tener varios productos.

Un producto puede estar en varios pedidos.

Django ofrece un campo llamado ManyToManyField justamente para este tipo de relaciones. Sin embargo, en nuestro caso necesitamos guardar información adicional sobre la relación: la cantidad de cada producto en el pedido. Por ejemplo, un pedido puede tener 2 audífonos y 1 mochila: el “2” y el “1” son datos de la RELACIÓN, no del producto ni del pedido por sí solos.

Cuando una relación muchos a muchos necesita información adicional, en lugar de usar ManyToManyField directamente, se crea un modelo intermedio (también llamado tabla intermedia o through model): un modelo que tiene un ForeignKey hacia cada uno de los modelos relacionados, más los campos extra que se necesiten.

Los modelos Pedido y PedidoDetalle



```
from django.db import models

class Pedido(models.Model):
    cliente = models.CharField(max_length=100)
    fecha = models.DateField(auto_now_add=True)

    def __str__(self):
        return f'Pedido #{self.id} - {self.cliente}'

class PedidoDetalle(models.Model):
    pedido = models.ForeignKey(Pedido, on_delete=models.CASCADE)
    producto = models.ForeignKey(Producto, on_delete=models.PROTECT)
    cantidad = models.PositiveIntegerField()

    def subtotal(self):
        return self.producto.precio * self.cantidad

    def __str__(self):
        return f'{self.cantidad} x {self.producto.nombre}'
```

Pedido: representa el pedido en sí, con el cliente y la fecha (auto_now_add=True guarda la fecha automáticamente al crearlo, igual que viste en la guía de Models).

PedidoDetalle: es el modelo intermedio. Cada fila representa “este producto, en esta cantidad, dentro de este pedido”.

on_delete=models.CASCADE en pedido: si se elimina un Pedido, se eliminan automáticamente sus detalles.

on_delete=models.PROTECT en producto: no se puede eliminar un Producto si todavía aparece en algún PedidoDetalle (recuerda las opciones de on_delete de la guía de Migraciones).

subtotal(): un método que calcula precio × cantidad para esa línea del pedido.

Pasos para crear los modelos

En models.py, agrega las clases Pedido y PedidoDetalle como en el ejemplo.

Ejecuta python manage.py makemigrations para generar el archivo de migración.

Ejecuta python manage.py migrate para crear las tablas en la base de datos.

Registra ambos modelos en admin.py para poder revisarlos desde el panel de administración.



Formularios para Pedido y PedidoDetalle

Igual que en la guía de Formularios, usamos ModelForm para construir los formularios automáticamente:

```
from django import forms
from .models import Pedido, PedidoDetalle

class PedidoForm(forms.ModelForm):
    class Meta:
        model = Pedido
        fields = ['cliente']

class PedidoDetalleForm(forms.ModelForm):
    class Meta:
        model = PedidoDetalle
        fields = ['producto', 'cantidad']
```

PedidoForm: solo pide el campo cliente (la fecha se llena sola, gracias a auto_now_add).

PedidoDetalleForm: pide producto (Django mostrará automáticamente un <select> con todos los productos, gracias al ForeignKey) y cantidad.

Vista para crear un pedido

```
from django.shortcuts import render, redirect
from .forms import PedidoForm

def crear_pedido(request):
    if request.method == 'POST':
        form = PedidoForm(request.POST)
        if form.is_valid():
            pedido = form.save()
            return redirect('detalle_pedido', pedido_id=pedido.id)
        else:
            form = PedidoForm()

    return render(request, 'productos/crear_pedido.html', {'form': form})
```

Observa que form.save() devuelve el objeto recién creado (el pedido). Usamos pedido.id para redirigir directamente a la página de detalle de ESE pedido, donde el usuario podrá empezar a agregarle productos.



Vista para agregar un producto al pedido

```
from django.shortcuts import render, redirect, get_object_or_404
from .models import Pedido
from .forms import PedidoDetalleForm

def agregar_producto_pedido(request, pedido_id):
    pedido = get_object_or_404(Pedido, id=pedido_id)

    if request.method == 'POST':
        form = PedidoDetalleForm(request.POST)
        if form.is_valid():
            detalle = form.save(commit=False)
            detalle.pedido = pedido
            detalle.save()
            return redirect('detalle_pedido', pedido_id=pedido.id)
        else:
            form = PedidoDetalleForm()

    return render(request, 'productos/agregar_producto.html', {'form': form, 'pedido':
pedido})
```

`form.save(commit=False)`: crea el objeto `PedidoDetalle` EN MEMORIA, pero todavía no lo guarda en la base de datos. Esto nos da la oportunidad de completar datos antes de guardar.

`detalle.pedido = pedido`: asignamos a qué pedido pertenece este detalle (el formulario no pide este dato porque ya lo sabemos: viene en la URL).

`detalle.save()`: ahora sí se guarda en la base de datos, con el pedido ya asignado.

Pasos para crear estas vistas

Crea el archivo `forms.py` (o agrega a tu archivo existente) las clases `PedidoForm` y `PedidoDetalleForm`.

Crea la vista `crear_pedido` y su template `crear_pedido.html` (similar a `crear.html` de la guía de Formularios).

Crea la vista `agregar_producto_pedido` y su template `agregar_producto.html`.

En `urls.py`, agrega las rutas `'pedidos/crear/'` y `'pedidos/<int:pedido_id>/agregar/'`, con sus respectivos `name`.



related_name y el “conjunto relacionado”

Cuando un modelo (PedidoDetalle) tiene un ForeignKey hacia otro (Pedido), Django nos permite, desde un objeto Pedido, acceder a TODOS los PedidoDetalle relacionados, usando NombreModelo_set en minúsculas:

```
pedido = Pedido.objects.get(id=1)
detalles = pedido.pedidodetalle_set.all() # todos los PedidoDetalle de este pedido
```

Vista de detalle de pedido

```
from django.shortcuts import render, get_object_or_404
from .models import Pedido

def detalle_pedido(request, pedido_id):
    pedido = get_object_or_404(Pedido, id=pedido_id)
    detalles = pedido.pedidodetalle_set.all()

    total = 0
    for detalle in detalles:
        total += detalle.subtotal()

    contexto = {
        'pedido': pedido,
        'detalles': detalles,
        'total': total,
    }
    return render(request, 'productos/detalle_pedido.html', contexto)
```

Template detalle_pedido.html

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

{% block titulo %}Detalle del pedido{% endblock %}

{% block content %}
<h2>{{ pedido }}</h2>
<p>Fecha: {{ pedido.fecha }}</p>

<table border="1">
  <tr>
    <th>Producto</th>
    <th>Cantidad</th>
    <th>Precio unitario</th>
    <th>Subtotal</th>
  </tr>
```



```
{% for detalle in detalles %}
<tr>
  <td>{{ detalle.producto.nombre }}</td>
  <td>{{ detalle.cantidad }}</td>
  <td>${{ detalle.producto.precio|floatformat:2 }}</td>
  <td>${{ detalle.subtotal|floatformat:2 }}</td>
</tr>
{% endfor %}
</table>

<h3>Total: ${{ total|floatformat:2 }}</h3>

<a href="{% url 'agregar_producto_pedido' pedido.id %}">Agregar producto</a>
{% endblock %}
```

Fíjate que `detalle.producto.nombre` y `detalle.producto.precio` acceden a campos del Producto relacionado, igual que `producto.categoria.nombre` en la guía de Templates: simplemente “se viaja” por el ForeignKey usando puntos.

Pasos para crear la vista de detalle

Crea la vista `detalle_pedido` siguiendo el ejemplo, calculando el total con un ciclo for en Python.

En `urls.py`, agrega la ruta `'pedidos/<int:pedido_id>/'` con `name='detalle_pedido'`.

Crea el template `detalle_pedido.html` mostrando la tabla de productos del pedido y el total.

Ambiente requerido: Ambiente de trabajo individual o en parejas con acceso a computador con Python 3.x instalado, entorno virtual activo, proyecto Django configurado y conexión a internet para consultar documentación.

Estrategias o técnicas didácticas activas: Aprendizaje basado en proyectos (ABP); lectura analítica de documentación técnica; resolución de problemas por indagación; trabajo colaborativo entre pares; retroalimentación formativa.

Materiales de formación Computador con Python 3, Django y las librerías requeridas instaladas; editor de código (VS Code o similar); navegador web; terminal o consola del sistema.

Material de apoyo: Documentación oficial de Django (<https://docs.djangoproject.com/>); Documentación de Python (<https://docs.python.org/3/>); guías y videos del instructor; repositorio del proyecto en GitHub.

Duración de la actividad: 4 horas.



3.3 Actividades de apropiación:

Descripción de la actividad:

Crea los modelos Pedido y PedidoDetalle en tu proyecto, genera y aplica las migraciones, y regístralos en el panel de administración. Verifica en /admin/ que ambas tablas existan y que puedas crear un pedido manualmente desde ahí.

Implementa crear_pedido y agregar_producto_pedido junto con sus templates y rutas.

Crea un pedido nuevo desde el navegador y agrégale al menos tres productos distintos, con diferentes cantidades.

Implementa detalle_pedido y su template.

Verifica que el total mostrado sea la suma correcta de todos los subtotales del pedido.

Agrega un producto más al pedido desde el enlace “Agregar producto” y confirma que el total se actualice.

Ambiente requerido: Ambiente de trabajo individual o en parejas con acceso a computador con Python 3.x instalado, entorno virtual activo, proyecto Django configurado y conexión a internet para consultar documentación.

Estrategias o técnicas didácticas activas: Aprendizaje basado en proyectos (ABP); lectura analítica de documentación técnica; resolución de problemas por indagación; trabajo colaborativo entre pares; retroalimentación formativa.

Materiales de formación Computador con Python 3, Django y las librerías requeridas instaladas; editor de código (VS Code o similar); navegador web; terminal o consola del sistema.

Material de apoyo: Documentación oficial de Django (<https://docs.djangoproject.com/>); Documentación de Python (<https://docs.python.org/3/>); guías y videos del instructor; repositorio del proyecto en GitHub.

Evidencias de aprendizaje: Código fuente funcional del proyecto Django con las funcionalidades implementadas. Capturas de pantalla que evidencien el correcto funcionamiento en el navegador.



Instrumentos de evaluación: Lista de chequeo de funcionalidades implementadas. Revisión de código por pares.

Duración de la actividad: 6 horas.

3.4 Actividades de Transferencia el Conocimiento:

Descripción de la actividad:

Implementa la vista `lista_pedidos`, su ruta `'pedidos/'` (`name='lista_pedidos'`) y su template `lista_pedidos.html`, mostrando el cliente, la fecha y un enlace al detalle de cada pedido.

Implementa la vista `eliminar_detalle_pedido` y su ruta `'pedidos/detalle/<int:detalle_id>/eliminar/'` (`name='eliminar_detalle_pedido'`).

En `detalle_pedido.html`, agrega un enlace “Quitar” junto a cada producto del pedido.

Protege las vistas `crear_pedido`, `agregar_producto_pedido` y `eliminar_detalle_pedido` con `@login_required` (recuerda la guía de Autenticación).

Prueba el flujo completo: crea un pedido, agrégale varios productos, quita uno, y verifica que el total se recalcule correctamente.

PARTE TEÓRICA:

Responde las siguientes preguntas:

Explica con tus palabras por qué la relación entre Pedido y Producto es “muchos a muchos” y no “uno a muchos”.

¿Por qué se creó el modelo `PedidoDetalle` en lugar de usar directamente un `ManyToManyField` entre Pedido y Producto?

¿Qué hace `pedido.pedidodetalle_set.all()` y de dónde viene ese nombre?

¿Qué diferencia hay entre `form.save()` y `form.save(commit=False)`? ¿En qué situación usarías cada uno?

En el modelo `PedidoDetalle`, ¿qué pasaría si el campo `producto` tuviera `on_delete=models.CASCADE` en lugar de `PROTECT`? Explica con un ejemplo.

Ambiente requerido: Ambiente de trabajo individual o en parejas con acceso a computador con Python 3.x instalado, entorno virtual activo, proyecto Django configurado y conexión a internet para consultar documentación.



Estrategias o técnicas didácticas activas: Aprendizaje basado en proyectos (ABP); lectura analítica de documentación técnica; resolución de problemas por indagación; trabajo colaborativo entre pares; retroalimentación formativa.

Materiales de formación Computador con Python 3, Django y las librerías requeridas instaladas; editor de código (VS Code o similar); navegador web; terminal o consola del sistema.

Material de apoyo: Documentación oficial de Django (<https://docs.djangoproject.com/>); Documentación de Python (<https://docs.python.org/3/>); guías y videos del instructor; repositorio del proyecto en GitHub.

Evidencias de aprendizaje: Actividad práctica con capturas de pantalla. Respuestas escritas a las preguntas teóricas (entregadas en documento o plataforma del instructor).

Instrumentos de evaluación: Rúbrica de evaluación de desempeño. Cuestionario de preguntas abiertas.

Duración de la actividad: 3 horas.

4. PLANTEAMIENTO DE EVIDENCIAS DE APRENDIZAJE PARA LA EVALUACIÓN EN EL PROCESO FORMATIVO.

Fase del proyecto formativo	Actividad del proyecto formativo	Actividad de Aprendizaje	Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación

Evidencia de Conocimiento: Respuestas a las preguntas teóricas de la actividad evaluativa (sección 3.4.2).

Evidencia de Desempeño: Implementación funcional de todas las actividades prácticas indicadas en las secciones 3.3 y 3.4.1, demostrada mediante capturas de pantalla y revisión del código fuente.

Evidencia de Producto: Proyecto Django actualizado con las funcionalidades de la guía correctamente implementadas, almacenado en repositorio Git.

5. GLOSARIO DE TÉRMINOS

Django: Framework web de alto nivel para Python que fomenta el desarrollo rápido y seguro.



ForeignKey: Campo de un modelo que crea una relación (muchos a uno) con otro modelo.

Relación muchos a muchos (M2M): Relación en la que muchos registros de un modelo se asocian con muchos registros de otro modelo.

ManyToManyField: Campo de Django para crear relaciones muchos a muchos directamente, sin necesidad de un modelo intermedio.

Modelo intermedio (through model): Modelo adicional que conecta dos modelos en una relación muchos a muchos y permite guardar información extra sobre la relación.

auto_now_add: Opción de un campo de fecha que guarda automáticamente la fecha y hora actuales al crear el registro.

related_name / conjunto relacionado (_set): Forma de acceder, desde un objeto, a todos los registros relacionados mediante un ForeignKey hacia ese objeto.

ModelForm: Tipo especial de Django Form que genera automáticamente los campos de un formulario a partir de un modelo.

form.save(commit=False): Crea el objeto a partir de un formulario válido sin guardarlo todavía en la base de datos, para poder completarlo antes de guardar.

get_object_or_404: Función que busca un objeto en la base de datos por su identificador o muestra automáticamente un error 404 si no existe.

@login_required: Decorador que obliga a que una vista solo pueda usarse si el usuario tiene una sesión iniciada.

6. REFERENTES BIBLIOGRÁFICOS

Django Software Foundation. (2024). Django documentation.
<https://docs.djangoproject.com/>

Mozilla Developer Network. (2024). Django Web Framework (Python).
<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django>

Holovaty, A. & Kaplan-Moss, J. (2009). The Definitive Guide to Django. Apress.

Vincent, W. S. (2022). Django for Beginners. Leanpub.

Python Software Foundation. (2024). The Python Tutorial.
<https://docs.python.org/3/tutorial/>



7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	Jheyson Fabian Villavisan	Instructor	CDM	25/06/2026

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					