



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

Imagina que tu catálogo de productos crece y crece: pasas de tener 5 productos a tener 500. Si un usuario quiere encontrar “audífonos”, no tiene sentido que tenga que revisar página por página hasta encontrarlos. Necesita un buscador, como el que usas en Mercado Libre, Amazon o YouTube.

Además, cuando hay muchos resultados, mostrarlos todos de una sola vez haría que la página cargue muy lento y se vea saturada. Por eso, casi todas las apps dividen los resultados en páginas: piensa en cuando buscas algo en Google y ves “Página 1, 2, 3...” al final, o cuando haces scroll en Instagram y la app va cargando publicaciones poco a poco.



En esta guía vas a aprender a hacer que tu aplicación de productos sea mucho más usable: el usuario podrá buscar productos por nombre, filtrarlos por categoría, y navegar los resultados por páginas, sin que la aplicación se vuelva lenta ni desordenada.

En esta guía vas a:

Repasar qué es un QuerySet y cómo usar `filter()` para consultar datos específicos.

Crear un buscador de productos por nombre usando el operador `icontains`.

Agregar un filtro por categoría usando un menú desplegable.

Implementar paginación con la clase `Paginator` de Django, para mostrar los resultados en bloques.

Combinar búsqueda, filtro y paginación en una sola vista, conservando los criterios del usuario al cambiar de página.

Al finalizar esta guía, tu catálogo de productos funcionará como el de una tienda en línea real: con buscador, filtros y navegación por páginas. ¡Comencemos!

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

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

3.1 Actividades de reflexión inicial:

Descripción de la actividad:

Entra a cualquier tienda en línea (Mercado Libre, Amazon, Falabella) y usa el buscador. ¿Cómo filtraste los resultados? ¿Pudiste filtrar por categoría, precio o marca? ¿Crees que los resultados estaban todos en una sola página o divididos? Reflexiona sobre cómo funciona esa búsqueda por detrás y escribe tus ideas.

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: QuerySets y filter()

Recuerda que `Producto.objects.all()` devuelve un `QuerySet` con todos los productos. Para obtener solo algunos, Django ofrece el método `filter()`, que recibe condiciones en forma de `campo=valor`.

Por ejemplo, para buscar productos cuyo nombre contenga cierto texto (sin importar mayúsculas o minúsculas), se usa el lookup `icontains`:

```
# Busca productos cuyo nombre contenga la palabra 'audífonos'
Producto.objects.filter(nombre__icontains='audífonos')
```

`icontains`: "contains" (contiene) + "i" de insensitive (no distingue mayúsculas/minúsculas).

`nombre__icontains='texto'`: el doble guion bajo (`__`) conecta el nombre del campo con el tipo de búsqueda (lookup).

Leyendo lo que el usuario escribió: `request.GET`

Cuando un formulario usa `method="GET"` (como un buscador), los datos enviados aparecen en la URL como parámetros, por ejemplo `/productos/lista/?q=audífonos`. Django nos permite leer esos parámetros con `request.GET.get('q')`:

```
def lista_productos(request):
    query = request.GET.get('q')
    productos = Producto.objects.all()

    if query:
        productos = productos.filter(nombre__icontains=query)
```



```
contexto = {'productos': productos, 'query': query}
return render(request, 'productos/lista.html', contexto)
```

request.GET.get('q'): obtiene el valor del parámetro q en la URL. Si no existe, devuelve None.

if query: solo se filtra si el usuario realmente escribió algo en el buscador.

El formulario de búsqueda en el template

Un formulario de búsqueda usa method="GET" (no necesita {% csrf_token %}), porque GET no modifica datos):

```
<form method="GET">
  <input type="text" name="q" value="{{ query|default:'' }}" placeholder="Buscar
producto...">
  <button type="submit">Buscar</button>
</form>
```

name="q": debe coincidir con el nombre del parámetro que lees con request.GET.get('q').

value="{{ query|default:'' }}": mantiene escrito en el buscador el último término buscado.

Pasos para crear el buscador

En views.py, modifica lista_productos para leer request.GET.get('q').

Si query existe, filtra el QuerySet con .filter(nombre__icontains=query).

Agrega query al contexto que se envía al template.

En lista.html, agrega el formulario de búsqueda con method="GET".

Filtrar usando relaciones (ForeignKey)

Recuerda que cada Producto tiene una categoría (ForeignKey hacia Categoría). Para filtrar productos por categoría, podemos usar el id de la categoría:

```
Producto.objects.filter(categoria_id=2)
```

Un menú desplegable (select) para elegir categoría

Para que el usuario elija una categoría, necesitamos enviarle al template la lista de categorías disponibles, y mostrarla en un <select>:



```
from .models import Producto, Categoria

def lista_productos(request):
    query = request.GET.get('q')
    categoria_id = request.GET.get('categoria')
    productos = Producto.objects.all()
    categorias = Categoria.objects.all()

    if query:
        productos = productos.filter(nombre__icontains=query)

    if categoria_id:
        productos = productos.filter(categoria_id=categoria_id)

    contexto = {
        'productos': productos,
        'categorias': categorias,
        'query': query,
        'categoria_id': categoria_id,
    }
    return render(request, 'productos/lista.html', contexto)
```

En el template, recorreremos categorías para generar las opciones del <select>, y marcamos como seleccionada la categoría que el usuario eligió:

```
<form method="GET">
  <input type="text" name="q" value="{{ query|default:'' }}" placeholder="Buscar
  producto...">

  <select name="categoria">
    <option value="">Todas las categorías</option>
    {% for cat in categorias %}
    <option value="{{ cat.id }}"
      {% if categoria_id == cat.id|stringformat:'s' %}selected{% endif %}>
      {{ cat.nombre }}
    </option>
    {% endfor %}
  </select>

  <button type="submit">Filtrar</button>
</form>

{% if categoria_id == cat.id|stringformat:'s' %}: compara el id de la categoría (convertido a
texto) con el valor recibido en la URL, para dejar la opción seleccionada.
```

Pasos para agregar el filtro por categoría



En views.py, lee request.GET.get('categoria').

Si categoria_id existe, filtra el QuerySet con .filter(categoria_id=categoria_id).

Envía categorías (todas las categorías) y categoria_id al contexto.

En lista.html, agrega el <select> con las categorías, dentro del mismo formulario de búsqueda.

La clase Paginator

Django incluye la clase Paginator, que toma un QuerySet (o cualquier lista) y un número de elementos por página, y se encarga de dividirlo en “páginas”:

```
from django.core.paginator import Paginator

def lista_productos(request):
    productos = Producto.objects.all()
    # ... aplicar filtros de búsqueda y categoría ...

    paginator = Paginator(productos, 5) # 5 productos por página
    numero_pagina = request.GET.get('page')
    page_obj = paginator.get_page(numero_pagina)

    contexto = {'page_obj': page_obj}
    return render(request, 'productos/lista.html', contexto)
```

Paginator(productos, 5): crea un paginador que reparte el QuerySet en grupos de 5 elementos.

request.GET.get('page'): lee el número de página desde la URL, por ejemplo ?page=2.

paginator.get_page(numero_pagina): devuelve un objeto Page con los productos de esa página. Si el número no es válido, devuelve la primera o la última página automáticamente, sin generar errores.

Usando page_obj en el template

page_obj se comporta como una lista (puedes recorrerla con {% for %}), pero además tiene información útil sobre la paginación:

page_obj.number: el número de la página actual.

page_obj.has_previous() / page_obj.has_next(): indican si existe una página anterior o siguiente.



page_obj.previous_page_number / page_obj.next_page_number: el número de esas páginas.

page_obj.paginator.num_pages: el total de páginas.

```
{% for producto in page_obj %}
  <!-- fila de la tabla con producto.nombre, producto.precio, etc. -->
{% endfor %}

<div class="paginacion">
  {% if page_obj.has_previous %}
    <a href="?page={{ page_obj.previous_page_number }}">Anterior</a>
  {% endif %}

  <span>Página {{ page_obj.number }} de {{ page_obj.paginator.num_pages }}</span>

  {% if page_obj.has_next %}
    <a href="?page={{ page_obj.next_page_number }}">Siguiente</a>
  {% endif %}
</div>
```

Pasos para agregar paginación

En views.py, importa Paginator desde django.core.paginator.

Crea el paginador con el QuerySet ya filtrado y un tamaño de página (por ejemplo, 5).

Obtén page_obj usando request.GET.get('page').

En lista.html, cambia el {% for %} para recorrer page_obj en lugar de productos.

Agrega los enlaces “Anterior” y “Siguiente” usando los métodos de page_obj.

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:

Agrega un buscador en tu catálogo de productos que permita filtrar por nombre. Prueba buscando un producto que exista (debe aparecer) y uno que no exista (debe mostrarse el mensaje de “no hay productos”, igual que en la guía de Templates).

Agrega el filtro por categoría a tu catálogo.

Verifica que el buscador por nombre y el filtro por categoría funcionen juntos (por ejemplo, buscar “audífonos” dentro de la categoría “Tecnología”).

Implementa la paginación en tu catálogo, mostrando un máximo de 5 productos por página.

Agrega al menos 8 productos a tu base de datos (puedes usar el formulario “Agregar producto” de la guía de Formularios) para poder ver más de una página.

Verifica que los enlaces “Anterior” y “Siguiente” funcionen correctamente, y que no aparezcan si no hay página anterior o siguiente.

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:

Asegúrate de que la vista `lista_productos` aplique, en este orden: filtro por nombre (si hay `q`), filtro por categoría (si hay categoría) y, al final, la paginación.

Modifica los enlaces “Anterior” y “Siguiete” para que incluyan los parámetros `q` y categoría, además de `page`.

Prueba el flujo completo: busca un término, elige una categoría, y navega entre páginas verificando que el buscador y el filtro NO se pierdan.

Verifica qué ocurre si escribes manualmente en la URL un número de página que no existe (por ejemplo, `?page=999`): la aplicación no debe mostrar un error.

PARTE TEÓRICA:

Responde las siguientes preguntas:

¿Qué hace el `lookup icontains` y en qué se diferencia de usar simplemente `contains`?

¿Por qué el formulario de búsqueda usa `method="GET"` y no `method="POST"`?

¿Qué problema resuelve la clase `Paginator` y qué pasaría si mostraras 500 productos en una sola página sin paginación?

¿Qué información ofrece el objeto `page_obj` además de la lista de elementos de la página actual?

¿Por qué es importante conservar los parámetros `q` y categoría en los enlaces de paginación? ¿Qué pasaría si no se conservaran?



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.

QuerySet: Colección de objetos obtenidos de la base de datos mediante el ORM de Django, sobre la cual se pueden aplicar filtros.

filter(): Método de un QuerySet que devuelve solo los objetos que cumplen una condición.

Lookup (icontains, exact, etc.): Forma especial de escribir una condición de filtro, conectada al nombre del campo con doble guion bajo (__).

request.GET: Diccionario con los parámetros enviados en la URL mediante un formulario con method="GET".

Parámetro de consulta (query parameter): Valor enviado en la URL después de un signo de interrogación, por ejemplo ?q=audifonos.

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

Paginator: Clase de Django que divide un QuerySet o lista en grupos (páginas) de un tamaño definido.

Page object (page_obj): Objeto que representa una página específica generada por un Paginator, con información de navegación.

has_next() / has_previous(): Métodos de un page_obj que indican si existe una página siguiente o anterior.

`{% if %} / {% else %} / {% endif %}`: Etiquetas DTL para mostrar contenido condicional dentro de un template.

`{% for %} / {% endfor %}`: Etiquetas DTL para recorrer una lista o QuerySet dentro de un template.

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)					