



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

En la guía anterior tu aplicación dejó de ser un simple mensaje de texto y se convirtió en una página web real, con catálogo, categorías y vista de detalle. Sin embargo, hasta ahora tu sitio es como una vitrina: el usuario solo puede mirar. Si quieres agregar, modificar o eliminar un producto, todavía tienes que meterte al shell de Django o al panel de administración.

Piensa en cualquier app que uses a diario: cuando editas tu perfil de Instagram, publicas una historia, agregas una canción a una playlist de Spotify o te registras en una nueva cuenta, en todos esos casos estás llenando un formulario que envía información al servidor. Esa información se procesa, se valida y finalmente se guarda en una base de datos.



En esta guía vas a aprender a construir esos formularios con Django, para que tu aplicación de productos deje de ser una vitrina y se convierta en una herramienta completa donde el usuario pueda crear, editar y eliminar productos directamente desde el navegador, sin tocar el shell ni el panel de administración.

En esta guía vas a:

Entender cómo funciona un formulario HTML y la diferencia entre los métodos GET y POST.

Usar Django Forms y, en especial, ModelForm para crear formularios automáticamente a partir de tus modelos.

Crear una vista y un template para agregar nuevos productos (Create).

Crear una vista y un template para editar productos existentes (Update).

Crear una vista para eliminar productos (Delete), completando así el ciclo CRUD desde la web.

Al finalizar esta guía, cualquier persona podrá usar tu aplicación para administrar el catálogo completo de productos: agregar nuevos, corregir precios o existencias, y eliminar los que ya no se necesiten, todo desde el navegador. ¡Empecemos!

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 las últimas veces que llenaste un formulario en línea (registro de cuenta, compra, encuesta). ¿Qué pasó cuando enviaste datos incorrectos? ¿La app te avisó del error? ¿Cómo crees que Django sabe qué datos son válidos y cuáles no? Escribe tu reflexión (mínimo 5 líneas).

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:

Formularios HTML: GET y POST

Un formulario HTML es el elemento que permite que un usuario escriba información y la envíe al servidor, por ejemplo un nombre de usuario, un precio o una descripción. En HTML, un formulario básico se ve así:

```
<form method="POST">  
  <input type="text" name="nombre">  
  <button type="submit">Enviar</button>  
</form>
```

El atributo method indica cómo se envían los datos al servidor:

GET: se usa normalmente para solicitar información (por ejemplo, mostrar una página). Los datos viajan visibles en la URL.

POST: se usa para enviar información que el servidor va a procesar o guardar, como los datos de un nuevo producto. Los datos NO se ven en la URL.

El token CSRF

Django protege tu aplicación de ataques donde un sitio externo intenta enviar formularios a tu servidor sin permiso. Por eso, todo formulario que use method="POST" en un template de Django debe incluir la etiqueta {% csrf_token %} justo después de <form ...>:



```
<form method="POST">
  {% csrf_token %}
  ...
</form>
```

Django Forms y ModelForm

Escribir a mano cada campo de un formulario HTML, y luego validar uno por uno los datos que llegan, sería muy repetitivo (¡y rompería el principio DRY!). Por eso Django ofrece la librería forms, que permite definir formularios como clases de Python.

La forma más rápida de crear un formulario para un modelo es usando ModelForm: Django genera automáticamente un campo de formulario por cada campo del modelo, junto con sus validaciones.

Crea un archivo forms.py dentro de tu aplicación productos:

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

class ProductoForm(forms.ModelForm):
    class Meta:
        model = Producto
        fields = '__all__'
```

La clase interna Meta le indica a Django:

model: a partir de qué modelo se debe construir el formulario (Producto).

fields: qué campos del modelo deben incluirse en el formulario. '__all__' significa "todos los campos".

Pasos para crear tu archivo de formularios

Dentro de la carpeta de tu aplicación (productos), crea un archivo llamado forms.py.

Importa la librería forms de Django y el modelo Producto.

Crea la clase ProductoForm que herede de forms.ModelForm.

Dentro de la clase, define la clase interna Meta indicando model = Producto y fields = '__all__'.

Una vista que procesa formularios

Una misma vista puede comportarse de dos formas distintas, según cómo se haya accedido a ella:



Si el usuario apenas entra a la página (método GET), la vista debe mostrar el formulario vacío.

Si el usuario ya llenó el formulario y lo envió (método POST), la vista debe tomar esos datos, validarlos y, si todo está correcto, guardarlos en la base de datos.

Django nos permite distinguir entre ambos casos revisando `request.method`:

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

def crear_producto(request):
    if request.method == 'POST':
        form = ProductoForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('lista_productos')
    else:
        form = ProductoForm()

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

¿Qué hace cada parte de esta vista?

`form = ProductoForm()`: crea un formulario vacío para mostrarlo la primera vez (GET).

`form = ProductoForm(request.POST)`: crea el formulario con los datos que el usuario envió.

`form.is_valid()`: revisa que los datos cumplan las reglas del modelo (por ejemplo, que precio sea un número).

`form.save()`: guarda un nuevo registro en la base de datos con esos datos.

`redirect('lista_productos')`: redirige al usuario a otra vista (en este caso, el catálogo) usando el name definido en `urls.py`.

El template del formulario

En el template, Django puede dibujar automáticamente todos los campos del formulario con `{{ form.as_p }}` (cada campo dentro de su propia etiqueta `<p>`):

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

{% block titulo %}Nuevo producto{% endblock %}

{% block content %}
```



```
<h2>Agregar nuevo producto</h2>
<form method="POST">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Guardar</button>
</form>
{% endblock %}
```

Pasos para crear la vista y el template

En views.py, importa redirect y ProductoForm.

Crea la función crear_producto siguiendo el ejemplo anterior.

En urls.py, agrega la ruta 'crear/' apuntando a crear_producto, con name='crear_producto'.

Crea el template crear.html que extienda de base.html, con el formulario, {% csrf_token %} y {{ form.as_p }}.

Editar un producto es muy parecido a crear uno, con una diferencia clave: el formulario debe aparecer ya lleno con los datos actuales del producto, y al guardar, debe actualizar ese registro en lugar de crear uno nuevo.

Para esto, ModelForm acepta un parámetro instance: si le pasamos un objeto ya existente, el formulario se llena automáticamente con sus datos, y al guardar, actualiza ese mismo objeto en vez de crear uno nuevo.

```
from django.shortcuts import render, redirect, get_object_or_404
from .models import Producto
from .forms import ProductoForm

def editar_producto(request, producto_id):
    producto = get_object_or_404(Producto, id=producto_id)

    if request.method == 'POST':
        form = ProductoForm(request.POST, instance=producto)
        if form.is_valid():
            form.save()
            return redirect('lista_productos')
    else:
        form = ProductoForm(instance=producto)

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



Observa la diferencia con crear_producto:

`get_object_or_404(Producto, id=producto_id)`: busca el producto que se quiere editar, igual que hiciste en la vista de detalle de la guía anterior.

`ProductoForm(instance=producto)`: en el método GET, llena el formulario con los datos actuales del producto.

`ProductoForm(request.POST, instance=producto)`: en el método POST, actualiza ese mismo producto con los nuevos datos enviados.

La ruta de edición necesita el id del producto, igual que la vista de detalle:

`path('editar/<int:producto_id>/', views.editar_producto, name='editar_producto')`,

El template `editar.html` puede ser casi idéntico a `crear.html`, cambiando únicamente el título y el texto del botón.

Pasos para crear la vista y el template

En `views.py`, crea la función `editar_producto` siguiendo el ejemplo anterior.

En `urls.py`, agrega la ruta `'editar/<int:producto_id>/'` con `name='editar_producto'`.

Crea el template `editar.html`, similar a `crear.html`, pero con el título “Editar producto”.

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 el archivo `forms.py` en tu aplicación productiva y define un `ModelForm` para tu modelo principal (el que usaste en las guías anteriores). Verifica que el formulario incluya todos los campos del modelo (nombre, precio, stock, imagen, categoría, marca, descripción, etc.).

Implementa la vista `crear_producto` en tu proyecto, usando tu `ModelForm`.

Agrega la ruta correspondiente en `urls.py`.

Crea el template `crear.html` con el formulario completo.

En `lista.html`, agrega un enlace o botón “Agregar producto” que lleve a esta nueva vista usando `{% url %}`.

Prueba el formulario: crea al menos dos productos nuevos desde el navegador y verifica que aparezcan en el catálogo.

Implementa la vista `editar_producto` y su ruta en `urls.py`.

Crea el template `editar.html`.

En `lista.html`, agrega un enlace “Editar” junto a cada producto que lleve a esta vista usando `{% url 'editar_producto' producto.id %}`.

Prueba editar al menos un producto y verifica que los cambios se reflejen en el catálogo.

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 `eliminar_producto` siguiendo el ejemplo anterior.

Agrega la ruta `'eliminar/<int:producto_id>/'` en `urls.py`, con `name='eliminar_producto'`.

Crea el template `eliminar.html` con el mensaje de confirmación.

En `lista.html`, agrega un enlace o botón “Eliminar” junto a cada producto que lleve a esta vista usando `{% url %}`.

Verifica el CRUD completo: crea, edita, consulta el detalle y elimina al menos un producto desde el navegador.

PARTE TEÓRICA:

Responde las siguientes preguntas:

¿Cuál es la diferencia entre los métodos GET y POST en un formulario HTML?

¿Para qué sirve la etiqueta `{% csrf_token %}` y qué pasaría si la quitaras de un formulario?

Explica con tus palabras qué hace un `ModelForm` y por qué resulta útil aplicar el principio DRY.

¿Qué diferencia hay entre la vista `crear_producto` y la vista `editar_producto`?

¿Por qué la operación de eliminar pide una confirmación (un formulario POST con botón) en lugar de borrar el producto directamente al hacer clic en un enlace?



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.

Proyecto: Contenedor principal de una aplicación web, con configuraciones globales.

Aplicación (App): Módulo reutilizable que se encarga de una funcionalidad específica dentro de un proyecto.

MVT (Model-View-Template): Patrón de diseño de Django que organiza el código en tres capas lógicas: datos (Model), lógica (View) y presentación (Template).

DRY (Don't Repeat Yourself): Principio de desarrollo que busca reducir la duplicación de código.

ORM y CRUD: El ORM traduce código Python a consultas SQL; CRUD son las operaciones básicas de Crear, Leer, Actualizar y Eliminar datos.

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

Template (Plantilla): Archivo, normalmente .html, que combina HTML con el Django Template Language (DTL) para presentar información al usuario.

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.

Formulario HTML: Elemento de una página web que permite a un usuario ingresar y enviar información al servidor.

Método GET: Método HTTP usado para solicitar información; los datos enviados son visibles en la URL.

Método POST: Método HTTP usado para enviar información que el servidor procesará o guardará; los datos no se muestran en la URL.

request.method: Atributo que indica con qué método HTTP (GET o POST) se hizo la petición a una vista.

CSRF / {% csrf_token %}: Mecanismo de seguridad de Django que protege los formularios contra envíos no autorizados desde otros sitios.



Django Forms: Sistema de Django para definir formularios como clases de Python, con validación automática de datos.

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

form.is_valid(): Método que revisa si los datos enviados en un formulario cumplen las reglas y tipos definidos en el modelo.

form.save(): Método que guarda (crea o actualiza) en la base de datos los datos validados de un formulario.

redirect(): Función que envía al usuario a otra URL o vista, normalmente después de procesar un formulario.

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)					