Proyecto Grupal 4

Adolfo García, Joaquín Vielma, Nicolás Labbé

Identifiquen claramente las diferencias entre los métodos GET y POST.
En el contexto del desarrollo web, los métodos GET y POST son dos de los verbos HTTP más utilizados para enviar y recibir datos entre un cliente (como un navegador web) y un servidor (como una aplicación Django). Ambos métodos son utilizados para transferir datos, pero existen diferencias clave entre ellos. Aquí están las principales diferencias entre los métodos GET y POST:

1. **Parámetros en la URL vs. Cuerpo de la solicitud:**

GET: Los parámetros se envían a través de la URL como parte de la cadena de consulta (query string). Por ejemplo, en una URL como https://example.com/?nombre=Juan&edad=30, los datos "nombre" y "edad" se están enviando mediante el método GET.
POST: Los parámetros se envían en el cuerpo (body) de la solicitud HTTP en lugar de la URL. Esto significa que los datos no son visibles directamente en la URL.

2. **Tamaño de datos:**

GET: Debido a que los datos se incluyen en la URL, la cantidad de información que se puede enviar a través de GET está limitada por la longitud máxima de una URL, que suele ser de varios miles de caracteres en la mayoría de los navegadores y servidores web.
POST: Al enviar datos a través de POST, el tamaño de los datos no está restringido por la longitud de la URL y puede ser mucho mayor. Esto hace que POST sea más adecuado para el envío de grandes cantidades de datos, como formularios extensos.

3. **Seguridad:**

GET: Como los datos se muestran en la URL, cualquier información sensible, como contraseñas, no debe enviarse a través del método GET, ya que quedaría expuesta en el historial del navegador, logs de servidores y otros registros.
POST: Al enviar datos a través de POST, los parámetros no son visibles en la URL, lo que proporciona una capa adicional de seguridad y privacidad para los datos sensibles.

4. **Almacenamiento en caché:**

GET: Las solicitudes GET pueden almacenarse en caché en el navegador o en servidores proxy, lo que puede llevar a problemas de caché y a que los datos enviados no se actualicen correctamente.
POST: Las solicitudes POST no se almacenan en caché, lo que significa que siempre se solicita una respuesta fresca del servidor.

5. **Seguridad contra CSRF (Cross-Site Request Forgery):**

GET: Los formularios que utilizan el método GET son más vulnerables a ataques CSRF, ya que los navegadores pueden seguir enlaces GET automáticamente sin necesidad de una interacción explícita del usuario.
POST: Django proporciona protección incorporada contra CSRF para las solicitudes POST, lo que hace que sea más seguro utilizar este método para enviar datos confidenciales o realizar acciones que modifican el estado del servidor.

Elabore una pequeña guía que sintetice los pasos necesarios para levantar un formulario desde Django con Forms y Models.

Para levantar un formulario desde Django con Forms y Models debes

1. Crea un proyecto de Django.
2. Crea un modelo para tu formulario.
3. Crea un formulario basado en tu modelo.
4. Muestra el formulario en una plantilla HTML.
5. Procesa los datos del formulario en una vista.

Aquí tienes más detalles sobre cada paso:

1. **Crea un proyecto de Django.**

Para crear un proyecto de Django, puedes usar el comando `django-admin startproject` en la terminal. Por ejemplo, para crear un proyecto llamado `TeLoVendo`, puedes usar el siguiente comando:

```
django-admin startproject TeLoVendo
```

2. **Crea un modelo para tu formulario.**

Para crear un modelo para tu formulario, puedes usar el comando `django-admin.py startapp TeLoVendo` en la terminal. Por ejemplo, para crear una aplicación llamada `TeLoVendo`, puedes usar el siguiente comando:

```
django-admin.py startapp TeLoVendo
```

Una vez que hayas creado la aplicación, puedes crear un modelo para tu formulario en el archivo models.py de la aplicación. Por ejemplo, para crear un modelo llamado `Contacto` con dos campos, name y email, puedes usar el siguiente código:

```
from django.db import models

class Contacto(models.Model):
    nombre = models.CharField(max_length=100)
    email = models.EmailField()
```

3. **Crea un formulario basado en tu modelo.**

Para crear un formulario basado en tu modelo, puedes usar la clase `forms.ModelForm`. Por ejemplo, para crear un formulario basado en el modelo `Contacto`, puedes usar el siguiente código:

```
from django.forms import ModelForm

class ContactoForm(ModelForm):
    class Meta:
        model = Contacto
```

4. **Muestra el formulario en una plantilla HTML.**

Para mostrar el formulario en una plantilla HTML, puedes usar la etiqueta `{% form %}`. Por ejemplo, para mostrar el formulario ContactoForm en la plantilla contact.html, puedes usar el siguiente código:

```
{% form form %}
```

5. **Procesa los datos del formulario en una vista.**

Para procesar los datos del formulario en una vista, puedes usar la función `forms.is_valid()` para verificar si el formulario es válido. Si el formulario es válido, puedes usar la función `forms.save()` para guardar los datos del formulario en la base de datos. Por ejemplo, para procesar los datos del formulario `ContactoForm` en la vista `ContactoFormulario`, puedes usar el siguiente código:

```
from django.shortcuts import render
from django.forms import is_valid, save

def ContactoFormulario(request):
     if request.method == "POST":
        formulario_post = ContactoForm (request.POST)
        if formulario_post.is_valid():
            formulario_post.save()
            messages.success(request, 'Proveedor creado
exitosamente.')
            return redirect(ContactoFormulario)
        else:
            messages.error(request, 'Ha ocurrido un error al crear
el proveedor. Por favor, verifica los datos ingresados.')
```