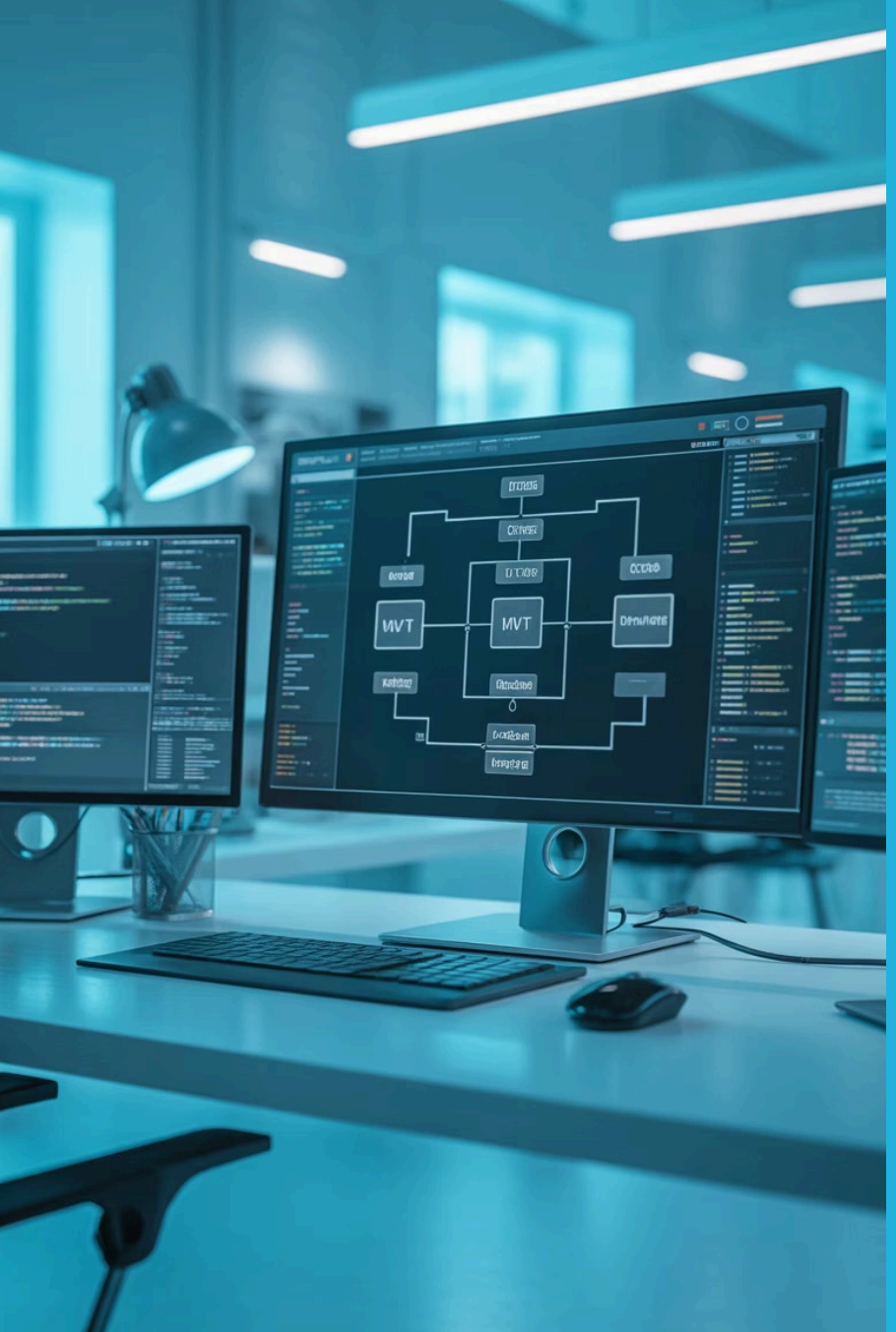


AE6 – Tokens, Enrutamiento y CRUD MVC con Django

Módulo 7 – Desarrollo de Aplicaciones Web con Python y Django

Objetivo general: Implementar una aplicación web MVC/MVT con operaciones CRUD seguras usando Django.

Profesor: Cristian Iglesias

🎯 Objetivos del Aprendizaje

Al finalizar esta clase, dominarás los conceptos fundamentales para construir aplicaciones web seguras y funcionales con Django, aplicando las mejores prácticas de desarrollo.

Seguridad CSRF

Comprender y aplicar el token CSRF en formularios Django para proteger tu aplicación de ataques maliciosos.

Enrutamiento Avanzado

Configurar correctamente el sistema de URLs y paso de parámetros dinámicos entre vistas.

CRUD Completo

Implementar operaciones completas bajo el patrón MVT (Modelo–Vista–Template) de Django.

Buenas Prácticas

Identificar patrones profesionales y evitar los errores más comunes en desarrollo Django.

🧠 ¿Qué es un Token CSRF?

Concepto Clave

Un ataque CSRF (Cross-Site Request Forgery) ocurre cuando un sitio malicioso intenta hacer que un usuario autenticado ejecute acciones sin su consentimiento en otro sitio donde tiene sesión activa.

Django previene estos ataques utilizando un token de seguridad único generado por el servidor. Este token actúa como una "contraseña temporal" que vincula cada formulario con la sesión del usuario.

> Ejemplo práctico: Un usuario autenticado podría ser engañado para hacer clic en un enlace que elimina un producto de su tienda. El token CSRF evita que esta acción no autorizada se ejecute.

⚙️ Cómo Funciona el Token CSRF

El mecanismo de protección CSRF en Django sigue un proceso de validación en cuatro pasos que garantiza la autenticidad de cada solicitud POST.

01

Generación del Token

Django crea un token único y aleatorio para cada sesión de usuario activa.

02

Inserción en Formulario

El token se inserta automáticamente en el HTML mediante la etiqueta {% csrf_token %}.

03

Envío con Datos

Cuando el usuario envía el formulario, el token viaja junto con los datos al servidor.

04

Validación

Django compara el token recibido con el guardado en sesión. Si no coinciden, la solicitud se bloquea con error 403.

Ejemplo Visual en HTML

```
<form method="POST">
 {% csrf_token %}
 {{ form.as_p }}
 <button type="submit">Guardar</button>
</form>
```

🧩 Ejemplo Práctico con Bikeshop

Implementemos la creación segura de bicicletas en nuestro proyecto *bikeshop*, aplicando correctamente la protección CSRF.

Vista en Django (views.py)

```
def crear_bicicleta(request):
    if request.method == 'POST':
        form = BicicletaForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('lista_bicicletas')
    else:
        form = BicicletaForm()

    return render(request,
        'bicicletas/crear_bicicleta.html',
        {'form': form})
```

Template HTML

```
<form method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">
        Crear Bicicleta
    </button>
</form>
```

> 👉 **El token CSRF debe incluirse antes del botón de envío para que el formulario funcione correctamente.**

🚫 Desactivando CSRF (Con Precaución)

En algunos casos específicos, como APIs públicas o pruebas con AJAX, puede ser necesario desactivar temporalmente la protección CSRF. Django ofrece el decorador `@csrf_exempt` para estos escenarios controlados.

Implementación

```
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def mi_vista_sin_csrf(request):
    # Procesar datos sin validación CSRF
    return HttpResponse("Vista sin protección CSRF")
```

⚠️ **ADVERTENCIA CRÍTICA:** Nunca uses este decorador en formularios de usuario final. Solo para casos muy específicos como webhooks o APIs internas controladas.

Casos válidos de uso:

- Endpoints de API REST
- Webhooks de servicios externos
- Pruebas automatizadas específicas

🌐 ¿Qué es el Enrutamiento en Django?

El sistema de enrutamiento es el mecanismo que conecta las URLs que escribe el usuario en el navegador con las vistas específicas que ejecutará Django para procesar la solicitud.

Archivo bicicletas/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    path('',
        views.lista_bicicletas,
        name='lista_bicicletas'),

    path('crear/',
        views.crear_bicicleta,
        name='crear_bicicleta'),
]
```

Funcionamiento

Cada `path()` define una ruta que mapea:

- **URL pattern: El patrón de la dirección**
- **Vista: La función que procesa la solicitud**
- **Name: Un identificador único para referenciar la URL**

Este sistema permite mantener URLs limpias, semánticas y fáciles de mantener.

🧭 Incluyendo URLs en el Proyecto

Para que Django reconozca las rutas de nuestra aplicación, debemos incluirlas en el archivo principal de configuración de URLs del proyecto usando la función `include()`.

1

urls.py Principal

Archivo de configuración central del proyecto

2

include()

Función que importa las URLs de cada app

3

urls.py de App

Rutas específicas de bicicletas

Configuración del Proyecto

```
from django.urls import path, include
from django.contrib import admin

urlpatterns = [
    path('admin/', admin.site.urls),
    path('bicicletas/', include('bicicletas.urls')),
]
```

Resultado: Ahora podemos acceder a las siguientes URLs en nuestro navegador:

`/bicicletas/` → Lista completa de bicicletas

`/bicicletas/crear/` → Formulario de creación

🔢 Paso de Parámetros en URLs

Django permite capturar valores dinámicos desde la URL y pasarlos como argumentos a nuestras vistas. Esto es fundamental para operaciones como ver, editar o eliminar registros específicos.

Definición de Ruta

```
path('<int:pk>/',
    views.detalle_bicicleta,
    name='detalle_bicicleta'),
```

El patrón `<int:pk>` captura un número entero y lo pasa como parámetro `pk` a la vista.

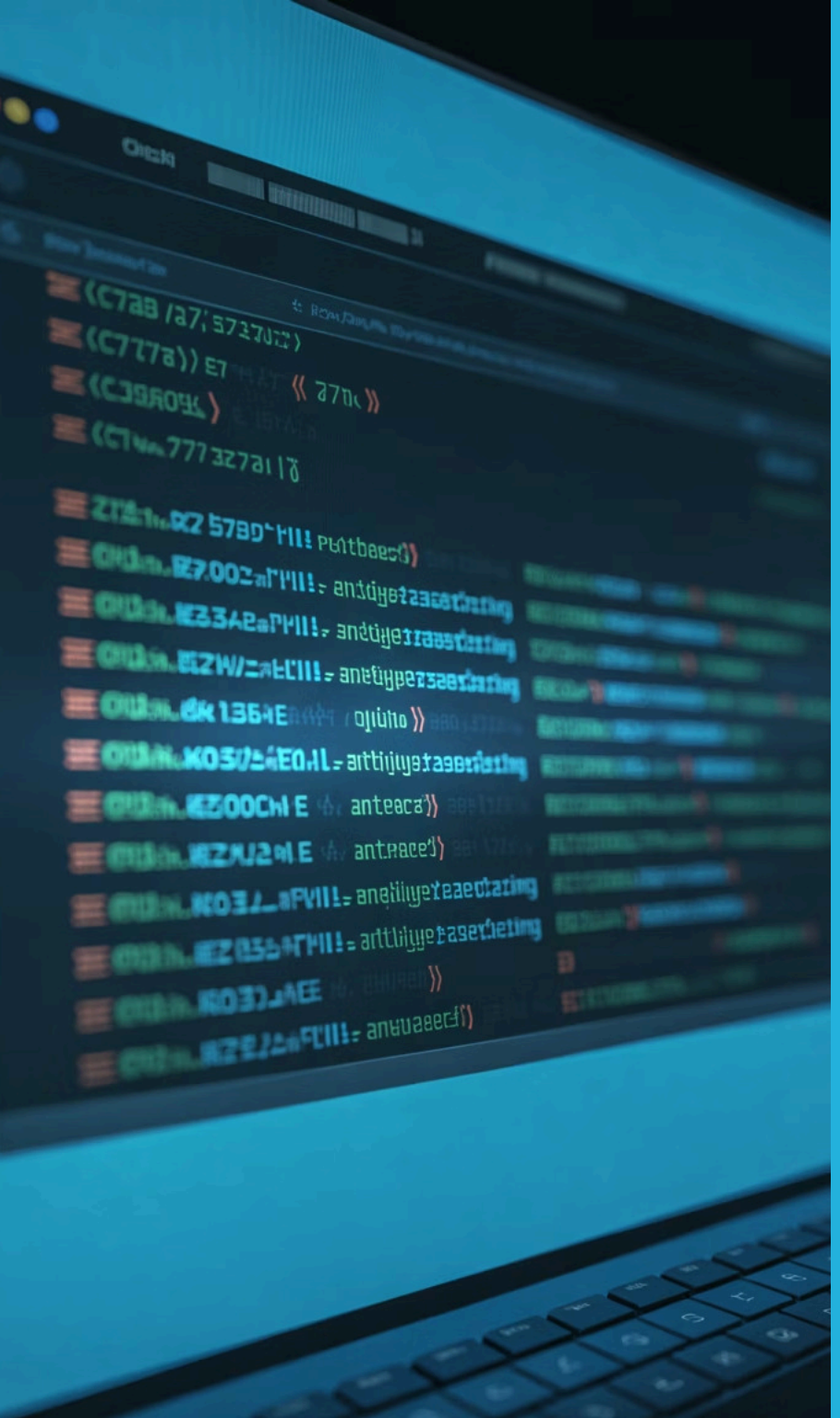
Vista Correspondiente

```
def detalle_bicicleta(request, pk):
    bicicleta = get_object_or_404(
        Bicicleta, pk=pk
    )
    return render(request,
        'bicicletas/detalle_bicicleta.html',
        {'bicicleta': bicicleta})
```

👩‍💻 Actividad Práctica

Abre tu navegador y prueba acceder a `/bicicletas/3/` para ver el detalle de la bicicleta con ID 3.

Observa: Cómo el número en la URL se convierte en el parámetro `pk` dentro de la vista.



🔄 Parámetros Opcionales con re_path

Para casos más complejos donde necesitamos parámetros opcionales o patrones de URL más flexibles, Django ofrece `re_path`, que utiliza expresiones regulares para definir rutas.

Sintaxis con Regex

```
from django.urls import re_path

re_path(
    r'^bicicleta/(?P<pk>\d+)?$',
    views.detalle_bicicleta
)
```

El `?` al final del grupo hace que el parámetro sea opcional.

Comportamiento

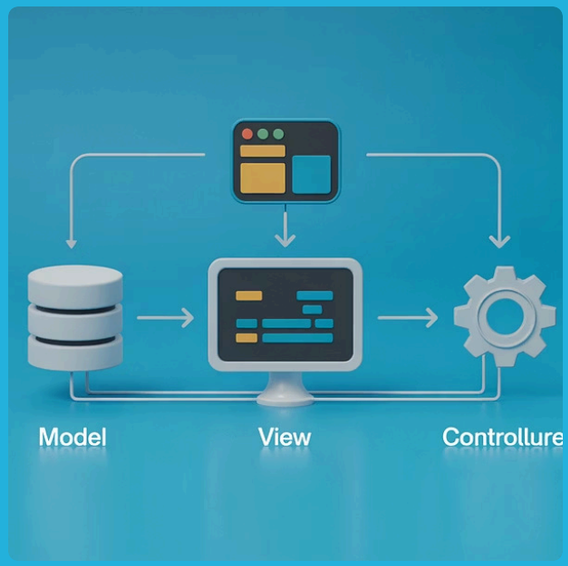
Con parámetro: `/bicicleta/5/` → Muestra la bicicleta con ID 5

Sin parámetro: `/bicicleta/` → Puede mostrar una lista general o mensaje predeterminado

💡 **Caso de uso ideal:** Páginas que combinan funcionalidad de listado general y detalle específico en una misma vista, optimizando la estructura de URLs.

MVC vs MVT: La Base del CRUD

Comprender la diferencia entre el patrón tradicional MVC y el enfoque MVT de Django es fundamental para desarrollar aplicaciones web correctamente estructuradas.



MVC Tradicional

- **Modelo: Gestión de datos**
- **Vista: Interfaz visual**
- **Controlador: Lógica de negocio**

MVT en Django

- **Model: Igual que MVC**
- **View: Lógica (rol del controlador)**
- **Template: Interfaz (rol de la vista)**

Resumen Clave

En Django, lo que tradicionalmente se llama "controlador" se denomina "view" (vista), y lo que antes era la "vista" ahora es el "template". Esta nomenclatura puede confundir al principio, pero es importante dominarla para trabajar efectivamente con Django.

🚲 Modelo Bicicleta: Ejemplo Práctico

El modelo define la estructura de datos de nuestra aplicación. En Django, cada modelo se traduce automáticamente en una tabla de base de datos con las columnas correspondientes.

Definición en models.py

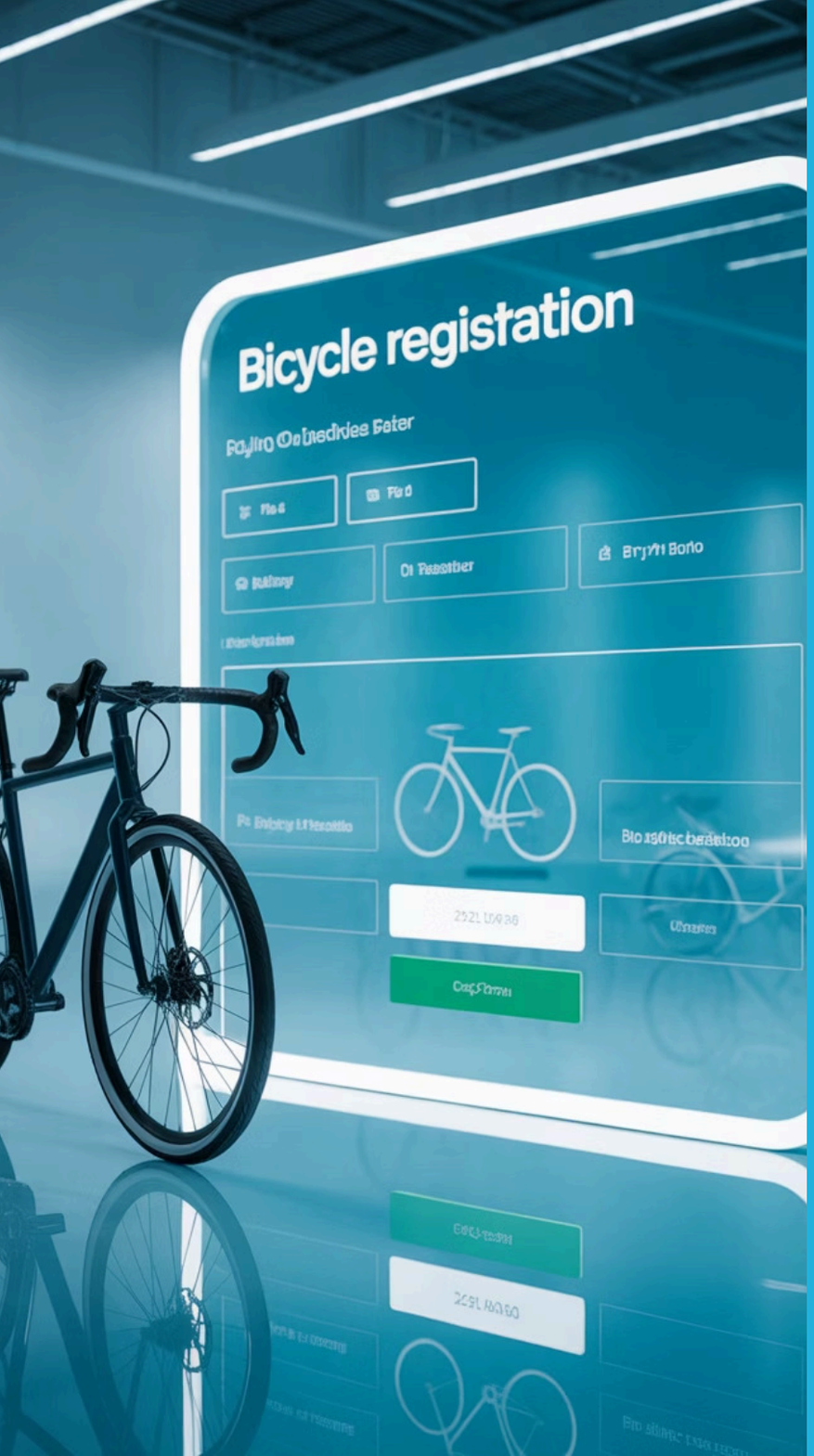
```
from django.db import models

class Bicicleta(models.Model):
    marca = models.CharField(
        max_length=50
    )
    modelo = models.CharField(
        max_length=50
    )
    tipo = models.CharField(
        max_length=20
    )
    precio = models.DecimalField(
        max_digits=10,
        decimal_places=2
    )
    disponible = models.BooleanField(
        default=True
    )
```

👉 Ejercicio Rápido

Agrega un campo `anio` de tipo `IntegerField` al modelo y ejecuta:

```
python manage.py makemigrations
python manage.py migrate
```

Tipos de campos más comunes: CharField (texto corto), TextField (texto largo), IntegerField (números enteros), DecimalField (decimales), BooleanField (verdadero/falso), DateField (fechas).



📄 Formulario con ModelForm

Django proporciona `ModelForm`, una clase especial que genera automáticamente formularios HTML basados en los modelos, ahorrando tiempo y garantizando consistencia entre la base de datos y la interfaz.

Definición en forms.py

```
from django import forms
from .models import Bicicleta

class BicicletaForm(forms.ModelForm):
    class Meta:
        model = Bicicleta
        fields = [
            'marca',
            'modelo',
            'tipo',
            'precio',
            'disponible',
            'anio'
        ]
```

Ventajas Clave

- **Validaciones automáticas:** Django valida tipos de datos y restricciones
- **Menos código:** No necesitas definir cada campo manualmente
- **Sincronización:** Los cambios en el modelo se reflejan automáticamente

🖥️ Vistas CRUD: Crear y Listar

Las operaciones de Crear y Listar son la base de cualquier aplicación CRUD. Veamos cómo implementarlas de forma profesional en Django.

Vista de Creación

```
def crear_bicicleta(request):
    if request.method == 'POST':
        form = BicicletaForm(request.POST)
        if form.is_valid():
            form.save()
            messages.success(request, 'Bicicleta creada exitosamente')
            return redirect('lista_bicicletas')
    else:
        form = BicicletaForm()
    return render(request, 'bicicletas/crear_bicicleta.html', {'form': form})
```

Vista de Listado

```
def lista_bicicletas(request):
    bicicletas = Bicicleta.objects.all().order_by('-id')
    return render(request, 'bicicletas/lista_bicicletas.html',
                  {'bicicletas': bicicletas})
```

👩‍💻 Actividad Guiada

1. **Crea una nueva bicicleta usando el formulario**
2. **Verifica que aparezca en la tabla de listado**
3. **Observa los campos "Editar" y "Eliminar" en cada fila**

🧰 Actualizar y Eliminar

Las operaciones de actualización y eliminación completan el ciclo CRUD. Es crucial implementarlas con validaciones y confirmaciones apropiadas.

Vista de Actualización

```
def actualizar_bicicleta(request, pk):
    bicicleta = get_object_or_404(
        Bicicleta, pk=pk
    )
    if request.method == 'POST':
        form = BicicletaForm(
            request.POST,
            instance=bicicleta
        )
        if form.is_valid():
            form.save()
            return
redirect('lista_bicicletas')
    else:
        form = BicicletaForm(
            instance=bicicleta
        )
    return render(request,
        'bicicletas/actualizar.html',
        {'form': form})
```

Vista de Eliminación

```
def confirmar_eliminar(request, pk):
    bicicleta = get_object_or_404(
        Bicicleta, pk=pk
    )
    if request.method == 'POST':
        bicicleta.delete()
        messages.success(
            request,
            'Eliminado correctamente'
        )
        return
redirect('lista_bicicletas')
    return render(request,

'bicicletas/confirmar_eliminar.html',
        {'bicicleta': bicicleta})
```

👉 **get_object_or_404(): Evita errores 500 devolviendo un 404 cuando no existe el objeto**

👉 **Confirmación de eliminación: Mejora la UX y previene eliminaciones accidentales**

🔐 Seguridad y CSRF en CRUD

La protección CSRF debe estar presente en todos los formularios que modifican datos en el servidor. Esto incluye las operaciones de crear, actualizar y eliminar.

Formulario de Creación

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

Formulario de Edición

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

Confirmación de Eliminación

```
<form method="POST">
  {% csrf_token %}
  <p>¿Eliminar?</p>
  <button>Confirmar</button>
</form>
```

🔒 Error Común: "403 Forbidden"

Este error aparece cuando falta el token CSRF en el formulario. Es una de las causas más frecuentes de problemas en desarrollo Django.

👨‍💻 **Actividad experimental:** Retira temporalmente el `{% csrf_token %}` de un formulario y observa el error 403 que Django genera. Luego, vuélvelo a agregar para restaurar la funcionalidad.

🧭 URLs Completas del CRUD

Una estructura de URLs bien organizada es esencial para mantener el código limpio y facilitar el mantenimiento. Aquí está la configuración completa para nuestro sistema CRUD de bicicletas.

Archivo bicicletas/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
    # Listar todas las bicicletas
    path('', views.lista_bicicletas, name='lista_bicicletas'),

    # Crear una nueva bicicleta
    path('crear/', views.crear_bicicleta, name='crear_bicicleta'),

    # Ver detalle de una bicicleta específica
    path('<int:pk>/', views.detalle_bicicleta, name='detalle_bicicleta'),

    # Actualizar una bicicleta existente
    path('actualizar/<int:pk>/', views.actualizar_bicicleta,
        name='actualizar_bicicleta'),

    # Eliminar una bicicleta (con confirmación)
    path('eliminar/<int:pk>/', views.confirmar_eliminar_bicicleta,
        name='confirmar_eliminar_bicicleta'),
]
```

Convención de nombres: Usar nombres descriptivos en el parámetro `name` facilita referenciar estas URLs desde templates usando `{% url 'nombre' %}`.

💬 Buenas Prácticas en Django CRUD

Seguir estas prácticas profesionales garantiza aplicaciones más seguras, mantenibles y con mejor experiencia de usuario.

- [ ] **Validación de Formularios**

  Siempre verifica `form.is_valid()` antes de guardar datos. Django proporciona validaciones automáticas robustas que debes aprovechar.

- [ ] **Redirecciones Post-Save**

  Después de guardar cambios, usa `redirect()` para evitar reenvíos duplicados al recargar la página (patrón Post-Redirect-Get).

- [ ] **Manejo de Errores**

  Utiliza `get_object_or_404()` en lugar de `.get()` para devolver errores 404 amigables en vez de errores 500 del servidor.

- [ ] **Protección CSRF**

  Incluye `{% csrf_token %}` en todos los formularios que usan método POST. Sin excepciones en aplicaciones de usuario.

- [ ] **Método HTTP Correcto**

  Nunca elimines datos con método GET. Las operaciones destructivas deben usar POST y requerir confirmación explícita.

- [ ] 👉 **Actividad Reflexiva**

  Pregunta: ¿Qué consecuencias de seguridad y experiencia de usuario tendría permitir eliminar registros directamente con un enlace GET sin confirmación?

🧪 Ejercicio Integrador

Ha llegado el momento de aplicar todo lo aprendido en un ejercicio práctico completo que integra todos los conceptos de la clase.

01

Crear Bicicletas

Usa el formulario de creación para agregar 3 bicicletas diferentes con datos completos (marca, modelo, tipo, precio, año).

02

Listar y Verificar

Accede a la vista de listado y confirma que las 3 bicicletas aparecen correctamente en la tabla.

03

Editar Registro

Selecciona una bicicleta y actualiza su precio. Verifica que los cambios se guarden correctamente.

04

Eliminar con Confirmación

Elimina otra bicicleta asegurándote de que el sistema pida confirmación antes de ejecutar la acción.

05

Validación Final

Verifica que todo el flujo CRUD funciona correctamente y que los tokens CSRF están presentes en todos los formularios.

👥 Actividad Grupal

Cada grupo debe explicar brevemente el rol específico de Modelo, Vista y Template en su implementación, identificando qué hace cada componente y cómo se comunican entre sí.

🏁 Cierre y Conclusiones

¡Felicitaciones! Has completado exitosamente el módulo de Tokens, Enrutamiento y CRUD con Django. Repasemos los conceptos clave que dominas ahora.

Tokens CSRF

Proteges tus formularios contra ataques maliciosos implementando correctamente la seguridad CSRF de Django.

Enrutamiento

Configuras sistemas de URLs profesionales con parámetros dinámicos y estructura modular.

CRUD Completo

Implementas operaciones completas de Crear, Leer, Actualizar y Eliminar bajo el patrón MVT de Django.

Mejores Prácticas

Aplicas estándares profesionales de validación, manejo de errores y experiencia de usuario.

🚀 Desafío Final

Implementa un CRUD completo para otro modelo de tu elección (por ejemplo: Clientes, Órdenes o Productos). Asegúrate de incluir:

- **Protección CSRF en todos los formularios**
- **Rutas con parámetros dinámicos**
- **Validaciones y manejo de errores**
- **Confirmación antes de eliminar**

¡Gracias por tu participación! Continúa practicando estos conceptos para dominar el desarrollo web con Django.