



**Cursos gratuitos de programación para  
estudiantes de bachillerato**

**NIVEL 3**

**PROGRAMACIÓN CON DJANGO**

<b>Django: ¡Haciendo que la Web sea Genial!</b>	<b>3</b>
¿Qué es Django?	3
¿Por qué Django?	3
Modelo-Vista-Controlador (MVC)	3
Bases de Datos Relacionales	3
Herramientas Incluidas	3
PIP: Tu Herramienta para Instalar Paquetes en Python 📦	4
Instalación de Django con PIP 🚀	4
Configuración de Pipenv para Paquetes	5
<b>Proyecto 1: clon de Linktree en Django</b>	<b>1</b>
Iniciando el proyecto	1
Creando modelos	2
Registrando el modelo en la página de administración	3
Creando las migraciones	3
Creando las vistas	4
Configurando las URLs	5
Creación de Templates	5
main_page.html (Página principal)	5
user_profile_list.html (Lista de perfiles)	6
user_profile_detail.html (Detalle de perfil)	7
Probar la aplicación	8
1. Crear un superusuario	8
Al ingresar a la sección de creación de perfiles desde el panel de administración, se mostrarán los campos definidos en el archivo models.py. Completá la información correspondiente en cada campo y hacé clic en el botón "Save" para guardar el nuevo perfil en la base de datos.	10
2. Verificar rutas principales	10
3. Consideraciones	11
Desafíos.	11
<b>Proyecto 2: simple Red Social en Django</b>	<b>1</b>
Iniciando el proyecto	1
Crear la app principal	1
Definir modelos	2
Relaciones entre modelos	2
Publicacion	2
Comentario	2
Relación clave	3
Crear migraciones	3
Registrar en el panel de administración	3
Crear superusuario y probar el panel admin	4
Crear formularios	4
Cómo funciona el código	4
Crear vistas	5
Crear URLs	6

Crear templates	7
base.html	7
publicaciones.html	8
nueva_publicacion.html	8
publicacion_detalle.html	9
Probar la aplicación	10
Desafíos	10
<b>Proyecto 3: nuestra tienda Amazon</b>	<b>1</b>
Inicializando el proyecto	1
Crear la app principal	1
Imágenes en Django	2
Instalación con pipenv	2
Configuración en settings.py	2
Definir modelos	2
Relaciones entre modelos	3
Category	3
Product	4
Relación entre modelos	4
Registrar en el panel de administración	4
Crear superusuario y probar el panel admin	5
Crear vistas	6
Crear URLs	7
Crear templates	8
base.html	8
products.html	9
categories.html	9
product_details.html	10
products_by_category.html	11
search.html	11
Probar la aplicación	12
Desafíos	12
<b>Glosario y comandos esenciales en Django</b>	<b>1</b>
Proyecto y app	1
Comandos básicos	1
Archivos comunes	2
Otros términos clave	2

# Django: ¡Haciendo que la Web sea Genial!

## ¿Qué es Django?

Django es como una navaja suiza para desarrolladores web: una herramienta poderosa que hace que la creación de aplicaciones web sea más rápida, fácil y segura. Es un framework web de alto nivel desarrollado en Python, diseñado para facilitar el desarrollo rápido y eficiente de aplicaciones web de cualquier tamaño y complejidad.

## ¿Por qué Django?

- **Ridículamente Rápido:** Django está diseñado para ayudar a los desarrolladores a llevar sus aplicaciones desde la idea hasta la finalización en el menor tiempo posible. Con su arquitectura basada en componentes reutilizables y un conjunto de herramientas integradas, Django acelera el proceso de desarrollo web.
- **Seguridad Garantizada:** Django se toma la seguridad muy en serio. Viene con características integradas que ayudan a los desarrolladores a evitar muchos errores comunes de seguridad, como la inyección de SQL y la falsificación de solicitudes entre sitios (CSRF).
- **Escalabilidad sin Límites:** Algunos de los sitios web más concurridos del mundo confían en Django para manejar su tráfico masivo. Con su capacidad para escalar rápidamente y de manera flexible, Django es una opción ideal para proyectos de cualquier tamaño.

## Modelo-Vista-Controlador (MVC)

Django sigue el patrón de diseño Modelo-Vista-Controlador (MVC), que divide una aplicación web en tres componentes principales:

- **Modelo:** Representa los datos y la lógica de la aplicación.
- **Vista:** Se encarga de la presentación de la información al usuario.
- **Controlador:** Controla la lógica de la aplicación y actúa como intermediario entre el modelo y la vista.

## Bases de Datos Relacionales

Django es compatible con una variedad de bases de datos relacionales, incluyendo PostgreSQL, MySQL, SQLite y Oracle. Utiliza un ORM (Mapeo Objeto-Relacional) para interactuar con la base de datos, lo que facilita el trabajo con datos en forma de objetos Python.

## Herramientas Incluidas

Django viene con una amplia gama de herramientas integradas que facilitan el desarrollo de aplicaciones web:

- **Admin Site:** Una interfaz de administración fácil de usar para gestionar los datos de la aplicación.
- **Autenticación de Usuarios:** Funcionalidades integradas para la autenticación y autorización de usuarios.
- **Formularios y Validaciones:** Herramientas para la creación y validación de formularios web.
- **Seguridad:** Funcionalidades integradas para proteger las aplicaciones web contra vulnerabilidades comunes.

## PIP: Tu Herramienta para Instalar Paquetes en Python

PIP es la herramienta estándar de Python para instalar y administrar paquetes de software. Con PIP, puedes instalar fácilmente bibliotecas y frameworks de Python, incluyendo Django, con solo una línea de comando.

## Instalación de Django con PIP

Siga estos sencillos pasos para instalar Django en tu sistema utilizando PIP:

### Paso 1: Verifica que PIP esté instalado

Abre una terminal o línea de comandos y ejecuta el siguiente comando para verificar si PIP está instalado:

```
pip --version
```

Si PIP está instalado, verás la versión actual. Si no lo está, puedes instalarlo siguiendo las instrucciones en la documentación oficial de Python.

### Paso 2: Instala Django

Una vez que tengas PIP instalado, puedes instalar Django ejecutando el siguiente comando:

```
pip install django
```

Este comando descargará e instalará la última versión estable de Django en tu sistema.

### Paso 3: Verifica la Instalación

Para verificar que Django se haya instalado correctamente, ejecuta el siguiente comando para verificar la versión instalada:

```
django-admin --version
```

Esto debería mostrar la versión de Django que acabas de instalar.

¡Y eso es todo! Ahora tienes Django instalado en tu sistema y estás listo para empezar a construir aplicaciones web asombrosas.

## Configuración de Pipenv para Paquetes

Para gestionar las dependencias de tu proyecto Django de manera más robusta y aislada, puedes utilizar [pipenv](#). Aquí te explico cómo configurarlo:

### Paso 1: Instala Pipenv (si no lo tienes)

Abre tu terminal o línea de comandos y ejecuta:

```
pip install pipenv
```

### Paso 2: Navega al directorio de tu proyecto Django

Si ya tienes un proyecto Django, asegúrate de estar dentro del directorio raíz del proyecto en tu terminal. Si no tienes un proyecto aún, puedes crear uno después de instalar Django:

```
django-admin startproject mi_proyecto
```

```
cd mi_proyecto
```

### Paso 3: Crea el entorno virtual con Pipenv

Ejecuta el siguiente comando en la raíz de tu proyecto:

```
pipenv --python 3
```

Esto creará un archivo [Pipfile](#) y un entorno virtual para tu proyecto.

## Paso 4: Instala Django con Pipenv

Ahora, en lugar de usar `pip install django`, utiliza `pipenv install django`:

```
pipenv install django
```

Pipenv instalará Django dentro del entorno virtual y actualizará los archivos `Pipfile` y `Pipfile.lock`. El archivo `Pipfile` registra las dependencias de alto nivel de tu proyecto, mientras que `Pipfile.lock` contiene un hash de las dependencias exactas instaladas, asegurando compilaciones consistentes.

## Paso 5: Activa el entorno virtual de Pipenv

Antes de ejecutar cualquier comando de Django (como `python manage.py`), activa el entorno virtual:

```
pipenv shell
```

Tu terminal ahora debería mostrar un prefijo indicando que el entorno virtual está activo (por ejemplo, `(mi_proyecto) $`).

## Paso 6: Instala otros paquetes con Pipenv

Para instalar cualquier otra biblioteca o paquete de Python que necesites para tu proyecto Django, utiliza el mismo comando `pipenv install`:

```
pipenv install requests
```

```
pipenv install psycopg2 # Ejemplo para PostgreSQL
```

## Beneficios de usar Pipenv:

- **Gestión de dependencias aislada:** Cada proyecto tiene sus propias dependencias, evitando conflictos entre proyectos.
- **Reproducibilidad:** El archivo `Pipfile.lock` asegura que todos los miembros del equipo utilicen las mismas versiones de los paquetes.
- **Seguridad:** Pipenv ayuda a detectar vulnerabilidades en las dependencias.
- **Flujo de trabajo simplificado:** Combina las funcionalidades de `pip` y `virtualenv`.

Al utilizar `pipenv`, aseguras una gestión de dependencias más organizada y confiable para tu proyecto Django.

# Proyecto 1: clon de Linktree en Django

Linktree es una plataforma que permite reunir y agrupar varios enlaces en una sola página personalizable, facilitando que los usuarios puedan acceder a diferentes destinos digitales desde un único enlace. Es especialmente popular en Instagram, donde solo se permite colocar un enlace en la biografía, por lo que Linktree ayuda a mostrar múltiples enlaces como sitio web, redes sociales, blogs o tiendas virtuales desde un solo URL amigable.

## Inicializando el proyecto

Crea una carpeta en tu directorio local

```
mkdir linktree-django # este commando creará la carpeta
cd linktree-django # este comando ingresará a la carpeta creada
```

Dentro de la carpeta ejecutar el siguiente comando (importante especificar el punto .)

```
django-admin startproject linktree .
```

Tendremos que crear una app dentro del proyecto, llamémosla **profiles**:

```
python manage.py startapp profiles
```

Esto creará una carpeta llamada **profiles** dentro del proyecto. Hasta el momento, la estructura debería verse así:

```

|— linktree
|   |— asgi.py
|   |— __init__.py
|   |— settings.py
|   |— urls.py
|   |— wsgi.py
|— manage.py
|— Pipfile
|— Pipfile.lock
|— profiles
|   |— admin.py
|   |— apps.py
|   |— __init__.py
|   |— migrations/
|   |— models.py
|   |— tests.py
|   |— views.py
```

Django nos permite crear proyectos modulares, como construir con bloques de Lego. Las apps son estos bloques: podemos modificarlos, quitarlos, o agregar nuevos de todos los colores.



Para que nuestro proyecto de Django pueda saber que queremos usar una app (un bloque), debemos agregarla al archivo `settings.py`.

Iremos a la sección `INSTALLED_APPS`.

Aquí se definen los bloques activos. Como verás, Django ya incluye varias apps que funcionan desde el principio.

Añadiremos nuestra nueva app `profiles`:

```
# linktree/settings.py

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'profiles', # ¡Añadir esta nueva línea!
]
```

## Creando modelos

Puedes revisar la documentación de modelos en:

<https://docs.djangoproject.com/en/5.2/topics/db/models/>

Para que nuestro proyecto funcione, necesitamos una configuración de base de datos. Usaremos SQLite, que es la opción predeterminada en Django (aunque también soporta PostgreSQL, MySQL, entre otros).

Modificaremos el archivo `models.py` dentro de la carpeta `profiles`, donde definiremos el esquema de nuestra base de datos.

Piensa que un esquema es una lista de características que todos los perfiles deben cumplir para poder existir en la aplicación.

Crearemos una clase que definirá el modelo, llamémosla `UserProfile`:

```
class UserProfile(models.Model):
    name = models.CharField(max_length=100, default='')
    lastname = models.CharField(max_length=100, default='')
    birthday = models.DateField(null=True)
    phone_number = models.CharField(max_length=20, default='')
    email = models.EmailField(default='')
    short_description = models.TextField(default='')
    profile_picture = models.URLField(default='')
```

Puedes agregar más campos después, si lo deseas.

## Registrando el modelo en la página de administración

Para que nuestro modelo sea editable desde la página de administrador de Django, debemos registrarlo en `admin.py` dentro de la carpeta `profiles`:

```
# profiles/admin.py

from django.contrib import admin

from .models import UserProfile

admin.site.register(UserProfile)
```

## Creando las migraciones

Las migraciones aplican los cambios que realizamos en los modelos a la base de datos.

Ejecuta:

```
python manage.py makemigrations
```

y luego:

```
python manage.py migrate
```

## Creando las vistas

Las vistas son funciones que reciben una petición (**request**) y devuelven una respuesta (**response**).

Ejemplo: para devolver todos los perfiles que existen en la base de datos:

Primero importamos el modelo:

```
from .models import UserProfile
```

Luego definimos la función:

```
# profiles/views.py

from django.shortcuts import render
from .models import UserProfile

def user_profile_list(request):
    profiles = UserProfile.objects.all()
    return render(request, 'user_profile_list.html', {'profiles':
profiles})

def main_page(request):
    return render(request, 'main_page.html')

def user_profile_detail(request, profile_id):
    profile = UserProfile.objects.get(id=profile_id)
    return render(request, 'user_profile_detail.html', {'profile':
profile})
```

### Explicación rápida:

- **UserProfile.objects.all()** consulta todos los perfiles.
- **UserProfile.objects.get(id=profile\_id)** obtiene un perfil específico según su ID.
- Usamos **render()** para devolver una plantilla **.html** junto con los datos necesarios (contexto).

## Configurando las URLs

Las URLs permiten mapear rutas específicas de la web a las vistas que definimos.

Corrigiendo tu guía anterior:

```
# linktree/urls.py

from django.contrib import admin
from django.urls import path
from profiles import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('profiles/', views.user_profile_list,
name='user_profile_list'),
    path('', views.main_page, name='main_page'),
    path('profiles/<int:profile_id>/', views.user_profile_detail,
name='user_profile_detail'),
]
```

## Creación de Templates

Crea una carpeta llamada `templates` dentro de la carpeta `profiles/`. Dentro de ella, crea los siguientes archivos HTML:

`main_page.html` (Página principal)

```
<html>
  <head>
    <title>LinkTree Clone - Tesape'a Django</title>
  </head>
  <body>
    <h1>Welcome to Tesape'a LinkTree 🌳🟡</h1>
    <p>An easy way to connect</p>
    <a href="{% url 'user_profile_list' %}">View all profiles</a>
  </body>
</html>
```

user\_profile\_list.html (Lista de perfiles)

```
<html>
  <head>
    <title>Profile List - Tesape'a Django</title>
  </head>
  <h1>List of users in Tesape'a LinkTree</h1>
  <main>
    {% for profile in profiles %}
    <div>
      <div>
        <h2>{{ profile.name }} {{ profile.lastname }}</h2>
        <a href="{% url 'user_profile_detail' profile.id %}">
          Go to page</a>
        </div>
      </div>
    </div>
    {% endfor %}
  </main>
</html>
```

## user\_profile\_detail.html (Detalle de perfil)

```
<html>
  <head>
    <title>
      {{ profile.name }} {{ profile.lastname }} - Tesape'a Django
    </title>
  </head>
  <body>
    

    <h1>{{ profile.name }} {{ profile.lastname }}</h1>
    <p>{{ profile.short_description }}</p>

    <div>
      <ul>
        <li>
          Birthday: {{ profile.birthday }}
        </li>
        <li>
          Contact Number:
          <a href="tel:{{ profile.phone_number }}"
            >{{ profile.phone_number }}</a>
        </li>
        <li>
          Email:
          <a href="mailto:{{ profile.email }}"
            >{{ profile.email }}</a>
        </li>
      </ul>
    </div>
  </body>
</html>
```

## Probar la aplicación

Una vez completados los templates, configuradas las vistas y definidas las URLs, es necesario ejecutar el servidor, crear un superusuario y verificar el correcto funcionamiento de la aplicación.

### 1. Crear un superusuario

Esto permite acceder al panel de administración de Django para crear y gestionar perfiles fácilmente:

```
python manage.py createsuperuser
```

Ingresar un nombre de usuario, correo electrónico y contraseña cuando se solicite.

Una vez creado, iniciar el servidor:

```
python manage.py runserver
```

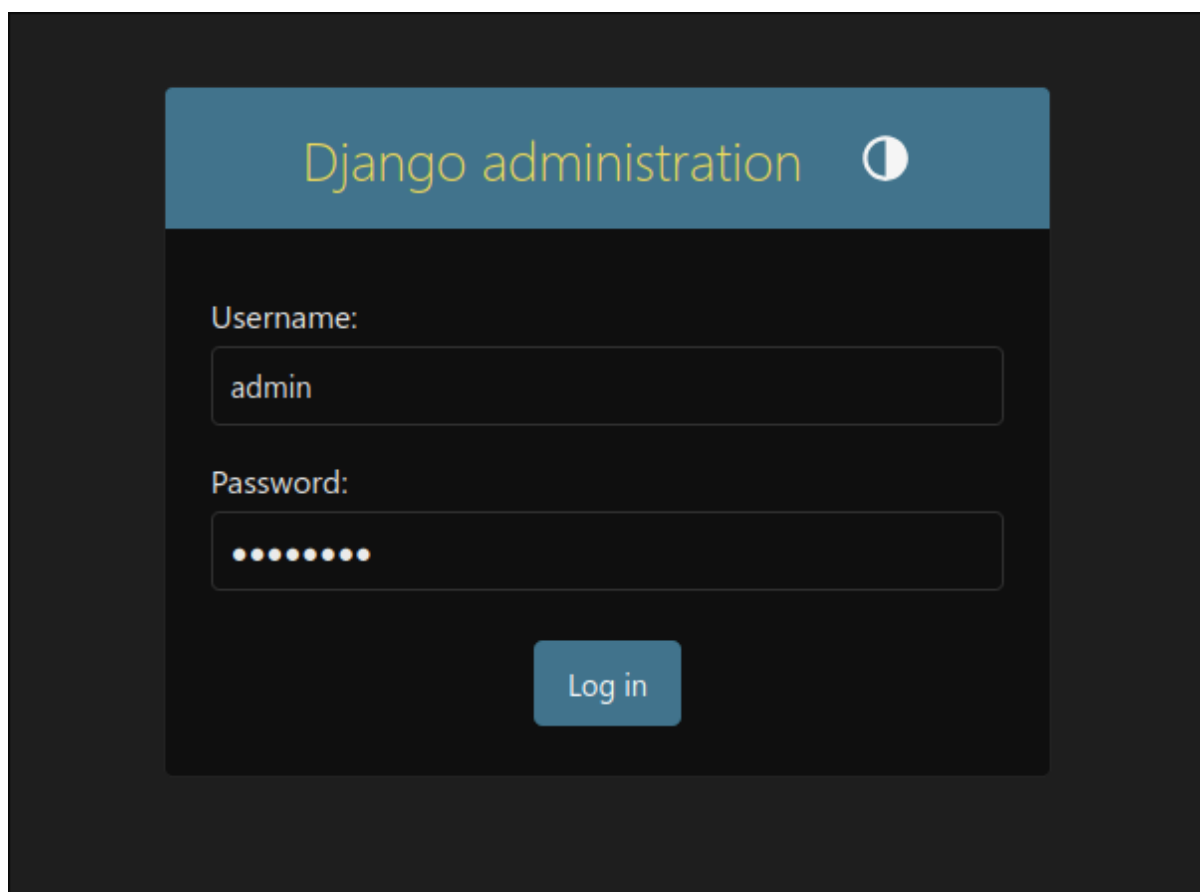
Accede a <http://127.0.0.1:8000/> desde tu navegador y podrás encontrar tu primer proyecto en Django.

#### Welcome to Tesape'a LinkTree

An easy way to connect

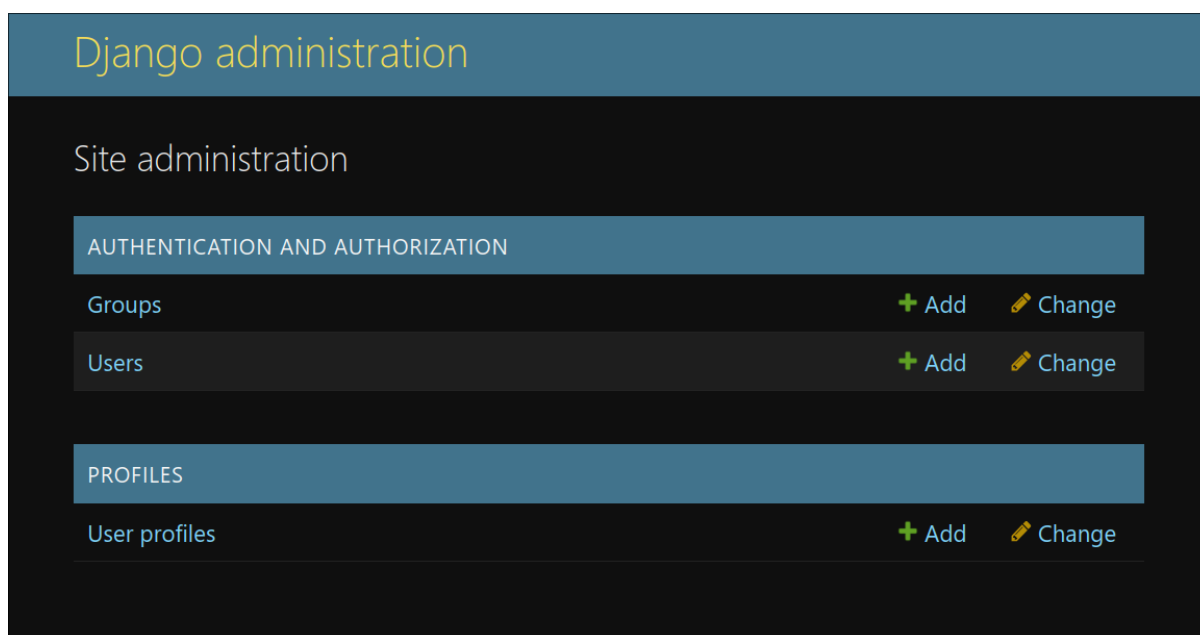
[View all profiles](#)

Acceder a <http://127.0.0.1:8000/admin> e iniciar sesión con las credenciales creadas.



The image shows the Django administration login interface. It features a dark blue header with the text "Django administration" and a moon icon. Below the header, there are two input fields: "Username:" with the value "admin" and "Password:" with a masked password represented by dots. A "Log in" button is positioned below the password field.

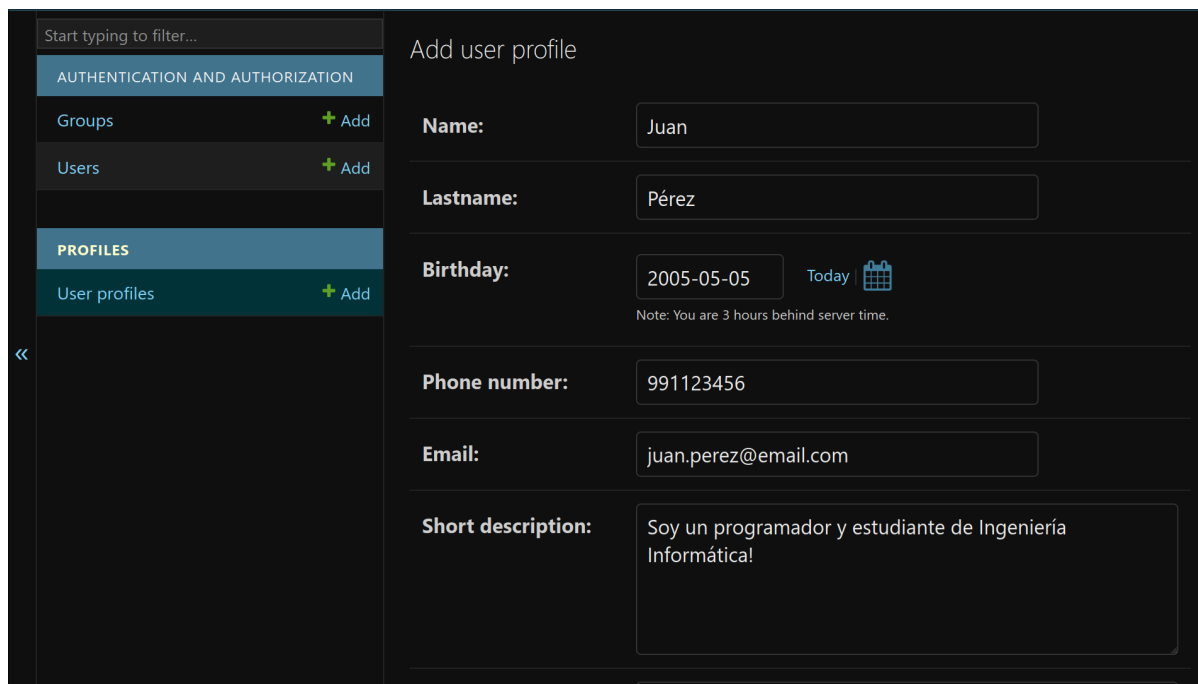
Desde allí, se pueden agregar registros de `UserProfile` y otros modelos necesarios para la prueba. Para agregar un nuevo usuario, dar click a "Add"



The image shows the Django administration site administration page. It has a dark blue header with the text "Django administration". Below the header, the page is titled "Site administration". There are two main sections: "AUTHENTICATION AND AUTHORIZATION" and "PROFILES". The "AUTHENTICATION AND AUTHORIZATION" section contains two rows: "Groups" and "Users", each with a green "+ Add" button and a yellow pencil icon for "Change". The "PROFILES" section contains one row: "User profiles", with a green "+ Add" button and a yellow pencil icon for "Change".



Al ingresar a la sección de creación de perfiles desde el panel de administración, se mostrarán los campos definidos en el archivo `models.py`. Completá la información correspondiente en cada campo y hacé clic en el botón **“Save”** para guardar el nuevo perfil en la base de datos.



Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

PROFILES


User profiles + Add

«

### Add user profile

**Name:**

**Lastname:**

**Birthday:**  Today   
Note: You are 3 hours behind server time.

**Phone number:**

**Email:**

**Short description:**

## 2. Verificar rutas principales

### Ruta raíz /

- Acceder a <http://127.0.0.1:8000/>
- Debe mostrarse la plantilla `main_page.html` con el título y un enlace a la lista de perfiles

### Ruta [/profiles/](#)

- Acceder a <http://127.0.0.1:8000/profiles/>
- Debe mostrarse la plantilla `user_profile_list.html` con todos los perfiles registrados, incluyendo imagen, nombre y enlace a su detalle

## Ruta `/profiles/<id>/`

- Acceder a una URL como `http://127.0.0.1:8000/profiles/1/`
- Debe mostrarse la plantilla `user_profile_detail.html` con los datos completos del perfil seleccionado

### 3. Consideraciones

- Si no hay perfiles registrados, crearlos desde el panel de administración
- Verificar que los archivos HTML estén en `profiles/templates/`
- En `settings.py`, dentro de `TEMPLATES`, debe estar activado `'APP_DIRS': True`
- Si se agregan campos nuevos a los modelos, repetir los comandos de migración

Con estos pasos completados, la aplicación estará lista para seguir ampliándose o ser presentada.

## Desafíos.

1. Agrega nuevos campos en el archivo `models.py`, como por ejemplo un campo de URL para una página web, o un campo para la dirección en donde vive el usuario.
2. Agregar estilos a los templates, puedes añadir un link a un archivo de CSS, usa la línea:

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/water.css@2/out/dark.css">
```

Dentro de tu `<head>` en cada uno de los archivos `.html`

## Proyecto 2: simple Red Social en Django

Este proyecto consiste en construir una red social muy básica donde los usuarios pueden crear publicaciones (posts) y dejar comentarios en cada una.

### Inicializando el proyecto

Crea una carpeta y accede a ella:

```
mkdir red-social-django  
cd red-social-django
```

Inicializa el proyecto:

```
django-admin startproject redsocial .
```

### Crear la app principal

```
python manage.py startapp profiles
```

Registrá la app en `redsocial/settings.py`:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'profiles',  
]
```

## Definir modelos

En `profiles/models.py` agregá los siguientes modelos:

```
from django.db import models

class Publicacion(models.Model):
    contenido = models.TextField()
    fecha = models.DateTimeField(auto_now_add=True)

class Comentario(models.Model):
    contenido = models.TextField()
    fecha = models.DateTimeField(auto_now_add=True)
    publicacion = models.ForeignKey(Publicacion,
    on_delete=models.CASCADE, related_name='comentarios')
```

## Relaciones entre modelos

En este proyecto se definen dos modelos principales:

### Publicacion

Representa un post o entrada en la red social. Tiene:

- `contenido`: el texto de la publicación.
- `fecha`: la fecha y hora en que se creó.

### Comentario

Representa un comentario que se deja en una publicación. Tiene:

- `contenido`: el texto del comentario.
- `fecha`: la fecha y hora en que se creó.
- `publicacion`: una **relación de muchos a uno** con `Publicacion`.

## Relación clave

```
publicacion = models.ForeignKey(Publicacion, on_delete=models.CASCADE,  
related_name='comentarios')
```

Esto significa:

- Cada comentario pertenece a **una sola publicación**.
- Una publicación puede tener **muchos comentarios**.
- Si se borra una publicación, **todos sus comentarios también se eliminan** (`on_delete=models.CASCADE`).
- `related_name='comentarios'` permite acceder a los comentarios desde una publicación usando `publicacion.comentarios.all()`.

En resumen: **una publicación puede tener muchos comentarios, pero cada comentario pertenece a una sola publicación.**

## Crear migraciones

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

## Registrar en el panel de administración

En `profiles/admin.py`:

```
from django.contrib import admin  
from .models import Publicacion, Comentario  
  
admin.site.register(Publicacion)  
admin.site.register(Comentario)
```

## Crear superusuario y probar el panel admin

```
python manage.py createsuperuser  
python manage.py runserver
```

Accedé a <http://127.0.0.1:8000/admin> y logueate con las credenciales del superusuario. Puedes agregar publicaciones y comentarios desde la interfaz de administración.

## Crear formularios

El archivo `forms.py` sirve para definir **formularios basados en modelos**. Django proporciona una clase llamada `ModelForm` que genera automáticamente un formulario a partir de un modelo.

En `profiles/forms.py`:

```
from django import forms  
from .models import Publicacion, Comentario  
  
class PublicacionForm(forms.ModelForm):  
    class Meta:  
        model = Publicacion  
        fields = ['contenido']  
  
class ComentarioForm(forms.ModelForm):  
    class Meta:  
        model = Comentario  
        fields = ['contenido']
```

### Cómo funciona el código

- `PublicacionForm` crea un formulario que solo incluye el campo `contenido` del modelo `Publicacion`.
- Django usa esta clase para mostrar un `<textarea>` en el HTML y para manejar la lógica de guardado.

## Crear vistas

En `profiles/views.py`:

```
from django.shortcuts import render, redirect, get_object_or_404
from .models import Publicacion, Comentario
from .forms import PublicacionForm, ComentarioForm

def publicaciones(request):
    publicaciones = Publicacion.objects.all().order_by('-fecha')
    return render(request, 'publicaciones.html', {'publicaciones':
publicaciones})

def nueva_publicacion(request):
    if request.method == 'POST':
        form = PublicacionForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('publicaciones')
    else:
        form = PublicacionForm()
    return render(request, 'nueva_publicacion.html', {'form': form})

def publicacion_detalle(request, pk):
    publicacion = get_object_or_404(Publicacion, pk=pk)
    comentarios = publicacion.comentarios.all()
    if request.method == 'POST':
        form = ComentarioForm(request.POST)
        if form.is_valid():
            nuevo_comentario = form.save(commit=False)
            nuevo_comentario.publicacion = publicacion
            nuevo_comentario.save()
            return redirect('publicacion_detalle', pk=pk)
    else:
        form = ComentarioForm()
    return render(request, 'publicacion_detalle.html', {
        'publicacion': publicacion,
        'comentarios': comentarios,
        'form_comentario': form
    })
```

## Crear URLs

En `profiles/urls.py`:

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

urlpatterns = [
    path('', views.publicaciones, name='publicaciones'),
    path('nueva/', views.nueva_publicacion, name='nueva_publicacion'),
    path('publicacion/<int:pk>/', views.publicacion_detalle,
name='publicacion_detalle'),
]
```

En `redsocial/urls.py`:

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

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

Hay dos archivos `urls.py` porque cada uno cumple una función distinta:

- `redsocial/urls.py` es el archivo principal del proyecto. Conecta todo y redirige a las apps.
- `profiles/urls.py` organiza las rutas específicas de la app `profiles`.

Esto permite separar las rutas por funcionalidad. El archivo principal delega, y cada app maneja sus propias URLs. Es una forma limpia y escalable de organizar el proyecto.



## Crear templates

Dentro de `profiles/`, crear una carpeta llamada `templates/` y dentro de ella:

### `base.html`

```
<!DOCTYPE html>
<html>
<head>
  <title>Red Social Tesape'a</title>
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/water.css@2/out/water.css" />
</head>
<body>
  <nav>
    <a href="{% url 'publicaciones' %}">Publicaciones</a>
    <a href="{% url 'nueva_publicacion' %}">Nueva publicación</a>
  </nav>
  <main>
    {% block content %}{% endblock %}
  </main>
</body>
</html>
```

## publicaciones.html

```
{% extends 'base.html' %}
{% block content %}
<h1>Publicaciones</h1>
{% for publicacion in publicaciones %}
    <div>
        <p>{{ publicacion.contenido }}</p>
        <p>{{ publicacion.fecha }}</p>
        <a href="{% url 'publicacion_detalle' publicacion.pk %}">Ver
comentarios</a>
    </div>
{% endfor %}
{% endblock %}
```

## nueva\_publicacion.html

```
{% extends 'base.html' %}
{% block content %}
<h1>Nueva publicación</h1>
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Publicar</button>
</form>
{% endblock %}
```

## publicacion\_detalle.html

```
{% extends 'base.html' %}
{% block content %}
<div>
  <p>{{ publicacion.contenido }}</p>
  <p>{{ publicacion.fecha }}</p>
  <h3>Comentarios</h3>
  {% for comentario in comentarios %}
    <div>
      <p>{{ comentario.contenido }}</p>
      <p>{{ comentario.fecha }}</p>
    </div>
  {% endfor %}
  <h3>Agregar comentario</h3>
  <form method="post">
    {% csrf_token %}
    {{ form_comentario.as_p }}
    <button type="submit">Comentar</button>
  </form>
</div>
{% endblock %}
```

## Probar la aplicación

1. Iniciar el servidor:

```
python manage.py runserver
```

2. Ir a:

- <http://127.0.0.1:8000/> para ver las publicaciones
- <http://127.0.0.1:8000/nueva/> para crear una nueva
- <http://127.0.0.1:8000/publicacion/1/> para ver los comentarios de una publicación específica

## Desafíos

1. Agregar nombre de autor a cada publicación y comentario
2. Permitir eliminar publicaciones
3. Agregar estilos con Bootstrap o CSS

## Proyecto 3: nuestra tienda Amazon

Este proyecto consiste en construir una red social muy básica donde los usuarios pueden crear publicaciones (posts) y dejar comentarios en cada una.

### Inicializando el proyecto

Crea una carpeta y accede a ella:

```
mkdir amazon-django  
cd amazon-django
```

Inicializa el proyecto:

```
django-admin startproject amazon .
```

### Crear la app principal

```
python manage.py startapp products
```

Registra la app en [amazon/settings.py](#):

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'products',  
]
```

## Imágenes en Django

Para permitir el uso de imágenes en el modelo `Product` (con el campo `ImageField`) y cargarlas desde el panel de administración de Django, es necesario instalar la librería **Pillow**, que permite el manejo de archivos de imagen.

### Instalación con pipenv

Instalar Pillow en el entorno virtual del proyecto:

```
pipenv install pillow
```

Verifica que **Pillow** esté en el archivo **Pipfile** bajo `[packages]`:

```
[packages]
pillow = "*"

```

### Configuración en `settings.py`

Importamos la dependencia `os`, para manipular direcciones del sistema

```
import os
```

Asegúrate de tener las siguientes líneas en tu archivo de configuración para servir archivos multimedia en desarrollo:

```
MEDIA_URL = 'media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Y en `urls.py` del proyecto, agregar lo siguiente para servir imágenes en modo de desarrollo:

```
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    # otras rutas
] + + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Con esto configurado, las imágenes cargadas a través del panel admin o formularios se almacenarán en la carpeta `media/uploads/product/`, según la configuración del campo `image` en el modelo `Product`.

## Definir modelos

En `products/models.py` agregá los siguientes modelos:

```
from django.db import models
```

```
class Category(models.Model):
    name = models.CharField(max_length=100, unique=True)
    description = models.TextField(blank=True)

    class Meta:
        verbose_name_plural = "categories"

    def __str__(self):
        return self.name

class Product(models.Model):
    category = models.ForeignKey(Category, on_delete=models.CASCADE,
related_name='products')
    name = models.CharField(max_length=200)
    description = models.TextField(blank=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    image = models.ImageField(upload_to='uploads/produc/')

    def __str__(self):
        return f'{self.name}, {self.category}'
```

## Relaciones entre modelos

### Category

Representa una categoría general a la que pertenecen varios productos. Tiene:

- **name**: el nombre de la categoría (por ejemplo: Electrónica, Ropa, Libros).
- **description**: una descripción opcional sobre la categoría.

Cada categoría puede estar relacionada con muchos productos.

## Product

Representa un producto disponible en el ecommerce. Tiene:

- **name**: el nombre del producto.
- **description**: una descripción del producto.
- **price**: el precio del producto.
- **image**: una imagen del producto.
- **category**: una relación de muchos a uno con el modelo Category.

## Relación entre modelos

Cada producto pertenece a una sola categoría.

Una categoría puede tener muchos productos.

Esta relación se establece mediante una clave foránea (**ForeignKey**) en el modelo Product que apunta al modelo Category.

## Crear migraciones

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

## Registrar en el panel de administración

En **products/admin.py**:

```
from django.contrib import admin
from .models import Category, Product

admin.site.register(Category)
admin.site.register(Product)
```



## Crear superusuario y probar el panel admin

```
python manage.py createsuperuser  
python manage.py runserver
```

Accedé a <http://127.0.0.1:8000/admin> y logueate con las credenciales del superusuario.

## Crear vistas

En `products/views.py`:

```
from django.shortcuts import render
from django.db.models import Q
from .models import Product, Category

def get_categories(request):
    categories = Category.objects.all()
    return render(request, 'categories.html', {'categories': categories})

def get_products_by_category(request, category_id):
    category = Category.objects.get(id=category_id)
    products = Product.objects.filter(category=category)
    return render(request, 'products_by_category.html', {'category':
category, 'products': products})

def get_products(request):
    products = Product.objects.all()
    return render(request, 'products.html', {'products': products})

def get_products_details(request, product_id):
    product = Product.objects.get(id=product_id)
    return render(request, 'product_details.html', {
        'product': product,
    })

def get_search(request):
    query = request.GET.get('search-input', '')
    products = Product.objects.filter(
        Q(name__icontains=query) | Q(description__icontains=query) |
Q(category__name__icontains=query)
    ) if query else Product.objects.none()
    return render(request, 'search.html', {'products': products, 'query':
query})
```

## Crear URLs

En `products/urls.py`:

```
from django.contrib import admin
from django.urls import path
from . import settings
from django.conf.urls.static import static
from products import views

urlpatterns = [
    path('', views.get_products, name="base"),
    path('admin/', admin.site.urls),
    path('products/', views.get_products, name="get_products"),
    path('categories/', views.get_categories, name="get_categories"),
    path('search/', views.get_search, name="get_search"),
    path('products/<int:product_id>', views.get_products_details,
name="get_products_details"),
    path('categories/<int:category_id>/products/',
views.get_products_by_category, name="get_products_by_category"),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

## Crear templates

Dentro de `products/`, crear una carpeta llamada `templates/` y dentro de ella:

### `base.html`

```
<!DOCTYPE html>
<html>
  <title>Amazon - Tesape'a</title>
</head>
<body>
  <header>
    <a href="{% url 'base' %}">
      <h1>Amazon - Tesape'a</h1>
    </a>
    <nav>
      <a href="{% url 'get_products' %}">Productos</a>
      <a href="{% url 'get_categories' %}">Categorías</a>
      <form action="{% url 'get_search' %}" method="get"
style="display:inline;">
        <input type="text" name="search-input"
placeholder="Buscar..." required>
        <button type="submit">Buscar</button>
      </form>
    </nav>
  </header>
  <main>
    {% block content %}{% endblock %}
  </main>
</body>
</html>
```

## products.html

```
{% extends "base.html" %}
{% block content %}
<h1>Products</h1>
<div>
    {% for product in products %}
        <div>
            <a href="{% url 'get_products_details' product.id %}">
                
                <h3>{{ product.name }}</h3>
            </a>
            <p>{{ product }}</p>
            <p>PYG {{ product.price }}</p>
        </div>
    {% endfor %}
</div>
{% endblock %}
```

## categories.html

```
{% extends "base.html" %}

{% block content %}
<h1>Categories</h1>
<ul>
    {% for category in categories %}
        <li>
            <a href="{% url 'get_products_by_category' category.id
%}">
                {{ category.name }}
            </a>
            <br>
            <small>{{ category.description }}</small>
        </li>
    {% endfor %}
</ul>
{% endblock %}
```

## product\_details.html

```
{% extends "base.html" %}
{% block content %}
<main>
  <article>
    <div>
      <figure>
        
      </figure>
      <div>
        <div>
          <h1>{{ product.name }}</h1>
          <p>
            Categoría: {{ product.category.name }}
          </p>
          {% if product.description %}
          <p>{{ product.description }}</p>
          {% endif %}
          <h3>Precio:</h3>
          <p>PYG {{ product.price }}</p>
        </div>
        <button>
           Comprar
        </button>
      </div>
    </div>
  </article>
</main>
{% endblock %}
```

## products\_by\_category.html

```
{% extends "base.html" %}
{% block content %}
<h1>Products in "{{ category.name }}"</h1>
<div>
    {% for product in products %}
        <div>
            <a href="{% url 'get_products_details' product.id %}">
                
                <h3>{{ product.name }}</h3>
            </a>
            <p>{{ product.description|truncatewords:15 }}</p>
            <p><strong>Price:</strong> ${{ product.price }}</p>
        </div>
    {% empty %}
        <p>No products found in this category.</p>
    {% endfor %}
</div>
{% endblock %}
```

## search.html

```
{% extends "base.html" %}
{% block content %}
<h2>You searched for "{{ query }}"</h2>
<div>
    {% for product in products %}
        <div>
            <a href="{% url 'get_products_details' product.id %}">
                
                <h3>{{ product.name }}</h3>
            </a>
            <p>{{ product }}</p>
        </div>
    {% endfor %}
</div>
{% endblock %}
```

## Probar la aplicación

2. Iniciar el servidor:

```
python manage.py runserver
```

3. Ir a:

- <http://127.0.0.1:8000/> para ver los productos
- <http://127.0.0.1:8000/categories/> para ver las categorías
- <http://127.0.0.1:8000/products/1/> para ver los productos específicos
- Busca algo con la barra de búsqueda de la tienda

## Desafíos

4. Agregar sección de comentarios (Puedes inspirarte de la clase pasada de Red Social)
5. Añadir calificaciones por parte de los usuarios
6. Añadir palabras clave a cada producto para mejorar la búsqueda
7. Agregar estilos con Bootstrap o CSS



# Glosario y comandos esenciales en Django

## Proyecto y app

### Proyecto

Conjunto principal de configuraciones y archivos que representan tu sitio web. Contiene `settings.py`, `urls.py`, etc.

### App

Módulo reutilizable que representa una funcionalidad específica dentro del proyecto. Por ejemplo, una app de perfiles o blog.

## Comandos básicos

### Crear un nuevo proyecto Django

```
django-admin startproject nombre_del_proyecto .
```

(El punto final indica que se cree en el directorio actual)

**Crear una nueva app** `python manage.py startapp nombre_de_la_app`

**Iniciar el servidor de desarrollo** `python manage.py runserver`

Abrir en el navegador: `http://127.0.0.1:8000` ó `localhost:8000`

**Crear una super cuenta de administrador** `python manage.py createsuperuser`

(Se solicitarán usuario, email y contraseña)

**Acceder al panel de administración** `http://127.0.0.1:8000/admin`

(Iniciá sesión con el superuser que creaste)

**Crear archivos de migración a partir de modelos** `python manage.py makemigrations`

**Aplicar las migraciones a la base de datos** `python manage.py migrate`

## Archivos comunes

### `settings.py`

Archivo de configuración global del proyecto. Aquí se definen apps instaladas, bases de datos, ubicación de archivos estáticos, etc.

### `urls.py`

Define las rutas (endpoints) del sitio. Enlaza URLs con vistas.

### `models.py`

Contiene clases que representan las tablas de tu base de datos.

### `views.py`

Funciones o clases que procesan las solicitudes HTTP y devuelven respuestas.

### `templates/`

Carpeta donde se guardan los archivos HTML. Usados con `render()` en las vistas.

## Otros términos clave

### **Migrations (Migraciones)**

Traducción de tus modelos a instrucciones SQL que modifican la base de datos.

### **ORM (Object Relational Mapper)**

Django convierte clases Python (`models.py`) en tablas reales dentro de la base de datos (como SQLite o PostgreSQL).

### **Admin panel**

Interfaz gráfica integrada de Django para administrar el contenido del sitio.

### **Contexto**

Diccionario de variables que se pasa desde una vista a un template para renderizar contenido dinámico.