PROYECTO TESAPEÁ

Cursos gratuitos de programación para estudiantes de bachillerato

NIVEL 3

PROGRAMACIÓN CON DJANGO

integradevs
SOFTWARE SOLUTIONS

PROYECTO TESAPEÁ

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>
      {% endfor %}
      </main>
</html>
```

`user_profile_detail.html` (Detalle de perfil)

```
<html>
    <head>
        <title>
            {{ profile.name }} {{ profile.lastname }} - Tesape'a Django
        </title>
    </head>
    <body>
            <img
                src="{{ profile.profile_picture }}"
                alt="{{ profile.name }} {{ profile.lastname }}"
            />

            <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
```

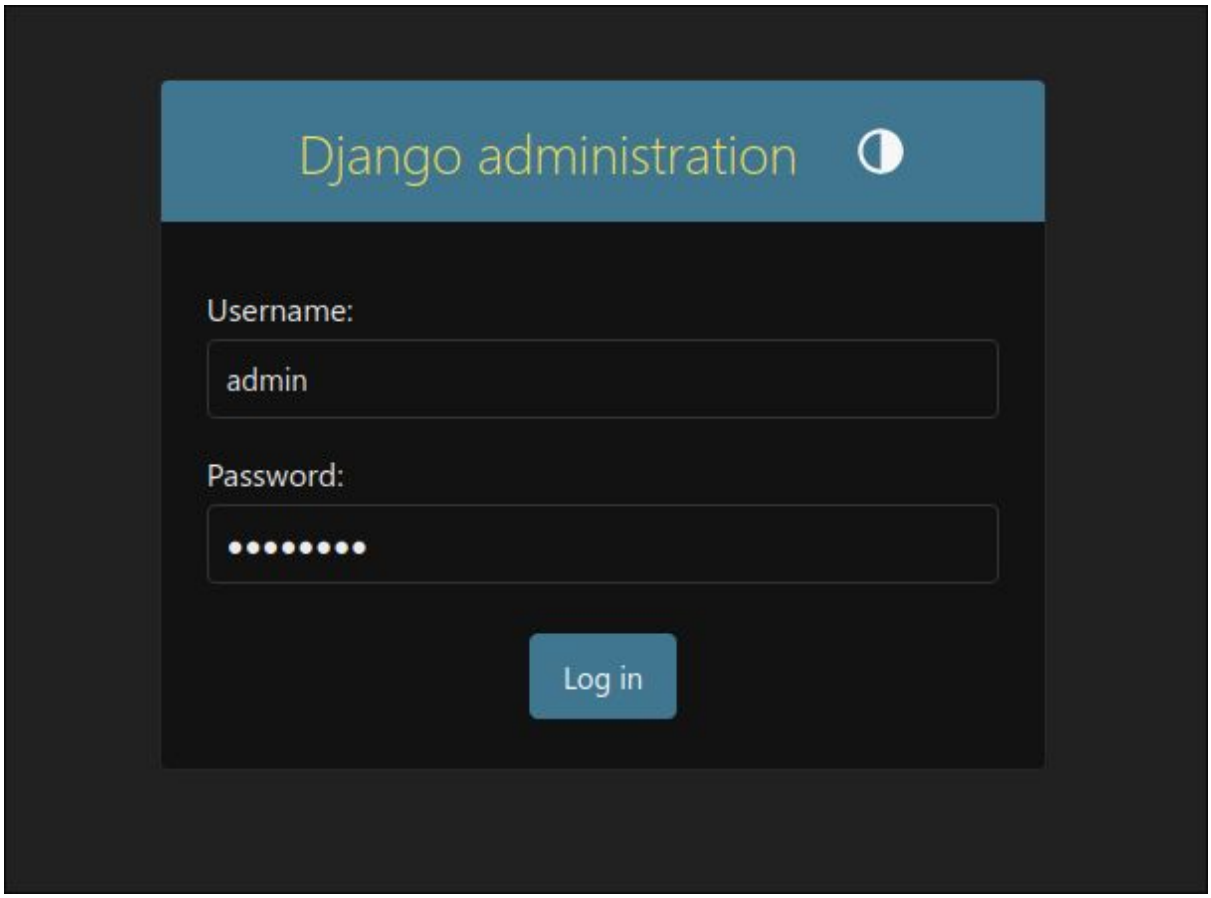
Ingresar un nombre de usuario, correo electrónico y contraseña cuando se solicite.

Una vez creado, iniciar el servidor:

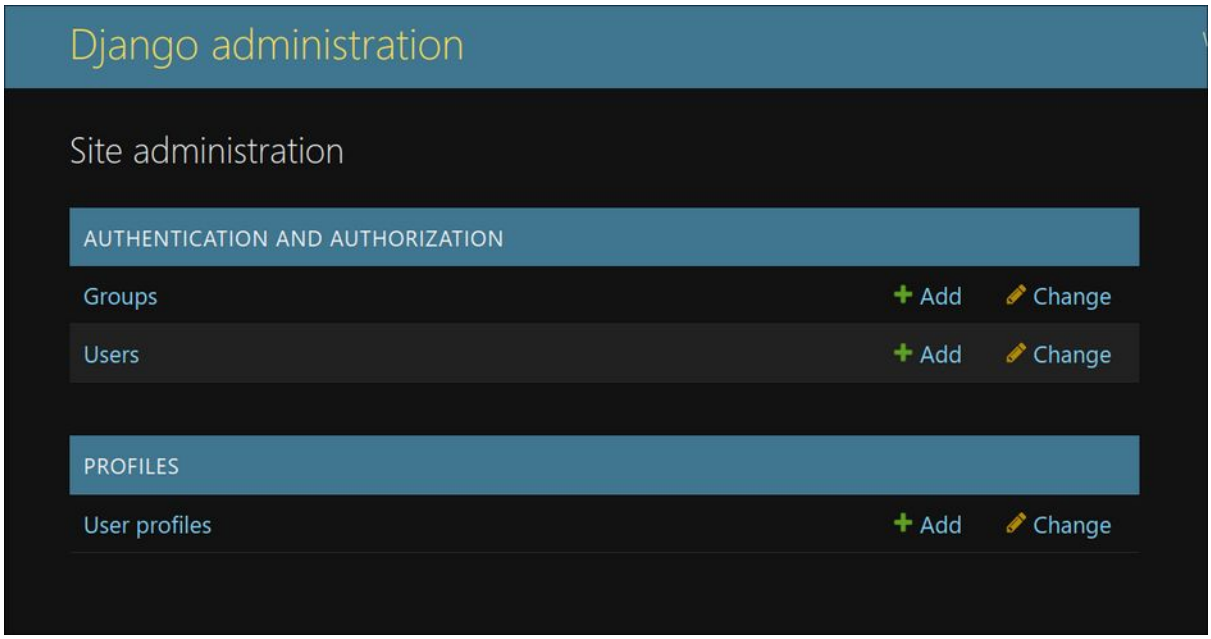
```
python manage.py runserver
```

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

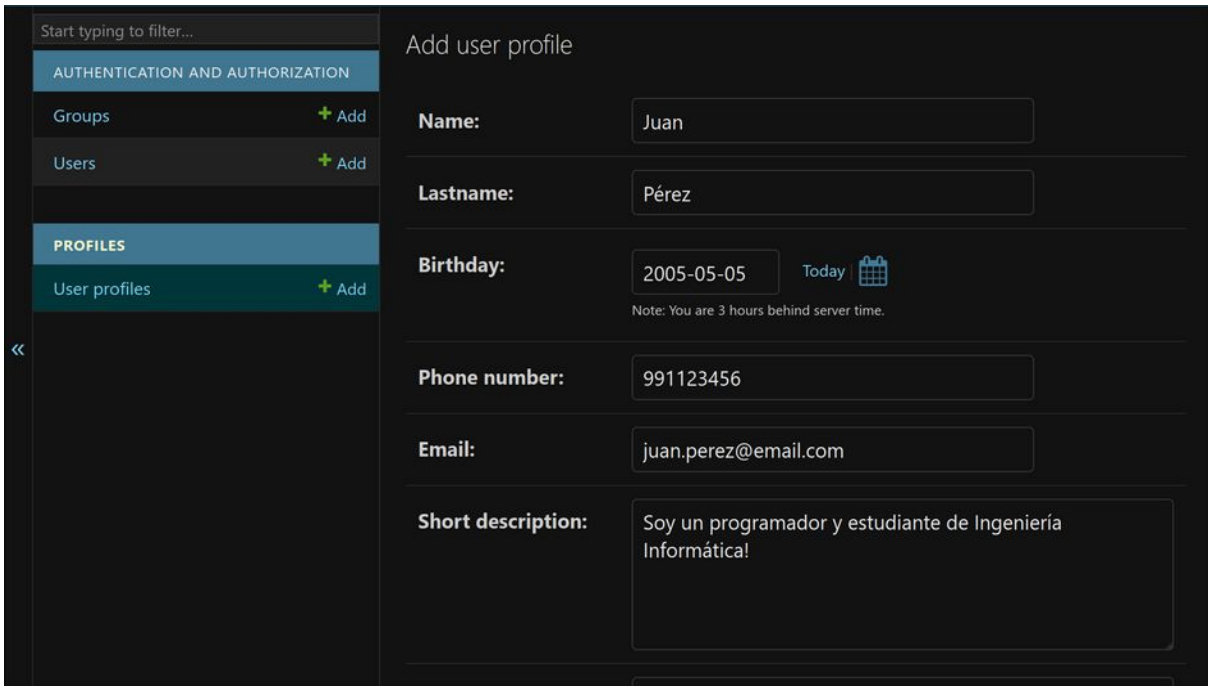
Acceder a `http://127.0.0.1:8000/admin` e iniciar sesión con las credenciales creadas.



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



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.



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.

Desafios.

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`

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.