



Fundamentos de Django

Centro de Servicios y Gestión Empresarial
SENA Regional Antioquia

Contextualización

Frameworks



Un **framework** es una estructura reutilizable de trabajo compuesta por código, librerías, patrones y herramientas predefinidas que facilitan el desarrollo de aplicaciones. Funciona como un esqueleto sobre el cual puedes construir tu propio programa.

"El framework da el cómo y tú le das el qué."

Tipos de Framework



Los **frameworks** pueden clasificarse según el propósito o tipo de aplicación:

1. **Frameworks Web:** Permiten construir aplicaciones web (frontend, backend o full-stack). **Ejemplo:** Django, Flask, FastAPI
2. **Frameworks de Pruebas:** Automatizan pruebas de software (unitarias, integración, etc.). **Ejemplo:** pytest, unittest, behave4.
3. **Frameworks de Ciencia de Datos / IA:** Facilitan análisis, modelado y despliegue de modelos de machine learning. **Ejemplo:** TensorFlow, PyTorch, Scikit-learn

Beneficios de usar un Framework



Ahorro de tiempo:
Reduce la necesidad
de escribir código
repetitivo.

Estandarización:
Impone buenas
prácticas y estructura
organizada.

Seguridad: Muchos
frameworks tienen
protección contra
errores comunes (e.g.,
inyección SQL en
Django).

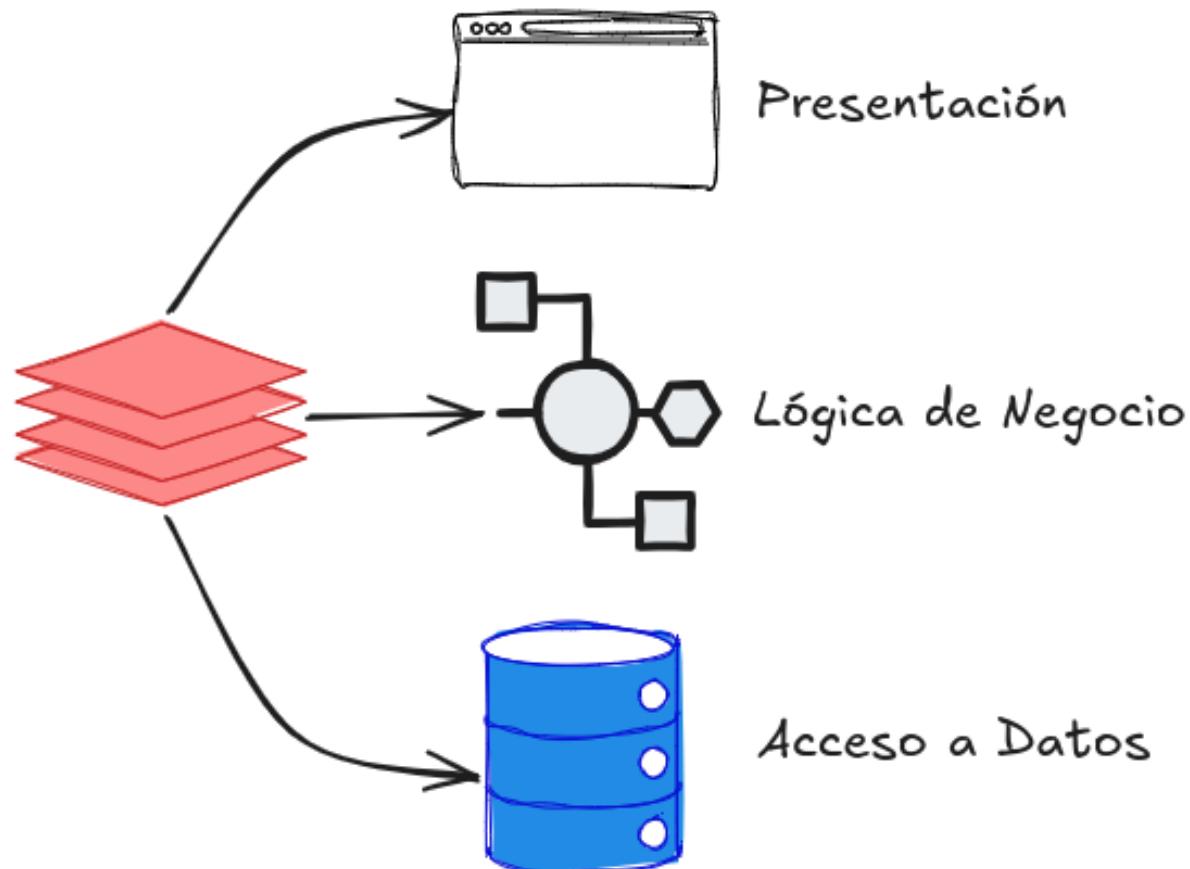
Escalabilidad: Facilita
el crecimiento del
proyecto a largo plazo.

Comunidad activa:
Acceso a
documentación,
plugins, foros y
soporte.

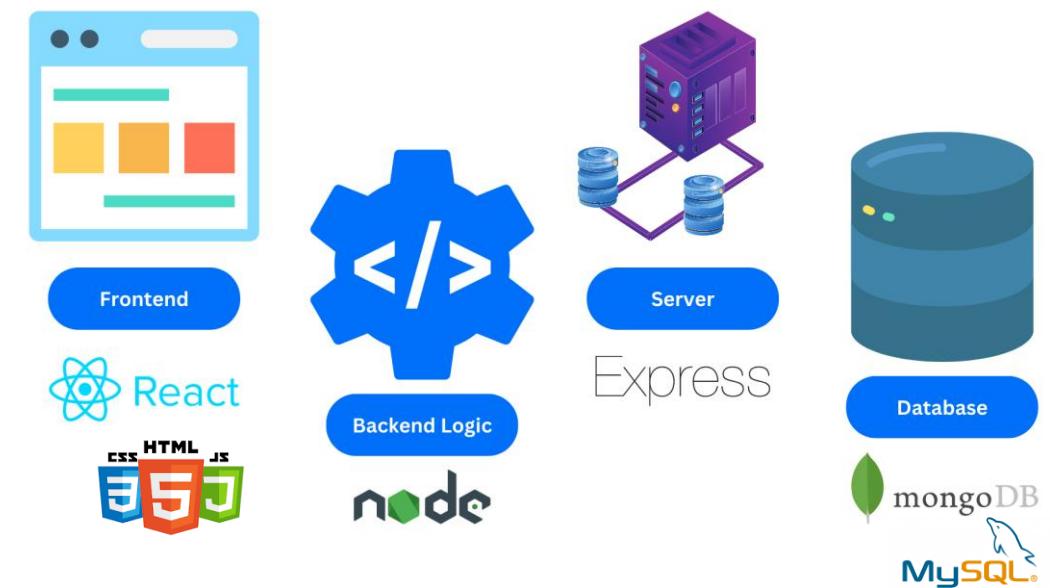
Arquitecturas Desarrollo Web



Arquitectura en Capas

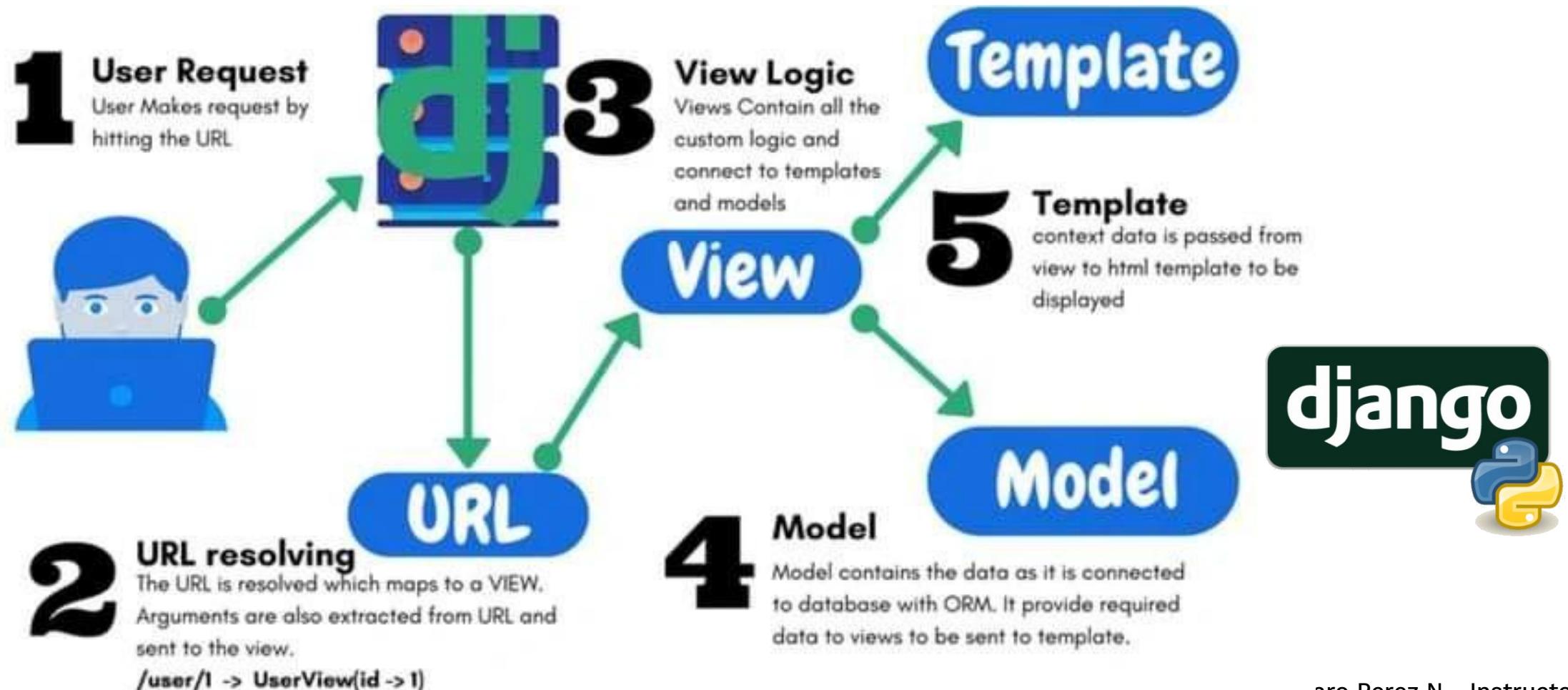


La arquitectura por capas es un estilo de diseño de software que organiza el sistema en niveles jerárquicos, donde cada capa tiene una responsabilidad específica y se comunica solo con las capas adyacentes.



Arquitecturas Desarrollo Web

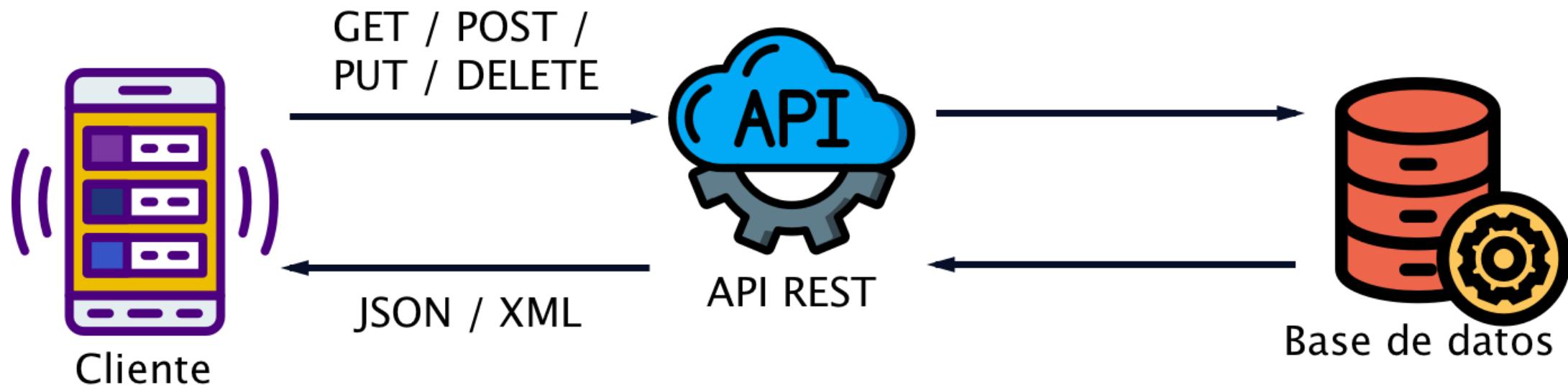
Arquitectura Modelo-Vista-Plantilla



Arquitecturas Desarrollo Web

Arquitectura Cliente-Servidor (RESTful APIs):

Es un modelo de comunicación donde un **cliente** realiza solicitudes a un **servidor**, que responde con datos o resultados. Las RESTful APIs siguen principios REST y utilizan protocolos como HTTP para intercambiar información en formato JSON, permitiendo una interacción eficiente.



Framework de Django

Django



Django es un framework web de alto nivel en Python que permite desarrollar sitios web de forma rápida, segura y con una estructura ordenada. Fue creado con el principio de "*no te repitas*" (*Don't Repeat Yourself - DRY*) y busca automatizar tareas comunes del desarrollo web.



"Django te da todo lo que necesitas para construir aplicaciones web completas sin reinventar la rueda."

¿Qué tipo de aplicaciones puedes crear con Django?

- Blogs, foros, portales de noticias
- Sistemas de gestión de contenido (CMS)
- APIs REST (usando Django REST Framework)
- Tiendas online y marketplaces
- Sistemas empresariales internos
- Plataformas educativas y LMS
- Sitios institucionales o gubernamentales

Django - Características principales



Característica	Descripción
Basado en MVC (MVT)	Usa el patrón Modelo–Vista–Template, variante de MVC.
ORM incorporado	Interactúa con bases de datos sin escribir SQL puro.
Admin automático	Genera una interfaz de administración con un par de líneas de código.
Gestión de usuarios	Maneja autenticación, sesiones y permisos.
Rutas limpias (URLs)	Mapeo elegante entre URLs y vistas con expresiones regulares o path().
Plantillas HTML	Motor de plantillas integrado para generar interfaces dinámicas.
Escalabilidad	Soporta alto tráfico con arquitecturas robustas.

Django - Características principales



Django viene con una gran cantidad de funcionalidades integradas:

- Sistema de autenticación
- ORM para bases de datos (SQLite, PostgreSQL, MySQL)
- Panel de administración
- Control de formularios
- Soporte para múltiples idiomas
- Pruebas automáticas
- Protección contra CSRF, XSS, SQL Injection

Django - Estructura



```
mi_proyecto/
    └── mi_app/
        ├── models.py          # App principal (puedes tener varias)
        ├── views.py           # Definición de tablas de BD (ORM)
        ├── urls.py            # Lógica de las respuestas
        └── templates/          # Rutas propias
                                # Archivos HTML

    └── mi_proyecto/
        ├── settings.py       # Configuración global del proyecto
        ├── urls.py           # Configuración general (BD, apps, rutas, etc.)
                                # Rutas globales

    └── db.sqlite3           # Base de datos (por defecto SQLite)
    └── manage.py            # Comando principal para administrar el proyecto
```

Django - Estructura

```
empresa/
  └── empresa/
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
      # Carpeta del proyecto (configuración principal)
      # Hace que esta carpeta sea un paquete Python
      # Configuración para el despliegue ASGI
      # Configuraciones generales del proyecto (apps, BD, rutas, etc.)
      # Rutas generales del proyecto (incluye rutas de las apps)
      # Configuración para el despliegue WSGI (usado en producción)

  └── empleados/
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── models.py
    ├── tests.py
    ├── views.py
    └── migrations/
      └── __init__.py
        # App encargada de la gestión de empleados
        # Registro del modelo Empleado en el panel admin
        # Configuración interna de la app
        # Modelo de datos de Empleado (ORM)
        # Archivo para pruebas automatizadas
        # (Opcional, si más adelante deseas vistas específicas de empleados)
        # Archivos de migración de Django para la base de datos

  └── landing/
    ├── __init__.py
    ├── apps.py
    ├── urls.py
    ├── views.py
    └── templates/
      └── landing/
        └── landing.html # Template HTML con los empleados y productos
      # App encargada de mostrar la landing page pública
      # Ruta de acceso a la landing page (home del sitio)
      # Vista que combina empleados y productos

  └── db.sqlite3
  └── manage.py
  └── env/
    # Base de datos local generada por Django
    # Script para ejecutar comandos del proyecto (runserver, migrate, etc.)
    # Carpeta del entorno virtual (excluida del control de versiones)
```

Django

Landing Page de Productos



Tu estilo deportivo comienza aquí

Descubre nuestra colección de ropa deportiva de alta calidad.

[Ver Productos](#)

Nuestros Productos



Camiseta Deportiva

Camiseta transpirable para entrenamiento

\$29.99

[Agregar al carrito](#)



Pantalón Deportivo

Pantalón cómodo para running

\$39.99

[Agregar al carrito](#)



Chaqueta Deportiva

Chaqueta ligera para actividades al aire libre

\$49.99

[Agregar al carrito](#)

Sobre Nosotros

Somos una tienda especializada en ropa deportiva de alta calidad.

En SportStyle nos dedicamos a proporcionar la mejor selección de ropa deportiva para todos los niveles de actividad física.

Nuestra misión es ayudar a los atletas y entusiastas del deporte a alcanzar sus objetivos con comodidad y estilo.

- Productos de alta calidad
- Envío rápido
- Atención personalizada



Contáctanos

Nombre

Email

Mensaje

[Enviar Mensaje](#)

Información de Contacto

Dirección: Calle Principal #123, Ciudad

Teléfono: (123) 456-7890

Email: info@sportstyle.com

Siguenos en redes sociales



Django - Prototipo



Nuestros Productos



Camiseta Deportiva

Camiseta transpirable para entrenamiento

\$29.99

[Agregar al carrito](#)



Pantalón Deportivo

Pantalón cómodo para running

\$39.99

[Agregar al carrito](#)



Chaqueta Deportiva

Chaqueta ligera para actividades al aire libre

\$49.99

[Agregar al carrito](#)

Sobre Nosotros

Somos una tienda especializada en ropa deportiva de alta calidad.

En SportStyle nos dedicamos a proporcionar la mejor selección de ropa deportiva para todos los niveles de actividad física. Nuestra misión es ayudar a los atletas y entusiastas del deporte a alcanzar sus objetivos con comodidad y estilo.

- Productos de alta calidad
- Envío rápido
- Atención personalizada



Django – Desarrollo de la aplicación



1. Crear entorno virtual e instalar Django

bash

Copiar

Editar

```
python -m venv env
source env/bin/activate      # En Windows: env\Scripts\activate
pip install django
```

Crear entornos virtuales es una práctica esencial en el desarrollo de software, ya que permite aislar las dependencias de cada proyecto. Esto evita conflictos entre bibliotecas y versiones, y garantiza que cada aplicación funcione de manera consistente sin afectar otros proyectos o el sistema operativo.

Django – Desarrollo de la aplicación



2. Crear proyecto y aplicación

```
bash                                         ⌂ Copiar ⌋ Editar

django-admin startproject empresa
cd empresa
python manage.py startapp empleados
python manage.py startapp productos
python manage.py startapp landing
```

- **django-admin startproject empresa:** Este comando crea un nuevo proyecto Django llamado empresa. Genera una estructura básica con archivos de configuración del proyecto.
- **python manage.py startapp empleados:** Este comando crea una nueva aplicación Django llamada empleados dentro del proyecto. Una aplicación en Django es un componente modular y reutilizable.

3. Registrar la app *empleados*, *productos* y *landing* en *settings.py* del proyecto

```
# Application definition
INSTALLED_APPS = [
    'empleados',
    'productos',
    'landing',
]
```

Para que Django la reconozca en el proyecto cada uno de los módulos debe agregarlos en su configuración:

- En *empresa/settings.py*, busca *INSTALLED_APPS* y agrega '*empleados*':

4. Crear el modelo de empleado

Edita *empleados/models.py*:

- El ORM de Django permite trabajar con la base de datos usando clases de Python en lugar de escribir sentencias SQL directamente.
- Cada clase del archivo models.py representa una tabla en la base de datos, y cada atributo de esa clase representa una columna.

```
from django.db import models

class Empleado(models.Model):
    nombre = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    rol = models.CharField(max_length=50)
    foto_url = models.URLField()

    def __str__(self):
        return self.nombre
```



5. Realizar las migraciones

```
bash                                     ⚒ Copiar   ⚒ Editar

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

- **python manage.py makemigrations:** Este comando crea un migración en el(los) modelos nuevos / actualizados del proyecto y lo almacena en la carpeta *migrations*.
- **python manage.py migrate:** Ejecuta los archivos de migración generados por *makemigrations* y aplica los cambios en la base de datos.

6. Crear superusuario (opcional)

bash

Copiar

Editar

```
python manage.py createsuperuser
```

- El comando *python manage.py createsuperuser* en Django se usa para crear un usuario administrador con acceso total al panel de administración del proyecto.
- Datos de acceso: Nombre de usuario, Correo electrónico y Contraseña (dos veces)
- <http://localhost:8000/admin>

7. Registrar el modelo en el panel de administrador

En **empleados/admin.py**:

```
from django.contrib import admin
from .models import Empleado

admin.site.register(Empleado)
```

Permite gestionar el Empleado desde la interfaz web de administración, desde la URL: <http://localhost:8000/admin> . Esta acción aplica para cada modulo o modelo del proyecto (productos, clientes, entre otros.)

8. Crear vista para la landing page

En `landing/views.py`

```
from django.shortcuts import render
from empleados.models import Empleado

def landing_page(request):
    empleados = Empleado.objects.all()
    return render(request, 'landing/landing.html', {
        'empleados': empleados,
    })
```

Cuando un usuario acceda a esta vista, verá la página HTML – Landing Page que muestra los productos y empleados almacenados en la base de datos.

9.1 Crear archivo de rutas para la app landing

Crea landing/urls.py

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

urlpatterns = [
    path('', landing_page, name='landing'),
]
```

En Django, cada vez que alguien visita una sección de la aplicación (*landing page*), el sistema necesita saber qué función ejecutar para mostrar esa página. Esto se logra configurando las URLs, es decir, asociando una dirección web con una vista (función o clase).

9.2 Enlazar rutas en el archivo principal

Crea empresa/urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('landingpage.urls')),
] + static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Se definen las URLs principales del proyecto. Se incluye la URL para el panel de administración (admin/) y también se enlazan las URLs definidas en la aplicación landing page utilizando include(). Esto permite organizar mejor las rutas, manteniendo las URLs específicas de cada aplicación dentro de su propio archivo.

9.3 Configurar los archivos estáticos

Crea empresa/settings.py

```
STATIC_URL = 'static/'  
  
import os  
STATICFILES_DIRS = [  
    os.path.join(BASE_DIR, 'static'),  
]
```

STATIC_URL y STATICFILES_DIRS son configuraciones de Django que definen cómo y desde dónde se sirven los archivos estáticos (como CSS, JavaScript e imágenes) durante el desarrollo.

10. Crear el template HTML

Crea la estructura de carpetas *landing/templates/landing/landing.html* y el archivo HTML:

```
landing/
  └── templates/
    └── landing/
      └── landing.html
```



10. Crear el template HTML



```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Empleados</title>
</head>
<body>
    <h1 style="text-align:center;">Lista de Empleados</h1>
    <div class="container">
        {% for emp in empleados %}
        <div class="card">
            
            <h3>{{ emp.nombre }}</h3>
            <p>Email: {{ emp.email }}</p>
            <p>Rol: {{ emp.rol }}</p>
        </div>
        {% endfor %}
    </div>
</body>
</html>
```

11. Ejecutar el servidor

```
bash                                     ⌂ Copiar ⌋ Editar

python manage.py runserver
```

En el navegador ejecuta:

- **http://127.0.0.1:8000** para ver la lista de empleados
- **http://127.0.0.1:8000/admin** para añadir empleados desde el panel de administración



G R A C I A S

Presentó: Alvaro Pérez Niño

Instructor Técnico

Correo: aperezn@sena.edu.co

<http://centrodesserviciosygestionempresarial.blogspot.com/>

Línea de atención al ciudadano: 01 8000 910270

Línea de atención al empresario: 01 8000 910682



www.sena.edu.co