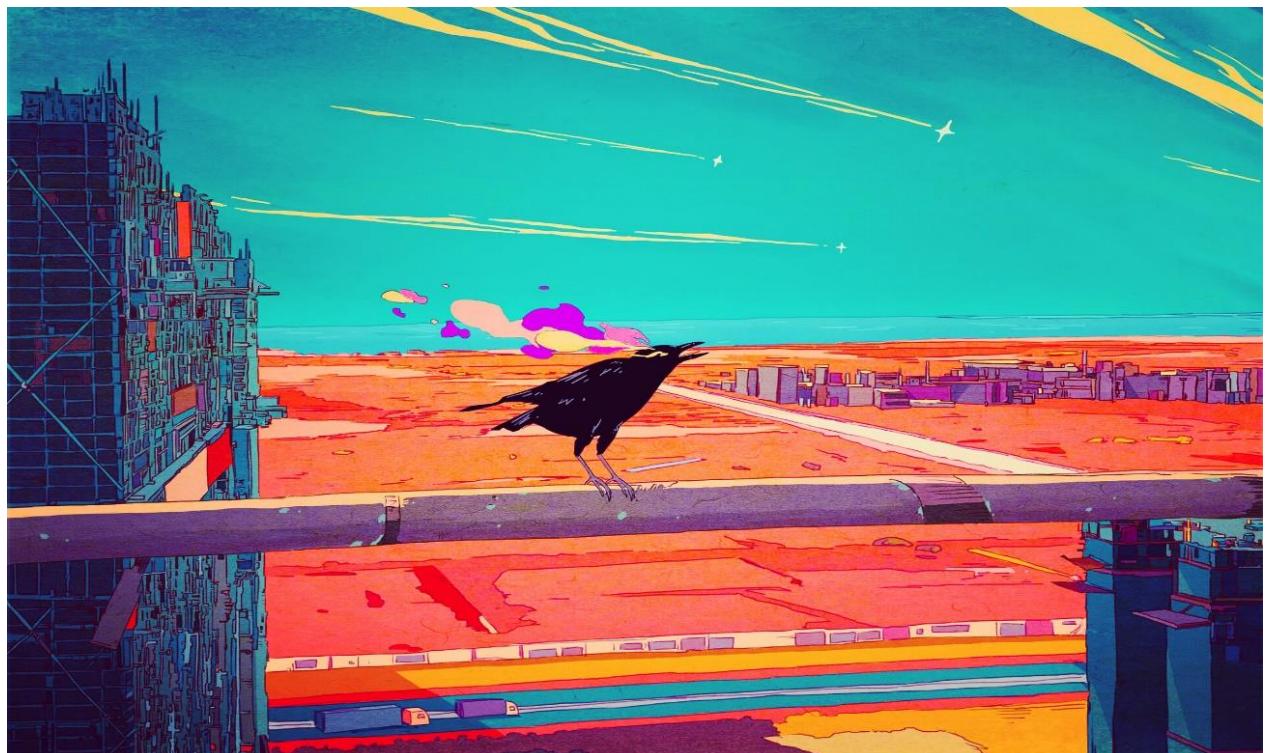


# Manual

# Django



J2D-MBUE

Daquilema Pullay Jhefferson David



## Contenido

1. Comandos que se usan en Django .....	3
2. Instalación de Django en Fedora .....	4
3. Crear la estructura del proyecto Django en Fedora (Ejemplo base).....	6
4. Abrir el proyecto en Visual Studio Code y configurar settings.py.....	7
5. Configurar archivos principales de las aplicaciones.....	10
6: Activar los modelos y crear la base de datos.....	15
7. Ejecutar el servidor .....	19
8. Conocimientos necesarios antes de empezar .....	21
<b>¿Cómo funciona la estructura MVT (Modelo-Vista-Template) en Django? .....</b>	21
<b>Cómo fluye la información en Django con MVT.....</b>	21
<b>¿Y cómo se accede a las páginas específicas? .....</b>	21
<span style="color: #808000;">📁</span> <b>¿Dónde se define eso en el proyecto? .....</b>	22
9. Crear Crud con HTML .....	27
10 Crear el Consulta con la función inicio() en views.py.....	33
<span style="color: #808000;">🧠</span> <b>¿Qué hace esto? .....</b>	41
<span style="color: #808000;">📦</span> <b>Estructura base del formulario.....</b>	41
<span style="color: #808000;">🛠</span> <b>¿Y los botones?.....</b>	42
11 Crear el Insertar para ingresar nuevos cargos (CREATE) .....	45
.....	55
12 Crea el Update para Actualizar datos en Django .....	55
13 Crear delete para Eliminar un cargo .....	64
14 ¿Cómo relacionar dos tablas con ForeignKey? .....	68
15. SUBIR UN PROYECTO A GITLAB DESDE FEDORA.....	90
<span style="color: #808000;">📘</span> <b>Manual detallado para usar una plantilla HTML descargada en un proyecto Django.....</b>	98
16 Manual paso a paso para validar formularios con jQuery Validation.....	106



## 1. Comandos que se usan en Django

**cd** se usa para ingresar a una carpeta desde la terminal.

**cd ..** sirve para retroceder una carpeta o volver al nivel anterior.

**mkdir** se usa para crear una nueva carpeta.

**django-admin startproject** crea la estructura base de un nuevo proyecto Django. Esto incluye archivos como manage.py y una carpeta con configuraciones del proyecto.

**django-admin startapp** crea una aplicación dentro del proyecto. En Django, una aplicación es como un módulo o parte del sistema (por ejemplo, una app para usuarios, otra para productos, etc.).

Se recomienda crear una carpeta llamada aplicaciones (o apps) y dentro de ella ir creando todas las aplicaciones del proyecto. Esto permite mantener una estructura más organizada, sobre todo en proyectos grandes.

**python manage.py runserver** levanta el servidor local para que puedas probar tu proyecto en el navegador (normalmente en [http://127.0.0.1:8000](http://127.0.0.1:8000)).

**python manage.py makemigrations** genera los archivos de migración que indican a Django qué cambios se deben hacer en la base de datos (por ejemplo, crear una tabla nueva o modificar campos).

**python manage.py migrate** ejecuta las migraciones y aplica los cambios reales en la base de datos.

“Es importante ejecutar primero makemigrations y luego migrate. Si haces migrate sin haber hecho makemigrations, pueden generarse errores porque Django no sabrá qué cambios aplicar. Primero se genera el plan con makemigrations, y luego se ejecuta con migrate.”

**python manage.py createsuperuser** crea un usuario administrador para acceder al panel de administración de Django. Este usuario tiene todos los permisos y puede gestionar los datos desde la interfaz web.



## 2. Instalación de Django en Fedora

Para poder trabajar con Django, lo primero que necesitas es tener Python instalado. En Fedora, Python ya viene instalado por defecto, así que no hace falta instalarlo manualmente. Lo único que necesitas es instalar pip, que es la herramienta que permite descargar e instalar paquetes de Python, como Django.

### Paso 1: Instalar pip para Python 3.13

En la terminal, ejecuta el siguiente comando para instalar o actualizar pip:

```
python3.13 -m ensurepip --upgrade
```

Este comando revisa si pip está disponible para esa versión de Python y, si no está, lo instala.

### Paso 2: Instalar Django

Una vez que ya tienes pip, puedes instalar Django escribiendo en la terminal:

```
pip install django
```

Con esto, ya tienes Django listo para crear proyectos y aplicaciones.

### Paso 3: Instalar Visual Studio Code

Para escribir y editar el código de manera más cómoda, se recomienda usar Visual Studio Code (VS Code), que es un editor de código muy completo.

No se recomienda instalar VS Code desde la tienda de Fedora, porque esa versión tiene limitaciones. Por ejemplo, no se ejecuta con permisos de administrador, y eso puede causar problemas al guardar archivos o al ejecutar extensiones.

Por eso, lo mejor es instalarlo desde el repositorio oficial de Microsoft.

#### 1. Agregar el repositorio de Visual Studio Code

Escribe estos comandos uno por uno en la terminal:

##### Comando 1:

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
```

##### Comando 2(Todo):

```
sudo sh -c 'echo -e "[code]\nname=Visual Studio Code\nbaseurl=https://packages.microsoft.com/yumrepos/vscode\nelement=1\ngpgcheck=1\ngpgkey=https://packages.microsoft.com/keys/microsoft.asc">\n/etc/yum.repos.d/vscode.repo'
```



Esto le indica al sistema dónde encontrar la versión oficial de VS Code.

## 2. Instalar VS Code

Después de haber agregado el repositorio, puedes instalar VS Code con este comando:

```
sudo dnf install code
```

Una vez instalado, ya puedes abrirlo desde el menú o escribiendo code en la terminal.



### 3. Crear la estructura del proyecto Django en Fedora (Ejemplo base)

Una vez que tienes Django instalado, lo ideal es comenzar con una estructura organizada para tu proyecto. A continuación te mostramos un ejemplo de cómo podrías hacerlo, pero recuerda que este es solo un modelo base: tú puedes cambiar los nombres, rutas y estructuras según lo que necesites para tu proyecto.

Recomendación:

Muchos desarrolladores crean una carpeta general donde guardan todos sus proyectos Django, y dentro de cada proyecto, una carpeta llamada Aplicaciones para mantener ordenadas todas las apps que vayan creando. Esto no es obligatorio, pero ayuda mucho cuando tu sistema empieza a crecer.

```
mbue@fedora:~$ mkdir ProyectosDJ
mbue@fedora:~$ cd ProyectosDJ
mbue@fedora:~/ProyectosDJ$ django-admin startproject trabajosDD
mbue@fedora:~/ProyectosDJ$ cd trabajosDD
mbue@fedora:~/ProyectosDJ/trabajosDD$ mkdir Aplicaciones
mbue@fedora:~/ProyectosDJ/trabajosDD$ cd Aplicaciones
mbue@fedora:~/ProyectosDJ/trabajosDD/Aplicaciones$ django-admin startapp Empleos
mbue@fedora:~/ProyectosDJ/trabajosDD/Aplicaciones$ django-admin startapp Empresas
```

Ejemplo práctico en la terminal:

1. Crear una carpeta general para tus proyectos Django: **mkdir ProyectosDJ**
2. Entrar en esa carpeta: **cd ProyectosDJ**
3. Crear un nuevo proyecto llamado trabajosDD (puedes cambiar ese nombre por el que tú quieras): **django-admin startproject trabajosDD**
4. Entrar en la carpeta del proyecto: **cd trabajosDD**
5. Crear una carpeta llamada Aplicaciones donde irán todas tus apps:  
**mkdir Aplicaciones**
6. Entrar en esa carpeta de aplicaciones: **cd Aplicaciones**
7. Crear una app llamada Empleos (puede ser cualquier otro nombre, según lo que estés desarrollando): **django-admin startapp Empleos**
8. Crear otra app llamada Empresas: **django-admin startapp Empresas**
9. Regresar a la carpeta principal del proyecto: **cd ..**
10. Verificar que la estructura quedó bien organizada: **ls**
11. Esto debería mostrar algo como: **Aplicaciones manage.py trabajosDD**

Recuerda:

Los nombres como trabajosDD, Empleos y Empresas son solo ejemplos. Puedes usar los nombres que tengan más sentido según el tipo de sistema que estés creando y puedes crear la cantidad e aplicaciones que necesites.



## 4. Abrir el proyecto en Visual Studio Code y configurar settings.py

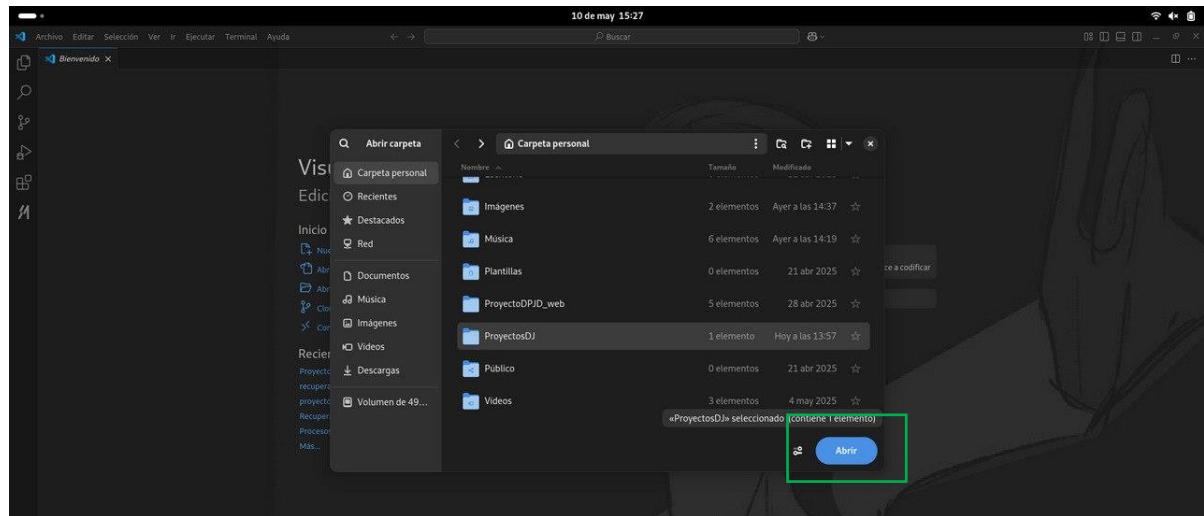
Una vez que ya creaste tu proyecto y tus aplicaciones, es momento de comenzar a trabajar desde un entorno más cómodo. En este caso usaremos Visual Studio Code (VS Code), que es uno de los editores más usados para desarrollo en Django.

Abrir la carpeta del proyecto en VS Code

Abre Visual Studio Code.

Haz clic en Archivo > Abrir carpeta.

Busca y selecciona la primera carpeta que creaste (en el ejemplo: ProyectosDJ).



Importante:

Al abrir la carpeta del proyecto, verás una estructura similar a esta:

ProyectosDJ/

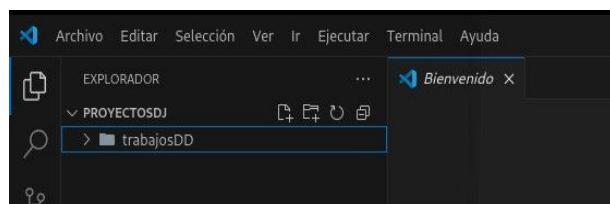
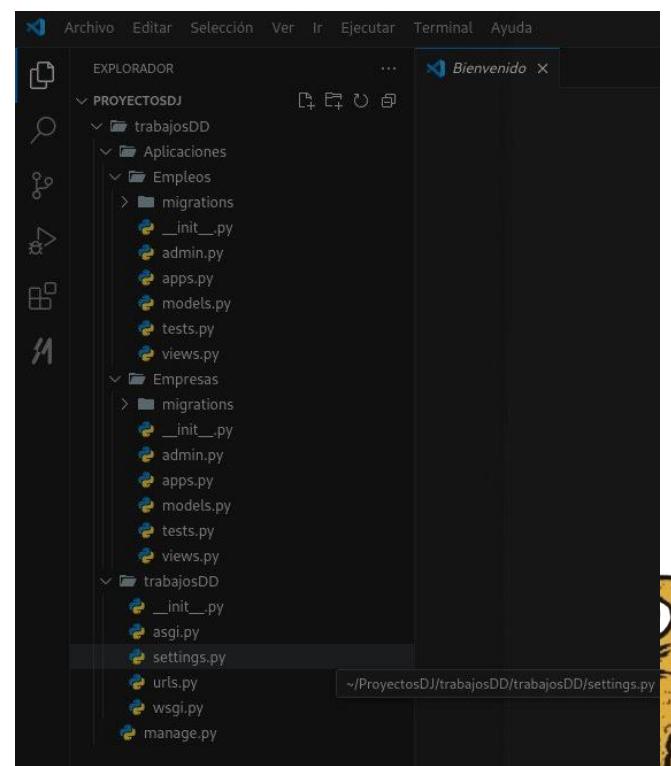
  └── trabajosDD/

    ├── Aplicaciones/

    ├── manage.py

    └── trabajosDD/

      └── settings.py

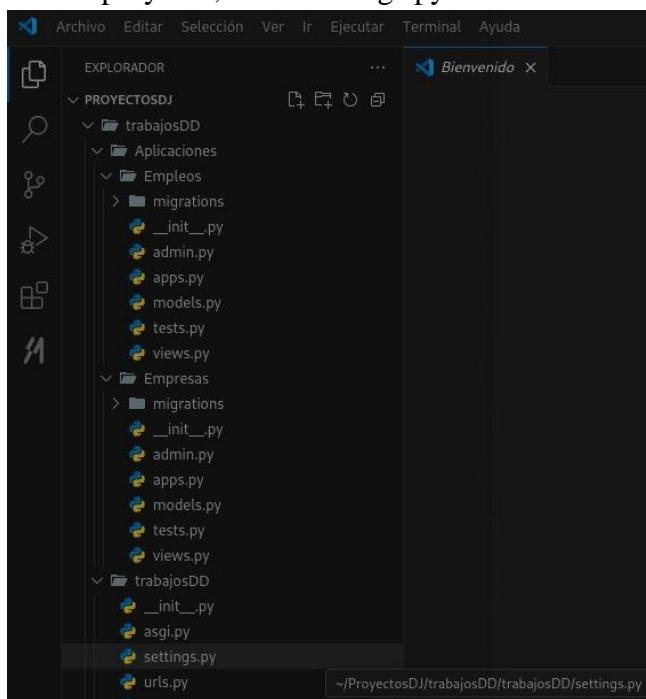


Esto puede parecer repetido, pero es normal. Al crear un proyecto Django, se genera una carpeta con el mismo nombre del proyecto (trabajosDD) dentro de la principal. Ahí dentro se guardan los archivos de configuración del proyecto, como settings.py.

## Configurar las aplicaciones en settings.py

Ahora vamos a registrar nuestras aplicaciones en el archivo settings.py, que se encuentra dentro de la segunda carpeta con el nombre del proyecto.

En el explorador de VS Code, abre la ruta:  
trabajosDD > trabajosDD > settings.py



Busca la lista INSTALLED\_APPS, que normalmente está a partir de la línea 33.

```
34     INSTALLED_APPS = [  
35         'django.contrib.admin',  
36         'django.contrib.auth',  
37         'django.contrib.contenttypes',  
38         'django.contrib.sessions',  
39         'django.contrib.messages',  
40         'django.contrib.staticfiles',
```

Allí se deben registrar todas las apps que creaste para que Django las reconozca.

Ejemplo con nuestras apps: [App nueva agregada](#)

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

¿Qué significa esto?

Aquí le estamos diciendo a Django que también debe tomar en cuenta nuestras aplicaciones personalizadas (Empleos y Empresas), que están dentro de la carpeta Aplicaciones.



## Configurar la base de datos y la zona horaria

Más abajo en el mismo archivo settings.py están otras configuraciones importantes:

- ◆ Base de datos

Por defecto, Django usa SQLite, que es una base de datos liviana ideal para pruebas o proyectos pequeños.

```
80 DATABASES = {  
81     'default': {  
82         'ENGINE': 'django.db.backends.postgresql',  
83         'NAME': 'sindicato',  
84         'USER': 'postgres',  
85         'PASSWORD': 'secret'  
86     }  
87 }
```

<pre>DATABASES = {     'default': {         'ENGINE':             'django.db.backends.sqlite3',         'NAME': BASE_DIR / 'db.sqlite3',     } }</pre>	Puedes cambiar el nombre 'db.sqlite3' por otro más personalizado, como 'empleos.sqlite3'.
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------

Esto solo afectará el nombre del archivo, no su funcionamiento.

- ◆ Idioma y zona horaria

```
03  
04  
05     # Internationalization  
06     # https://docs.djangoproject.com/en/5.2/topics/i18n/  
07  
08 LANGUAGE_CODE = 'es-ec'  
09  
10 TIME_ZONE = 'America/Guayaquil'  
11  
12 USE_I18N = True  
13  
14 USE_TZ = True  
15  
16  
17     # Static files (CSS, JavaScript, Images)  
18     # https://docs.djangoproject.com/en/5.2/howto/static-files/  
19  
20 STATIC_URL = 'static/'  
21  
22     # Default primary key field type  
23     # https://docs.djangoproject.com/en/5.2/ref/settings/#default-auto-field  
24  
25 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'  
26
```



<p>Casi al final del archivo, usualmente desde la línea 107, verás esto:</p> <pre>LANGUAGE_CODE = 'en-us' TIME_ZONE = 'UTC'</pre> <p>En este caso, si estás en Ecuador, es recomendable cambiarlo a:</p> <pre>LANGUAGE_CODE = 'es-ec' TIME_ZONE = 'America/Guayaquil'</pre>	<p>¿Por qué es importante esto? Porque al usar migraciones o guardar datos que tengan que ver con fechas y horas, Django tomará como referencia la zona horaria correcta. Esto evita errores en la base de datos y asegura que todo funcione como se espera.</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## 5. Configurar archivos principales de las aplicaciones

Cuando creas una aplicación (app) en Django con el comando:

**python manage.py startapp nombreapp**

Django genera automáticamente una carpeta llamada nombreapp con varios archivos importantes por defecto. Estos archivos son necesarios para que la app funcione correctamente y cada uno tiene una función específica.

Sin embargo, por razones de organización, es común que los desarrolladores coloquen sus apps dentro de una carpeta llamada Aplicaciones o similar, para mantener el proyecto más ordenado cuando hay varias apps. Esto significa que la ruta de la app cambia y hay que ajustar algunas configuraciones.

Ahora vamos a configurar los 3 archivos importantes en cada aplicación Django que creamos. Esto permite que el proyecto reconozca y use correctamente las apps y sus modelos (las tablas que queremos que se creen en la base de datos).

Así que configuraremos 3 archivos que se crean automáticamente en cada aplicación, pero los vamos a revisar y modificar si hace falta:

### A. Configurar apps.py

- **Qué es:**

Este archivo contiene la configuración principal de tu app. Django usa esta configuración para identificar tu app dentro del proyecto. 🌟 ¿Dónde está ese archivo?

Está dentro de la carpeta de tu aplicación. Por ejemplo:

- Aplicaciones/Empleos/apps.py
- Aplicaciones/Empresas/apps.py

- **Qué pasa si creamos las apps en una carpeta:**

Si creas tu app dentro de una carpeta llamada Aplicaciones para organizar mejor



tu proyecto (por ejemplo Aplicaciones/nombreapp), debes actualizar esta propiedad name para que refleje la ruta completa, así:

- **¿Por qué hay que hacer este cambio?**

Porque Django necesita la ruta exacta para importar y cargar correctamente la app. Si no actualizas name, Django no encontrará la app y dará error.

### 📌 ¿Para qué sirve?

Este archivo le dice a Django cómo se llama tu aplicación y en qué carpeta está exactamente.

### 📝 ¿Qué hay que escribir?

Django lo crea así:

```
class EmpleosConfig(AppConfig):
```

```
    default_auto_field =  
        'django.db.models.BigAutoField'  
  
    name = 'Empleos'
```

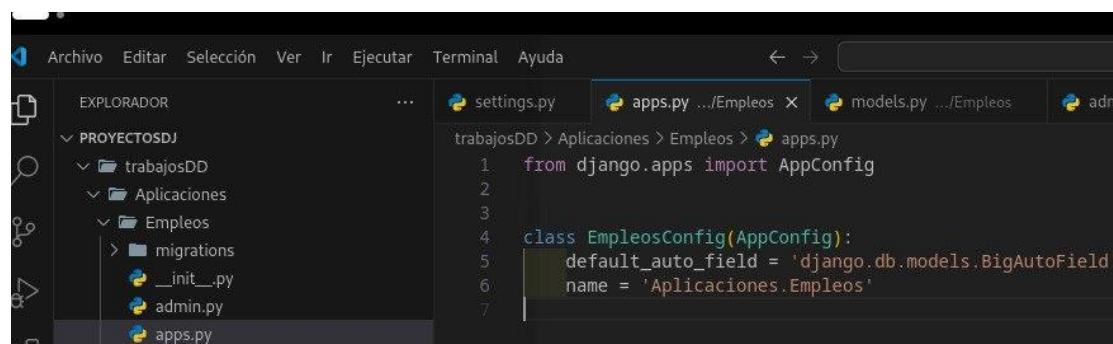
💡 Pero como nuestras apps están dentro de la carpeta Aplicaciones, hay que escribir el nombre completo con puntos, así:

```
class EmpleosConfig(AppConfig):  
  
    default_auto_field =  
        'django.db.models.BigAutoField'  
  
    name = 'Aplicaciones.Empleos'
```

### 📝 Observación:

El atributo name debe incluir el nombre de la carpeta donde está ubicada la app, separado por puntos.

¡Ya está! Esto ayuda a que Django sepa dónde encontrar esa aplicación en tu carpeta del proyecto.



```
trabajosDD > Aplicaciones > Empleos > apps.py  
1   from django.apps import AppConfig  
2  
3  
4   class EmpleosConfig(AppConfig):  
5       default_auto_field = 'django.db.models.BigAutoField'  
6       name = 'Aplicaciones.Empleos'
```

### 📌 ¿Para qué sirve esto?

Indica a Django dónde está ubicada exactamente la app. Si no se configura bien, puede que no funcione al hacer migraciones o al llamar los modelos.

Haz lo mismo para cualquier aplicación que tengas, en nuestro caso loaremos igual en la otra app llamada Empresas:

```
class EmpresasConfig(AppConfig):
```



```
default_auto_field = 'django.db.models.BigAutoField'  
name = 'Aplicaciones.Empresas'
```

The screenshot shows a code editor with the following structure:

- EXPLORADOR**: Shows the project structure:
  - PROYECTOSDJ
  - trabajosDD
  - Aplicaciones
    - Empleos
      - migrations
      - \_\_init\_\_.py
      - admin.py
      - apps.py
      - models.py
      - tests.py
      - views.py
    - Empresas
      - migrations
      - \_\_init\_\_.py
      - admin.py
      - apps.py
      - models.py
      - tests.py
- TrabajosDD > Aplicaciones > Empresas > apps.py**: The current file being edited.

The code in the editor is:

```
from django.apps import AppConfig  
  
class EmpresasConfig(AppConfig):  
    default_auto_field = 'django.db.models.BigAutoField'  
    name = 'Aplicaciones.Empresas'
```

## B. Crear modelos en models.py

Ubicación del archivo: Aplicaciones/NombreDeLaApp/models.p

Qué es:

Este archivo es donde defines los modelos de tu aplicación. Un modelo es una clase que representa una tabla en la base de datos.

Cómo se genera por defecto:

El archivo models.py está vacío cuando creas la app, listo para que agregues tus clases. Por ejemplo, una clase simple podría verse así:

### Por qué es importante:

Los modelos definen la estructura de los datos que tu aplicación va a manejar. Django usa estos modelos para crear las tablas en la base de datos mediante las migraciones.

Ejemplo simple para Aplicaciones/Empleos/models.py:

En Empleos/models.py:

```
class Cargo(models.Model):  
  
    id = models.AutoField(primary_key=True)  
  
    empleo = models.CharField(max_length=100)  
  
    def __str__(self):  
  
        return f'{self.id} - {self.empleo}'
```

¿Qué significa eso?



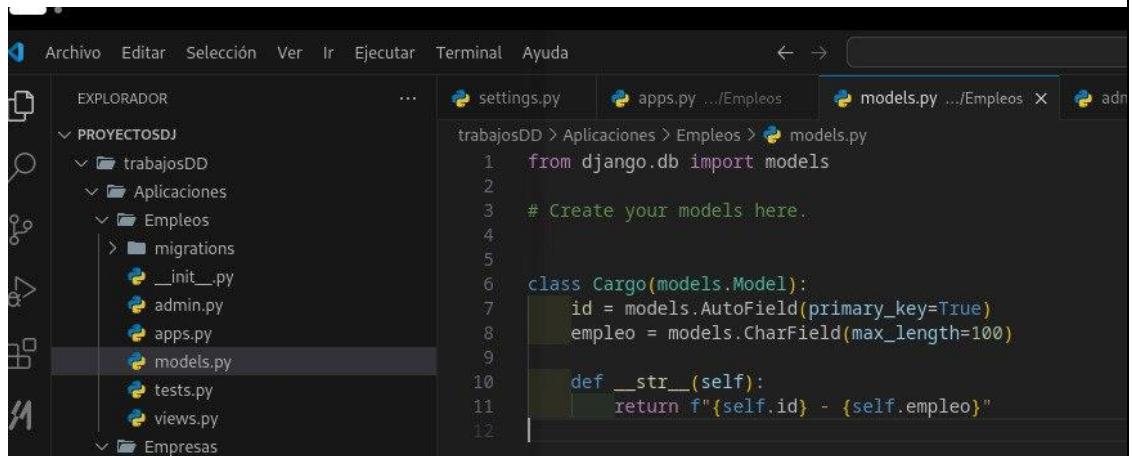
AutoField: el ID se crea solo, como un contador

CharField: texto corto

Define una tabla llamada Cargo con dos campos: un ID que se genera solo, y un nombre del empleo.

`__str__` sirve para que, cuando lo veas en el admin, aparezca de forma legible.

Puedes agregar más campos si deseas. Esto es solo un ejemplo.



The screenshot shows a code editor with the following structure:

- Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
- EXPLORADOR
- PROYECTOSDJ
- trabajosDD
- Aplicaciones
- Empleos
- migrations
- \_\_init\_\_.py
- admin.py
- apps.py
- models.py
- tests.py
- views.py
- Empresas

models.py content:

```
1  from django.db import models
2
3  # Create your models here.
4
5
6  class Cargo(models.Model):
7      id = models.AutoField(primary_key=True)
8      empleo = models.CharField(max_length=100)
9
10     def __str__(self):
11         return f"{self.id} - {self.empleo}"
```

### En Empresas/models.py:

```
class Cargo(models.Model):
    id = models.AutoField(primary_key=True)
    nombres = models.CharField(max_length=100)
    funciones = models.TextField()
    horario = models.CharField(max_length=500)
    requisitos = models.TextField()
    sueldo = models.DecimalField(decimal_places=2, max_digits=10)

    def __str__(self):
        return f"{self.id} - {self.nombres} | Funciones: {self.funciones} | Horario: {self.horario} | Requisitos: {self.requisitos} | Sueldo: ${self.sueldo}"
```

💡 ¿Qué hay aquí?

TextField → para textos largos

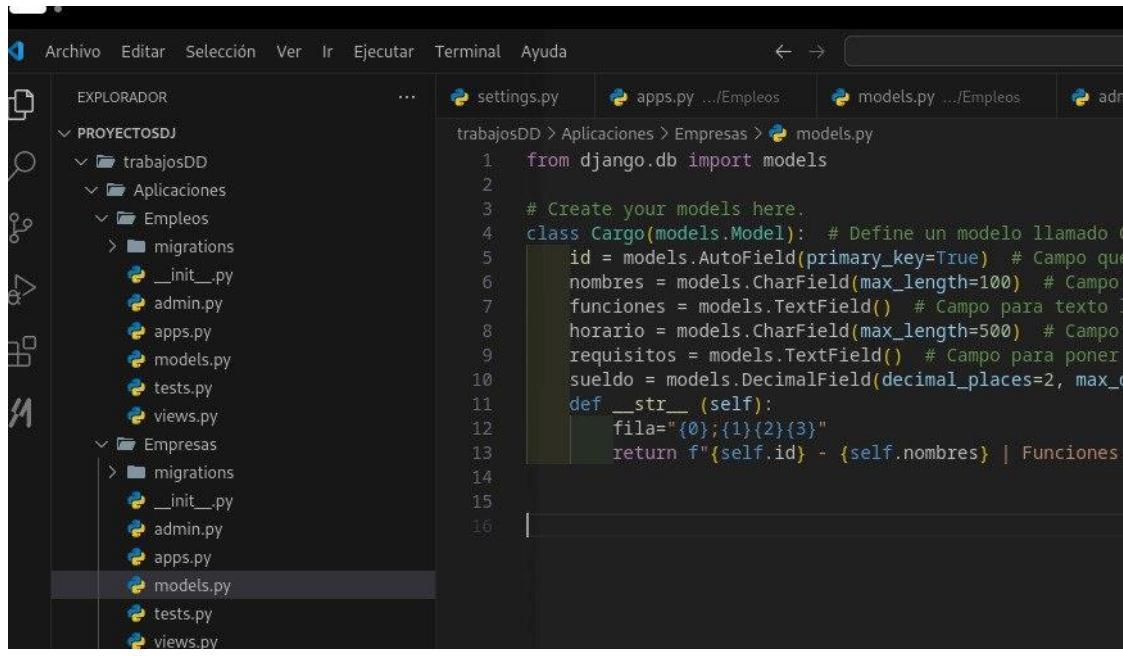
DecimalField → para guardar dinero o precios



max\_digits → cuántos dígitos en total (ej: 999999.99 tiene 8)

decimal\_places → cuántos decimales (ej: 2 para dólares y centavos)

💡 Puedes crear más modelos si quieres. No estás obligado a usar solo



The screenshot shows a code editor with the following details:

- Project Structure:** PROYECTOSDJ > trabajosDD > Aplicaciones > Empresas.
- Current File:** models.py (highlighted in the sidebar).
- Code Content:**

```
1  from django.db import models
2
3  # Create your models here.
4  class Cargo(models.Model): # Define un modelo llamado Cargo
5      id = models.AutoField(primary_key=True) # Campo que contiene el ID de la tabla
6      nombres = models.CharField(max_length=100) # Campo para nombres
7      funciones = models.TextField() # Campo para texto
8      horario = models.CharField(max_length=500) # Campo para horario
9      requisitos = models.TextField() # Campo para poner requisitos
10     sueldo = models.DecimalField(decimal_places=2, max_digits=8)
11
12     def __str__(self):
13         fila=f'{0};{1};{2};{3}'
14         return f'{self.id} - {self.nombres} | Funciones: {self.funciones} | Horario: {self.horario} | Requisitos: {self.requisitos} | Sueldo: {self.sueldo}'
```

### ☑ C. Registrar modelos en admin.py

Este archivo sirve para que Django muestre tus tablas (modelos) en el **panel de administración (el que se abre con /admin)**.

#### 📄 ¿Qué tienes que escribir?

Solo importa registrar tus modelos:

#### 📄 En Empleos/admin.py:

```
from .models import Cargo
```

```
admin.site.register(Cargo)
```

#### 🧠 ¿Qué hace esto?

Le dice a Django: “Este modelo llamado Cargo quiero que aparezca en el panel de administración para que lo pueda ver, agregar o editar.”

💡 Puedes registrar varios modelos si tienes más:



```

trabajosDD > Aplicaciones > Empleos > admin.py
1   from django.contrib import admin
2   from.models import Cargo
3   # Register your models here.
4
5   admin.site.register(Cargo)

```

### En Empresas/admin.py:

```

from .models import Cargo
admin.site.register(Cargo)

```

### ¿Qué significa .models?

.models se refiere al archivo models.py que está en **la misma carpeta**.

Al registrar con admin.site.register(), le decimos a Django: “muéstrame esta tabla en el panel de administración”.

```

trabajosDD > Aplicaciones > Empresas > admin.py
1   from django.contrib import admin
2   from.models import Cargo
3   # Register your models here.
4   admin.site.register([Cargo])
5

```

## 6: Activar los modelos y crear la base de datos

### ¿Qué significa esto?

Hasta ahora solo hemos escrito el diseño de las tablas (modelos), pero todavía no existen en la base de datos. Para que Django cree esas tablas en la base de datos real (por defecto usa SQLite), necesitamos activar los modelos y aplicar las migraciones.

#### A. Navegar a la carpeta del proyecto

Abrimos la terminal y vamos a la carpeta donde está el archivo manage.py. Ese archivo es como el control maestro del proyecto.



ProyectosDJ/trabajosDD

Ls

Debs ver algo asi

Aplicaciones manage.py trabajosDD

## B. Crear migraciones

```
python manage.py makemigrations
```

```
mbue@fedora:~/ProyectosDJ/trabajosDD$ python manage.py makemigrations
Migrations for 'Empleos':
    Aplicaciones/Empleos/migrations/0001_initial.py
        + Create model Cargo
Migrations for 'Empresas':
    Aplicaciones/Empresas/migrations/0001_initial.py
        + Create model Cargo
mbue@fedora:~/ProyectosDJ/trabajosDD$
```

### Qué hace:

Este comando crea archivos llamados migraciones. Estos archivos contienen las instrucciones necesarias para que la base de datos se actualice según los cambios que hiciste en tus modelos (en models.py).

### Por qué es importante:

Aquí no se cambia todavía la base de datos, solo se generan los archivos que indican qué cambios se deben hacer.

### Ejemplo:

Si creaste un modelo nuevo llamado Producto, makemigrations va a crear un archivo que dice “crear tabla Producto con estos campos”.



## C. Aplicar migraciones

```
python manage.py migrate
```

```
NameError: name 'Cargo' is not defined, did you mean: 'Cargo'?
mbuefedor@~/ProyectosDJ/trabajosDD$ python manage.py makemigrations
Migrations for 'Empleos':
    + Create model Cargo
Aplicaciones/Empleos/migrations/0001_initial.py
Migrations for 'Empresas':
    + Create model Cargo
Aplicaciones/Empresas/migrations/0001_initial.py
mbuefedor@~/ProyectosDJ/trabajosDD$ python manage.py migrate
Operations to perform:
  Apply all migrations: Empleos, Empresas, admin, auth, contenttypes, sessions
Running migrations:
  Applying Empleos.0001_initial... OK
  Applying Empresas.0001_initial... OK
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002.Alter_permission_name_max_length... OK
  Applying auth.0003_Alter_user_email_max_length... OK
  Applying auth.0004_Alter_user_username_opts... OK
  Applying auth.0005_Alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_Alter_validators_add_error_messages... OK
  Applying auth.0008_Alter_user_username_max_length... OK
  Applying auth.0009_Alter_user_last_name_max_length... OK
  Applying auth.0010_Alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_Alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
mbuefedor@~/ProyectosDJ/trabajosDD$
```

### Qué hace:

Este comando ejecuta las instrucciones guardadas en los archivos de migración y realmente cambia la base de datos.

### Por qué es importante:

Aquí es cuando se crean, modifican o eliminan las tablas en la base de datos de verdad.

### Ejemplo:

Luego de makemigrations, al correr migrate se crea físicamente la tabla Producto en la base de datos

### ¿Por qué primero makemigrations y luego migrate?

- Primero ejecutas makemigrations, que crea los archivos con las instrucciones sobre qué cambios deben hacerse en la base de datos. Este comando **no cambia nada aún** en la base de datos, solo prepara esos archivos de migración.
- Luego ejecutas migrate, que es el que **realmente aplica esos cambios en la base de datos**. Aquí es donde se crean o modifican las tablas y campos.

### Importante:

Cada vez que haces cualquier cambio en tus modelos (como cambiar el nombre de un campo, agregar uno nuevo o eliminar uno), debes ejecutar makemigrations para que Django detecte esos cambios y cree la migración correspondiente.



### **Nota sobre “No changes detected”:**

A veces, cuando ejecutas makemigrations, puede salir el mensaje “No changes detected”. Esto significa que Django no encontró diferencias entre los modelos actuales y las migraciones previas.

- Puede ser porque no guardaste los cambios en models.py.
- O porque ya hiciste las migraciones para esos cambios antes.  
Esto no es un error, solo indica que no hay nada nuevo que migrar.

### **Recuerda:**

No basta con hacer solo makemigrations. Si no ejecutas después migrate, los cambios **no se aplican en la base de datos**. Por eso es necesario usar siempre ambos comandos en ese orden.

- Si sale error, no te asustes. Django te indica exactamente **la carpeta, archivo y línea** donde está el problema.

Aunque salga mucho texto rojo, puede ser solo una coma o punto mal puesto.

### **D. Crear un superusuário**

```
python manage.py createsuperuser
```

```
mbue@fedora:~/ProyectosDJ/trabajosDD$ python manage.py createsuperuser
Nombre de usuario (leave blank to use 'mbue'): root
Dirección de correo electrónico: a@gmail.com
Password:
Password (again):
La contraseña es demasiado similar a la de nombre de usuario.
This password is too short. It must contain at least 8 characters.
Esta contraseña es demasiado común.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
mbue@fedora:~/ProyectosDJ/trabajosDD$
```

Este es un usuario especial que te permitirá entrar al panel de administración para ver y manejar los datos de todas tus tablas.

Responde a las preguntas como este ejemplo:

Nombre de usuario (leave blank to use 'mbue'): root

Dirección de correo electrónico: a@gmail.com

Password: \*\*\*\*\*

Password (again): \*\*\*\*\*

Bypass password validation and create user anyway? [y/N]: y

Superuser created successfully.



## 📌 ¿Para qué sirve esto?

Con este usuario vas a poder entrar al panel de administración en <http://127.0.0.1:8000/admin> y manejar los datos de todas las tablas: crear, modificar, borrar, ver registros.

¡Ya tienes lista la base de datos y el acceso al panel administrativo!

## 7. Ejecutar el servidor

```
mbue@fedora:~/ProyectosDJ/trabajosDD$ python manage.py createsuperuser
Nombre de usuario (leave blank to use 'mbue'): 
Dirección de correo electrónico: a@gmail.com
Password:
Password (again):
La contraseña es demasiado similar a la anterior.
This password is too short. It must contain at least 8 characters.
Esta contraseña es demasiado común.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
mbue@fedora:~/ProyectosDJ/trabajosDD$ python manage.py runserver
Watching for file changes with StatReloader...
Performing system checks...

System check identified no issues (0 silenced).
May 10, 2025 - 16:18:45
Django version 5.2, using settings 'trabajosDD.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

Abrir el enlace
Copiar Mayús+Ctrl+C
Copiar enlace
Pegar Mayús+Ctrl+V
Seleccionar todo Mayús+Ctrl+A
Seleccionar ninguno
Sólo lectura
Reiniciar
Reiniciar y limpiar
Inspecionar terminal
```

Para ver tu proyecto funcionando, ejecuta este comando:

```
python manage.py runserver
```

Verás algo como:

```
Starting development server at http://127.0.0.1:8000/
```

Eso significa que el proyecto está corriendo. Puedes abrir el navegador y copiar esa dirección

Haz clic derecho o doble clic (con dos dedos en touchpad) sobre esa dirección para abrirla en el navegador.



## Ver el panel de Django y entrar al admin

1. Abre el navegador y entra a:

<http://127.0.0.1:8000/> → Verás la página de bienvenida de Django.

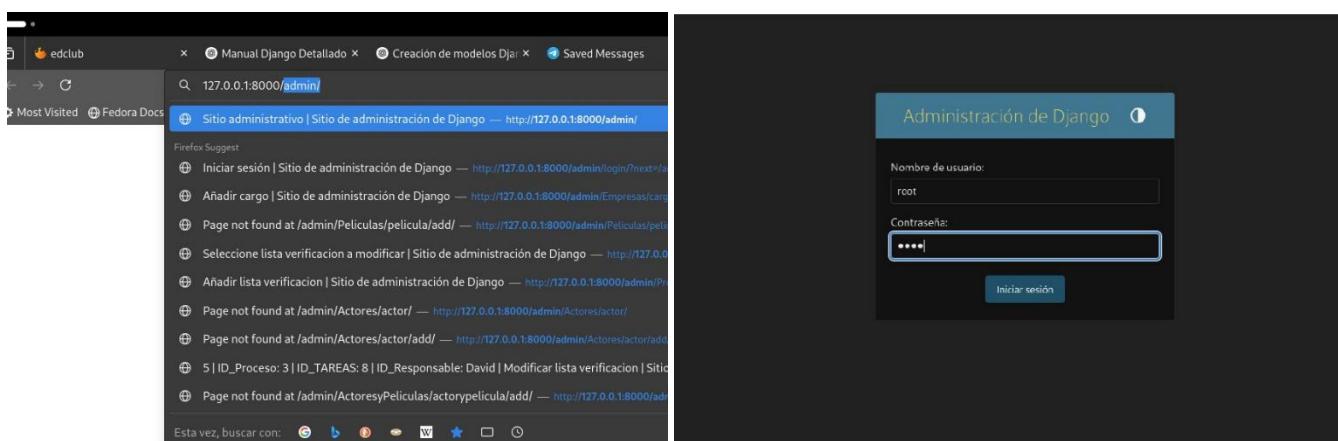


[Documentación de Django](#) [Tutorial: Una aplicación de encuesta](#) [Comunidad Django](#)

2. Luego escribe:

<http://127.0.0.1:8000/admin>

Inicia sesión con el usuario y contraseña del superusuario que creaste.



## 🔥 ¿Qué verás?

Un panel donde puedes ver y administrar todas las tablas que registraste. ¡Un **CRUD** completo listo sin escribir una sola línea de HTML!

## 📌 Django ya tiene su propio servidor y base de datos integrada.

Esto hace que crear, ver, actualizar y borrar datos sea muy fácil para pruebas y desarrollo.

## 🎉 Ya puedes hacer un CRUD completo desde aquí:

- Crear datos nuevos
- Ver registros existentes
- Editar registros
- Eliminar registros



## 8. Conocimientos necesarios antes de empezar

### ¿Cómo funciona la estructura MVT (Modelo-Vista-Template) en Django?

Para entender cómo mostramos los datos (como los cargos) en una página web, es importante conocer la estructura que usa Django llamada **MVT**, que significa:

- **Modelo (Model):** Es donde defines cómo se guardan los datos en la base de datos. En nuestro caso, el modelo `Cargo` define la tabla y sus columnas.
- **Vista (View):** Es la parte que recibe la solicitud (cuando alguien visita una página web), consulta los datos que necesitas (por ejemplo, todos los cargos) y decide qué mostrar.
- **Template (Plantilla):** Es el archivo HTML que define cómo se verá la página web. Aquí colocamos el diseño, textos, imágenes, y también mostramos los datos que la vista envía.

### Cómo fluye la información en Django con MVT

#### 1. El usuario pide una página web

Cuando trabajas con Django, normalmente ejecutas el servidor de desarrollo local con este comando:

```
python manage.py runserver
```

Al hacer eso, Django arranca un servidor web en tu computadora. La dirección principal (raíz) de ese servidor es:

```
http://localhost:8000/
```

- localhost** significa que el servidor está corriendo en tu propia máquina.
- 8000** es el número de puerto que Django usa por defecto.

#### ¿Y cómo se accede a las páginas específicas?

Para acceder a otras páginas de tu aplicación, **se agregan rutas después del /**.

Por ejemplo:

- `http://localhost:8000/` → Página principal o raíz
- `http://localhost:8000/nuevoCargo` → Página para agregar un nuevo cargo
- `http://localhost:8000/listarCargos` → Página para ver todos los cargos

👉 En Django, solo necesitas definir el **nombre de la ruta** (lo que viene después del `/`) en el archivo `urls.py`, y Django ya sabe que debe agregarlo al final de la raíz del servidor.



## 📁 ¿Dónde se define eso en el proyecto?

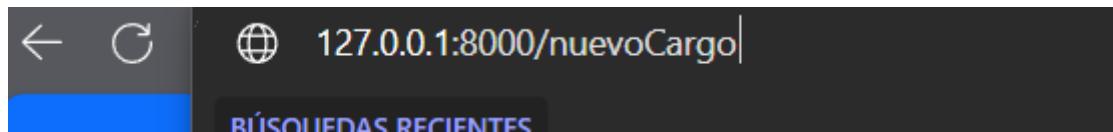
Estas se crean en el archivo urls.py, se escribe algo como:

```
path('nuevoCargo', views.nuevoCargo),
```

Aquí no escribes `http://localhost:8000/nuevoCargo`, porque Django ya sabe que la dirección completa empieza en `localhost:8000`.

Tú solo defines **la parte de la ruta**, que sería:

```
nginx  
CopiarEditar  
nuevoCargo
```



## 2. Django recibe esa solicitud y revisa su archivo urls.py

Django no muestra páginas por sí solo. Primero necesita saber qué hacer con esa dirección que llegó.

Entonces va al archivo Empresas/urls.py y busca si hay alguna ruta que coincida con lo que el usuario escribió.

The screenshot shows a code editor with two tabs open: "views.py" and "urls.py". The "urls.py" tab is active and displays the following code:

```
#Configuración de rutas específicas de la App Empresas
from django.urls import path
from . import views
urlpatterns=[
    path('', views.inicio),
    path('nuevoCargo', views.nuevoCargo),
    path('guardarCargo', views.guardarCargo),
    path('ECargo/<id>', views.eliminarCargo, name='ECargo'),
    path('EditarCargo/<id>', views.Mformualario),
    path('PECargo', views.ProcesarEdicionCargo)
]
```

The line `path('nuevoCargo', views.nuevoCargo),` is highlighted with a yellow box. The "views.py" tab is also visible in the background.

Como encontró una coincidencia con `nuevoCargo`, Django decide ejecutar la función llamada `nuevoCargo` que está dentro del archivo `views.py`.



### 3. Django va al archivo views.py y ejecuta la función

Dentro de views.py, está esta función:

The screenshot shows the PyCharm IDE interface. On the left, the 'EXPLORADOR' (File Explorer) shows the project structure with files like 'views.py', 'urls.py', 'Aplicaciones', 'Empleos', 'Empresas', '\_\_pycache\_\_', 'migrations', and 'templates'. Inside 'templates', there are files: 'Editar.html', 'inicio.html', 'nuevoCargo.html', 'plantilla.html', and '\_\_init\_\_.py'. The 'views.py' file is open in the main editor area. It contains Python code for a Django application named 'Empresas'. The code includes imports from 'django.contrib.messages' and defines two functions: 'inicio' which returns a rendered 'inicio.html' template with a list of all cargos, and 'nuevoCargo' which returns a rendered 'nuevoCargo.html' template. A red box highlights the 'nuevoCargo' function.

```
from django.contrib import messages
# Create your views here.
# Esta función para renderizar el listado de cargos
def inicio(request):
    listadoCargos=Cargo.objects.all()
    return render(request,"inicio.html",
{'cargo':listadoCargos})
def nuevoCargo(request):
    return render(request,"nuevoCargo.html")
```

Lo que hace esta función es preparar y enviar(**con el render**) la plantilla **HTML** que contiene el formulario para agregar un nuevo cargo.

Django busca el archivo **nuevoCargo.html** dentro de la carpeta de **Templates**, lo carga y se lo envía al navegador del usuario para que lo muestre.

### 4. El usuario llena el formulario y presiona "Guardar"

The screenshot shows a web browser window with the URL '127.0.0.1:8000/nuevoCargo'. The page title is 'Nuevo Cargo'. It contains a form with the following fields:

- Nombre: Programador
- Requisitos: conocimientos basicos
- Funciones: Programar
- Horario: 12
- Sueldo: 122

At the bottom of the form is a blue button labeled 'Guardar' and a red 'Cancelar' button.

Cuando el usuario escribe los datos y presiona el botón para guardar, el formulario está configurado para enviar esos datos a una ruta específica:



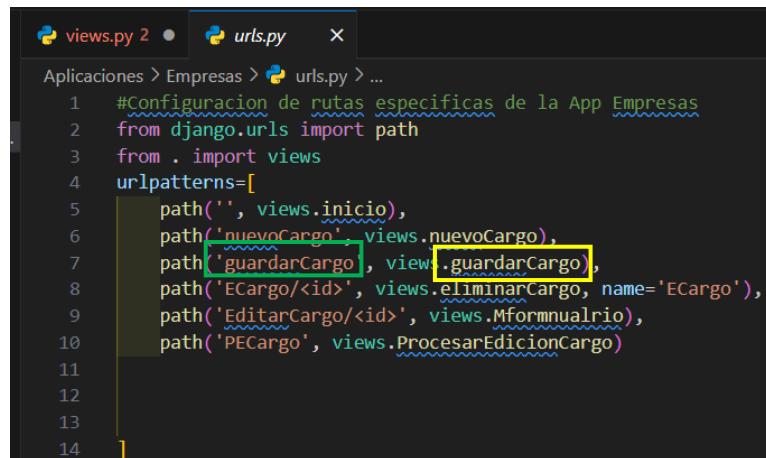
<http://localhost:8000/guardarCargo>

Esto genera **una nueva solicitud**, pero esta vez es de tipo **POST**, porque se están enviando datos al servidor.

## 5. Django otra vez revisa su archivo urls.py

Ahora la dirección es **guardarCargo**, entonces va al mismo archivo urls.py y busca si esa ruta existe.

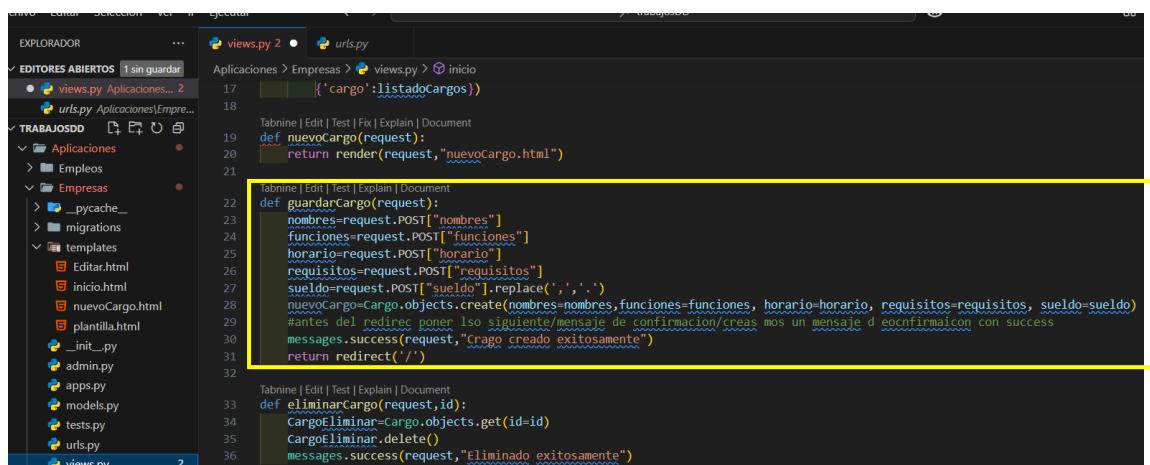
Encuentra esto:



```
#Configuración de rutas específicas de la App Empresas
from django.urls import path
from . import views
urlpatterns=[
    path('', views.inicio),
    path('nuevoCargo', views.nuevoCargo),
    path('guardarCargo', views.guardarCargo),
    path('ECargo/<id>', views.eliminarCargo, name='ECargo'),
    path('EditarCargo/<id>', views.Mformualario),
    path('PECargo', views.ProcesarEdicionCargo)
]
```

Como existe, Django ejecuta la función **guardarCargo()** que está en **views.py**.

## 6. Django ejecuta la función guardarCargo() para guardar los datos



```
def nuevoCargo(request):
    nombres=request.POST["nombres"]
    funciones=request.POST["funciones"]
    horario=request.POST["horario"]
    requisitos=request.POST["requisitos"]
    sueldo=request.POST["sueldo"].replace(',','.')
    nuevoCargo=Cargo.objects.create(nombres=nombres,funciones=funciones,horario=horario,requisitos=requisitos,sueldo=sueldo)
    messages.success(request,"Cargo creado exitosamente")
    return redirect('/')
```

Lo que hace esta función:

1. **Recoge todos los datos del formulario** que vinieron en la solicitud POST (`request.POST[...]`) cada campo donde se llenó los datos como nombre y eso.
2. **Crea un nuevo registro en la base de datos**, usando el modelo Cargo.



3. **Prepara un mensaje de éxito**, para que se muestre al usuario en la próxima pantalla.
4. **Redirecciona automáticamente al usuario a la página principal (/)**, que muestra todos los cargos guardados.

## 7. Django ahora recibe la ruta / y repite el proceso

Como el usuario fue redireccionado a la página principal, ahora el navegador solicita esta dirección:

The screenshot shows a code editor with two tabs open: `urls.py` and `views.py`. The `urls.py` file contains URL patterns for various views, including `views.inicio`. The `views.py` file contains the implementation of the `inicio` view, which retrieves all cargo objects from the database and renders them into the `inicio.html` template. The sidebar on the left shows the project structure, including the `Aplicaciones` and `Empresas` apps, their templates, and other files like `admin.py` and `models.py`.

```

EXPLORADOR
...
EDITORES ABIERTOS 1 sin guardar
• views.py Aplicaciones... 2
× urls.py Aplicaciones\Empres...
TRABAJOS DDD
Aplicaciones
> Empleos
Empresas
> __pycache__
> migrations
templates
    Editar.html
    inicio.html
    nuevoCargo.html
    plantilla.html
__init__.py
admin.py
apps.py
models.py
tests.py
urls.py
views.py 2
urls.py
Aplicaciones > Empresas > urls.py > ...
#Configuración de rutas específicas de la App Empresas
from django.urls import path
from . import views
urlpatterns=[
    path('', views.inicio),
    path('nuevoCargo', views.nuevoCargo),
    path('guardarCargo', views.guardarCargo),
    path('ECargo/<id>', views.eliminarCargo, name='ECargo'),
    path('EditarCargo/<id>', views.Mformulario),
    path('PECargo', views.ProcesarEdicionCargo)
]

```

Django va al archivo `urls.py`, busca la ruta vacía ("") y encuentra:

`path("", views.inicio)`

Entonces va a `views.py` y ejecuta esta función:

The screenshot shows the `views.py` file with the `inicio` function highlighted. This function retrieves all cargo objects from the database and renders them into the `inicio.html` template. Other functions shown include `nuevoCargo` and `guardarCargo`.

```

templates
    Editar.html
    inicio.html
    nuevoCargo.html
    plantilla.html
__init__.py
admin.py
apps.py
models.py
tests.py
urls.py
views.py 2
# Esta función para renderizar el listado de cargos
def inicio(request):
    listadoCargos=Cargo.objects.all()
    return render(request,"inicio.html",
    {'cargo':listadoCargos})
def nuevoCargo(request):
    return render(request,"nuevoCargo.html")
def guardarCargo(request):
    nombres=request.POST["nombres"]
    funciones=request.POST["funciones"]

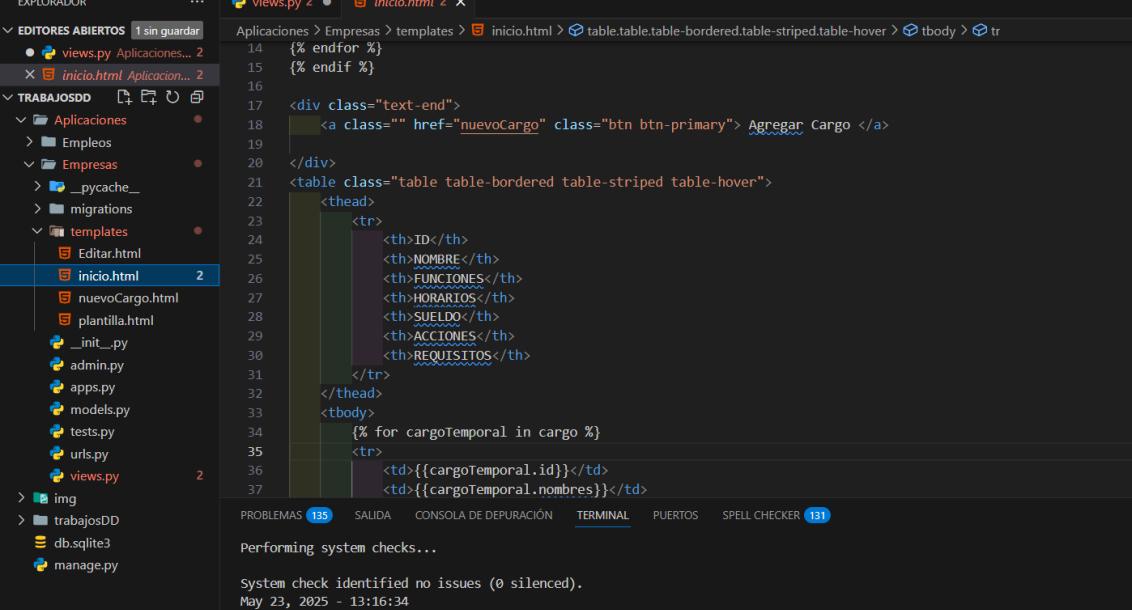
```

Esta función **consulta todos los cargos guardados en la base de datos**, y se los manda a una plantilla HTML llamada `inicio.html`.



## 8. La plantilla HTML muestra los datos en la página web

En el archivo inicio.html, hay código que recorre la lista de cargos recibida y los muestra uno por uno. Por ejemplo:



The screenshot shows a code editor interface with two tabs open: 'views.py' and 'inicio.html'. The 'views.py' tab contains standard Python code for a Django application. The 'inicio.html' tab displays the following HTML code:

```
<div class="text-end">
    <a class="" href="#">nuevoCargo</a>
</div>


| ID | NOMBRE      | FUNCIONES | HORARIOS | SUELDO                | ACCIONES | REQUISITOS                                                             |
|----|-------------|-----------|----------|-----------------------|----------|------------------------------------------------------------------------|
| 12 | Programador | Programar | 12       | conocimientos basicos | 122.00   | <span style="color: blue;"> </span> <span style="color: red;"> </span> |


```

The code uses Jinja2 templating syntax to iterate over a list of 'cargoTemporal' objects and render them into the table rows.

## 9. El usuario ve la página con la información actualizada

Finalmente, el navegador muestra todo en pantalla y el usuario puede ver el cargo que acaba de crear, junto con los anteriores.

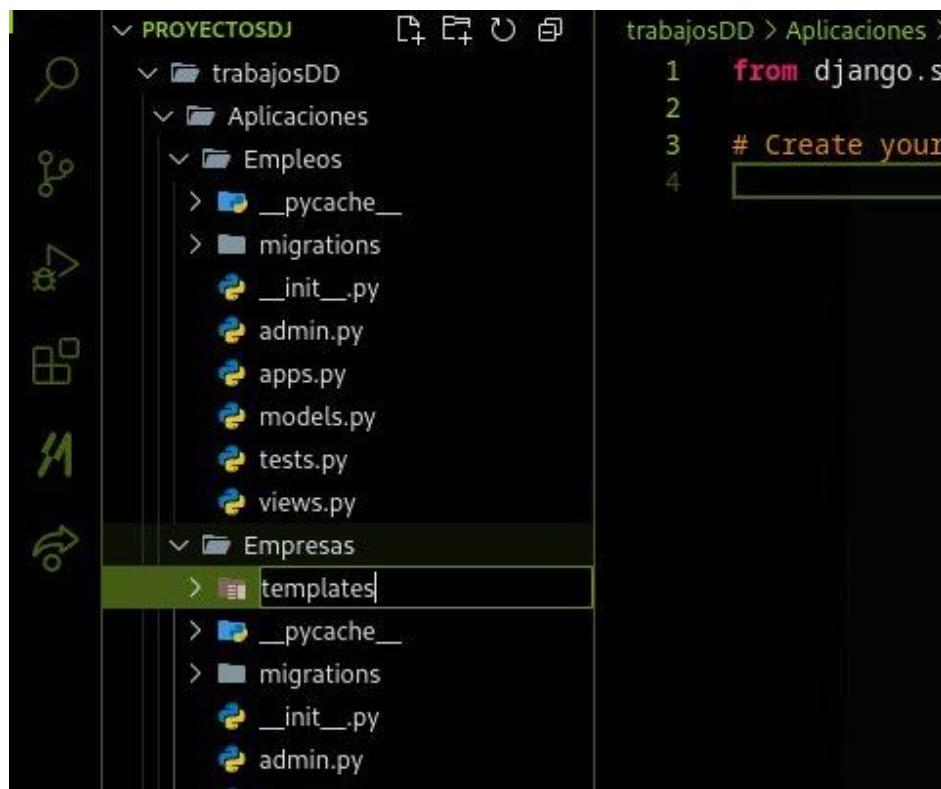


## 9. Crear Crud con HTML

Hasta ahora ya tenemos el modelo Cargo creado y registrado en el panel de administración de Django.

Pero no todos van a entrar al admin. Por eso, ahora vamos a crear una página web normal (con HTML) donde se puedan ver todos los cargos guardados en la base de datos.

### 📁 Crear la carpeta templates en la app Empresas



Para mostrar las páginas web que verán los usuarios, debemos crear los archivos HTML llamados plantillas o templates.

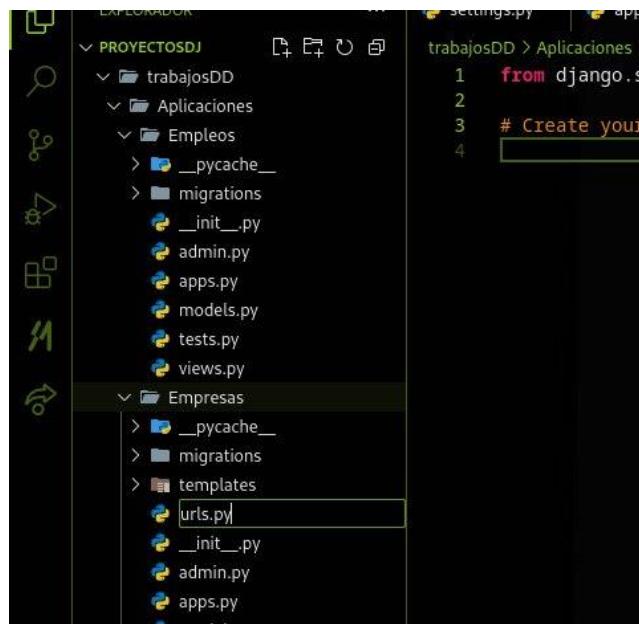
Para que Django pueda encontrarlos fácilmente, dentro de la carpeta de la app Empresas (la que está dentro de la carpeta Aplicaciones), debes crear una carpeta nueva llamada templates.

En esta carpeta templates guardaremos todos los archivos HTML que usará esta aplicación para mostrar las páginas. Django busca dentro de esta carpeta los archivos que las vistas van a enviar para mostrar la información al usuario.

Así, las vistas (que están en views.py) pueden “renderizar” estas plantillas y mostrar la información de la base de datos en formato web.



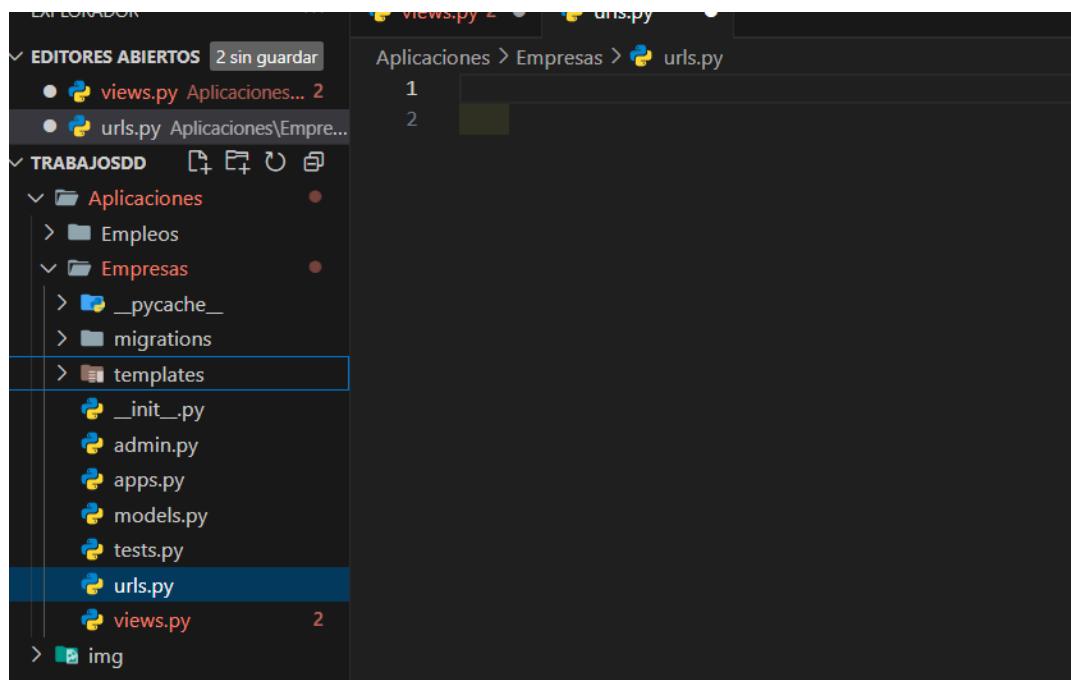
## 🌐 Crear el archivo urls.py en la app Empresas



Dentro de la misma carpeta de la app Empresas, crea un archivo nuevo llamado **urls.py**.

Este archivo define las rutas internas de la app. Son las "direcciones" que el usuario puede escribir en el navegador.

Cada ruta se conecta con una función, que se encuentra en el archivo (views.py), que decide qué mostrar.



## ¿Por qué NO es tan bueno solo poner esto en urls.py?

```
path('cargo/', views.cargo)
```

y sí es mejor poner esto:

```
path('cargo/', views.cargo, name="cargo")
```

---

### 1. ¿Qué hace cada cosa?

- `path('cargo/', views.cargo)`

Le dices a Django:

**"Cuando alguien pida la URL que termina en cargo/, ejecuta la función cargo del archivo views.py."**

Esto funciona, pero la ruta **no tiene un nombre**.

- `path('cargo/', views.cargo, name="cargo")`

Hace exactamente lo mismo que arriba, pero además le das un **nombre a la ruta**, que es "cargo".

Ese nombre funciona como una **etiqueta para llamar a esa ruta desde cualquier parte del código**.

---

### 2. ¿Para qué sirve ponerle un nombre a la ruta?

Cuando usas `name="cargo"`, puedes en tus archivos HTML o en el código hacer esto:

```
<a href="{% url 'cargo' %}>Ir a cargos</a>
```

Esto es bueno porque:

- No necesitas escribir la URL exacta (como `/cargo/`),
  - Si cambias la URL en urls.py (por ejemplo, de `'cargo/'` a `'cargos/'`), solo cambias ahí, y no en todas las plantillas,
  - Evitas errores de escribir mal la dirección,
  - Haces que tu código sea más fácil de mantener y entender.
- 

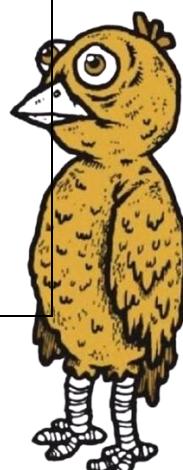
### 3. ¿Qué pasa si no usas name?

Si no usas el nombre, en tus archivos HTML debes escribir la URL exacta así:

```
<a href="/cargo/">Ir a cargos</a>
```

Esto es un problema porque:

- Si cambias la ruta en urls.py, tienes que buscar y cambiar en todos los archivos donde escribiste `/cargo/`,
  - Es fácil equivocarte al escribir la URL manualmente,
  - El código queda menos ordenado y más difícil de mantener.
- 



#### 4. ¿Cuál es la forma recomendada?

Siempre pon nombre a tus rutas en urls.py así:

```
path('cargo/', views.cargo, name='cargo')
```

Y cuando necesites el link en tus archivos HTML o código, usa el nombre así:

```
<a href="{% url 'cargo' %}>Ver Cargos</a>
```

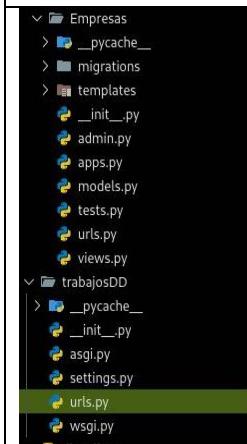
## Conectar las rutas de la app con el proyecto principal

🔗 Ahora debes enlazar este archivo al urls.py general del proyecto

Después de crear el archivo urls.py dentro de tu app Empresas, debes decirle a Django que también use esas rutas así bien las incluya en su búsqueda en las .

Para eso, abre el archivo urls.py que está en la carpeta principal del proyecto (donde está también settings.py, asgi.py, etc.) y modifica el contenido para incluir la app.

Django genera un archivo urls.py con el siguiente contenido:



```
14     1. Import the include() function: from d
15     2. Add a URL to urlpatterns: path('blog/
16     """
17     from django.contrib import admin
18     from django.urls import path
19
20     urlpatterns = [
21         path('admin/', admin.site.urls),
22     ]
23
24 
```



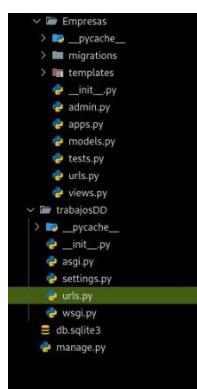
¿Qué significa?

- Solo tiene la ruta para /admin/, que muestra el panel de administración.
- No hay ninguna conexión a otras apps creadas por el usuario.
- No se usa include todavía, porque no hay apps ni rutas personalizadas.

Cuando creamos una app en nuestro caso se llamada Empresas y queremos que sus rutas funcionen, debemos modificar el urls.py del proyecto principal así:



```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('Aplicaciones.Empresas.urls')),
]
```



```
14 1. Import the include() function: from django.urls import inclu
15 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
16 """
17 from django.contrib import admin
18 from django.urls import path,include
19
20 urlpatterns = [
21     path('admin/', admin.site.urls),
22     path ('',include('Aplicaciones.Empresas.urls'))
23 ]
24
25 ]
```

📌 ¿Qué cambia aquí y por qué?

1. Cambios: Se agrega include en la importación:

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

Esto es obligatorio si queremos redirigir las rutas hacia otro archivo que no sea este.

Se agrega una nueva línea en urlpatterns:

```
path("", include('Aplicaciones.Empresas.urls')),
```

Con esto, le decimos a Django:

“si alguien entra a la URL principal (<http://localhost:8000/>), entonces busca las rutas en el archivo Aplicaciones/Empresas/urls.py”.

Recuerda este es un ejemplo y debe poner los nombres de tu proyecto.

En Django, cuando el usuario escribe una dirección en el navegador, como <http://localhost:8000/nuevoCargo>, lo primero que hace Django es revisar el archivo urls.py principal del proyecto (el que está junto al archivo settings.py). Pero si en ese archivo encuentra una línea como path("", include('Empresas.urls')), eso significa que no va a buscar las rutas directamente ahí, sino que se va a la app llamada Empresas, y revisa su propio archivo urls.py para ver si ahí está la ruta escrita. El include sirve justamente para eso: para decirle a Django que busque las rutas dentro de otra app, y no todas en un solo lugar. Así el proyecto queda más ordenado, y cada aplicación maneja sus propias direcciones.

## ¡Evita confusiones! El mejor orden para trabajar en Django

Cuando estás programando en Django, especialmente al hacer un CRUD (Crear, Leer, Actualizar y Eliminar datos), necesitas trabajar con tres archivos principales en tu aplicación:



- ◆ views.py → donde va la lógica (qué hace cada acción del sistema)
  - ◆ urls.py → donde defines las rutas (cómo acceder a cada acción)
  - ◆ .html → donde diseñas la interfaz (lo que ve y usa el usuario)
- 📌 ¿Qué pasa si no sigues un orden?

Puedes confundirte fácilmente, olvidar conectar algo, o tener errores que cuestan tiempo encontrar.

Por eso, este manual te recomienda un orden claro para que tú, como programador principiante, puedas avanzar sin perderte, claro tu decides tu propio orden siempre piensa lo mejor para ti.

## 🔗 ORDEN RECOMENDADO: FUNCIONA COMO UNA CADENA

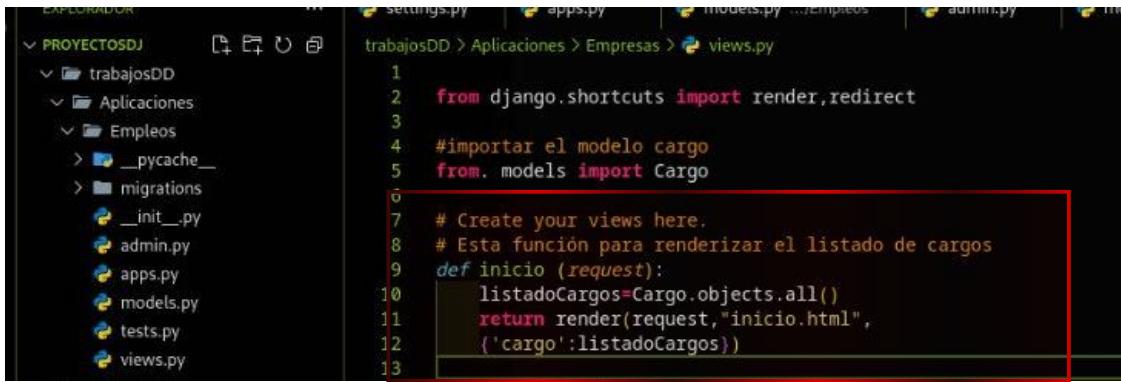
<i>Orden</i>	<i>Archivo</i>	<i>¿Qué se hace aquí?</i>	<i>¿Por qué se hace primero?</i>
1	views.py	Aquí creamos la <b>función</b> que hace todo el trabajo. Es como el "motor" del carro. Esta función decide qué hacer: si debe mostrar una página, consultar algo de la base de datos o guardar datos nuevos.	Si no hay función, no hay nada que mostrar. Por eso siempre se empieza aquí.
2	urls.py	Aquí escribimos la <b>ruta</b> o dirección web. Le decimos a Django: "cuando un usuario entre a /nuevoCargo, usa la función nuevoCargo() que hicimos en views.py."	Una vez que tenemos el motor, ahora le damos un camino para que los usuarios lleguen a él.
3	templates/archivo.html	Aquí diseñamos la <b>página web</b> que verá el usuario: un formulario, una tabla, una alerta... todo va aquí. Puedes usar variables que mandaste desde la función en views.py.	Como ya sabemos qué datos manda la función, ya podemos usar esos datos y mostrarlos bien en HTML.



## 10 Crear el Consulta con la función inicio() en views.py

Ahora vamos a crear la función que se encargará de mostrar la página principal del sistema, donde se verá la lista de cargos registrados en la base de datos.

Ubicación: Aplicaciones/Empresas/views.py



```
from django.shortcuts import render, redirect
from .models import Cargo
from django.contrib import messages

def inicio(request):
    listadoCargos = Cargo.objects.all()
    return render(request, "inicio.html", {'cargo': listadoCargos})
```

```
from django.shortcuts import render, redirect
from .models import Cargo
from django.contrib import messages

def inicio(request):
    listadoCargos = Cargo.objects.all()
    return render(request, "inicio.html", {'cargo': listadoCargos})
```

Cuando escribimos código en Python, a veces necesitamos usar funciones, clases o variables que ya existen en otros archivos o librerías. Para poder usarlas, tenemos que importarlas.

La forma más común y clara de importar algo es usando la instrucción:

```
from nombre_ubicacion import nombre_cosa
```

Esta línea le dice a Python que traiga solo la parte que necesitamos (**la nombre\_cosa**) desde un lugar específico (**nombre\_ubicacion**), para poder usarla directamente en nuestro código.

Cuando vayas a trabajar con una tabla, **siempre debes importar su modelo** desde el archivo models.py, usando esta estructura:

Ejemplo :

```
from .models import Cargo
```



<pre>1. from django.shortcuts import render, redirect</pre> <p>Este archivo sí trae por defecto la línea:</p> <pre>from django.shortcuts import render</pre> <p>Pero tú debes agregar redirect para poder usar esa función más adelante (por ejemplo, para redirigir al usuario después de guardar, eliminar, etc).</p> <p>¿Qué hace cada una?</p> <ul style="list-style-type: none"> <li>render: Muestra un HTML en pantalla.</li> <li>redirect: Redirige al usuario a otra página después de una acción.</li> </ul>	<pre>2. from .models import Cargo</pre> <p>💡 Esta línea importa el modelo Cargo desde el archivo models.py que está en la misma app.</p> <ul style="list-style-type: none"> <li>• El punto (.) significa “desde esta misma carpeta”, es decir: la app Empresas.</li> <li>• “models” es el archivo donde están nuestras clases que representan las tablas.</li> <li>• “import Cargo” significa que queremos usar la clase Cargo que está en models.py.</li> </ul>
	<pre>from django.contrib import messages</pre> <p>Esta línea también la debes agregar tú y al usaremos más después.</p> <p>Sirve para usar el sistema de mensajes de confirmación de Django (por ejemplo: “Cargo guardado con éxito”).</p> <p>Estos mensajes se pueden mostrar en el HTML si configuras la plantilla para recibirlas.</p>
<pre>3. def inicio(request):</pre> <p>💡 Estamos creando una función en Python llamada inicio().</p> <p>Esta función será la encargada de mostrar la página principal con los cargos listados.</p> <p>🧠 ¿Qué es request?</p> <p>Es un parámetro obligatorio en las vistas de Django.</p> <p>Significa "la solicitud" que hace el usuario al abrir una página.</p> <p>Cada vez que un usuario abre una URL, Django crea un objeto llamado request que contiene:</p> <ul style="list-style-type: none"> <li>• Si la solicitud fue GET o POST.</li> <li>• Información del usuario.</li> <li>• Las cookies.</li> <li>• Datos enviados por formularios.</li> <li>• Etc.</li> </ul> <p>💡 Esto es importante: Toda vista en Django debe recibir request como primer parámetro.</p>	<pre>4. listadoCargos = Cargo.objects.all()</pre> <p>💡 Esto es una consulta a la base de datos (query).</p> <ul style="list-style-type: none"> <li>• Cargo.objects.all() busca todos los registros de la tabla Cargo.</li> <li>• objects es un administrador de Django que nos permite consultar, insertar, eliminar, etc.</li> <li>• all() significa “traer todos”.</li> </ul> <p>🧠 Entonces: listadoCargos es una es una variable donde guardamos esa lista de resultados.</p> <p>Esta variable la vamos a enviar al HTML para mostrarla.</p> <p>🧠 En palabras simples: aquí estás trayendo todos los cargos de la base.</p>
<pre>5. return render(request, "inicio.html", {'cargo': listadoCargos})</pre>	



Esta línea es la que se encarga de mostrarle una página web al usuario. Pero no solo eso, también le manda datos desde el código Python hasta el HTML, para que esa página se construya con la información de la base de datos.

Veamos cada parte:

- ◆ return

Esta palabra significa “devuelve” o “regresa”.

En este caso, le está diciendo a Django: “devuélvole al usuario una página web como respuesta”.

- 
- ◆ render(...)

Es una función de Django que sirve para mostrar una página HTML cuando el usuario entra a una URL.

Lo que hace es “juntar” el diseño HTML con los datos que tú le mandes desde Python.

- 
- ◆ Primer parámetro: request

Este siempre debe ir como primer parámetro.

Es la solicitud que hace el navegador del usuario (por ejemplo, cuando alguien entra a una dirección web).

- 
- ◆ Segundo parámetro: "inicio.html"

Es el nombre del archivo HTML que queremos mostrar.

Este archivo debe estar dentro de la carpeta templates.

En este caso, el archivo se llama inicio.html, y es donde vamos a escribir el diseño de la página para mostrar los cargos.

- 
- ◆ Tercer parámetro: {'cargo': listadoCargos}

Esto es un diccionario de Python.

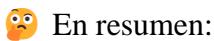
Un diccionario funciona como una “caja” que guarda algo con un nombre, como:

 'cargo' →  listadoCargos

Así, en el HTML vamos a poder usar la palabra cargo para acceder a los datos que están guardados en listadoCargos.

'cargo' → es el nombre corto o etiqueta que vas a usar en el HTML.

listadoCargos → es la lista real de cargos que sacaste de la base de datos con .objects.all().



Esta línea hace lo siguiente:

“Muestra la página inicio.html al usuario, y además mándale la lista de cargos que obtuviste de la base de datos, usando el nombre 'cargo' para que se pueda usar dentro del HTML.”



# Crear el archivo inicio.html para mostrar los datos

## 💡 Crear la plantilla base: plantilla.html

Antes de crear el archivo inicio.html, vamos a crear una **plantilla principal** que se usará como base en todas las páginas del sitio.

### 🧠 ¿Qué es una plantilla base en Django?

Es un archivo .html que contiene la **estructura común de todas las páginas**: el encabezado, los estilos, el menú, el pie de página, etc. Así evitamos repetir lo mismo en cada archivo HTML.

Django nos permite usar una plantilla base gracias a su sistema de **herencia de plantillas**.

🔧 En Django usamos **{% block %}** y **{% endblock %}** para marcar esa zona de contenido variable.

## 📁 Crear el archivo plantilla.html

Ubícalo en la carpeta templates de tu app.

### 📁 Estructura esperada:

Empresas/

```
|   └── templates/  
|       |   └── plantilla.html  
|       |   └── (otros HTML)
```

## 📋 Código completo de plantilla.html

Copia y pega este contenido exactamente así en el archivo:

```
plantilla.html x  
Aplicaciones > Empresas > templates > plantilla.html > script  
1  <!-- Encabezado -->  
2  <h1>Contenido del encabezado aquí</h1>  
3  
4  <!-- Contenido principal -->  
5  <div class="container-fluid">  
6      {% block contenido %}  
7      {% endblock %}  
8  </div>  
9  
10 <!-- Pie de página -->  
11 <h1>Contenido del pie de página aquí</h1>  
12  
13 <!-- Mensajes emergentes con SweetAlert2 -->  
14 {% if messages %}  
15     {% for mensaje in messages %}  
16         <script>  
17             Swal.fire({  
18                 title: "Mensaje",  
19                 text: "{{ mensaje }}",  
20                 icon: "success"  
21             });  
22         </script>  
23     {% endfor %}  
24 {% endif %}  
25
```



```

<!-- Encabezado -->
<h1>Contenido del encabezado aquí</h1>

<!-- Contenido principal -->
<div class="container-fluid">{ % block contenido % } { % endblock % }</div>
<!-- IMportacion link de sweetalert2-->
<script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
<!-- Pie de página -->
<h1>Contenido del pie de página aquí</h1>

<!-- Mensajes emergentes con SweetAlert2 -->
{ % if messages % } { % for mensaje in messages % }
<script>
Swal.fire({
    title: "Mensaje",
    text: "{{ mensaje }}",
    icon: "success",
});
</script>
{ % endfor % } { % endif % }

```

## ¿Qué es plantilla.html y para qué sirve?

plantilla.html es el **archivo base** que todas las demás páginas de tu aplicación van a usar. Con esta plantilla puedes definir una estructura general que se repetirá en cada página: el encabezado, el contenido, el pie de página y los mensajes al usuario.

Todo esto se escribe usando **Jinja2**, el sistema de plantillas que viene con Django. Es un lenguaje que usa llaves y porcentajes para indicar qué partes del HTML van a cambiar.

## Explicación del código

Parte	¿Qué es y para qué sirve?
<!-- Encabezado -->	Puedes poner un título, menú o logo. Se repite en todas las páginas.
{ % block contenido % } / { % endblock % }	Esto es de <b>Jinja2</b> . Marca una zona que <b>cambiará en cada página</b> . Las demás páginas pondrán su contenido aquí.
<!-- Pie de página -->	Puedes poner el autor, derechos reservados o fecha. Es igual en todas las páginas.



## Parte

### ¿Qué es y para qué sirve?

Código de **Jinja2** que **muestra mensajes** (como { % if messages % } ... { % endif % } "cargo guardado") usando **SweetAlert2**. Es útil para avisar al usuario.

```
{ { mensaje } }
```

Esto también es de **Jinja2**. Sirve para **mostrar el contenido de una variable** en HTML.

```
<script  
src="https://cdn.jsdelivr.net/  
npm/sweetalert2@11"></script>
```

Esta línea **carga la librería SweetAlert2 desde internet** (CDN). Es necesario colocarla antes de usar Swal.fire(...). Si no se pone, **los mensajes emergentes no funcionarán**.

### 🧠 ¿Cómo se usa esta plantilla?

Cuando crees una página como listar.html, debes decirle que **use plantilla.html como base**. Para eso usas { % extends 'plantilla.html' % }.

#### Ejemplo:

```
{% extends 'plantilla.html' % }

{% block contenido %}

<h2>Listado de cargos</h2>

<!-- Aquí se muestra el contenido de esta vista --&gt;

{% endblock %}</pre>
```

### 📌 ¿Qué es Jinja2?

- Es el **lenguaje de plantillas** que usa Django.
- Permite **mezclar HTML con variables y bloques de lógica** (como if, for, etc.).
- Se escribe usando { % ... % } para instrucciones y { { ... } } para mostrar variables.
- Todo esto se interpreta en el servidor y se convierte en HTML para el navegador.



Ahor si continuaremos con inicio.html

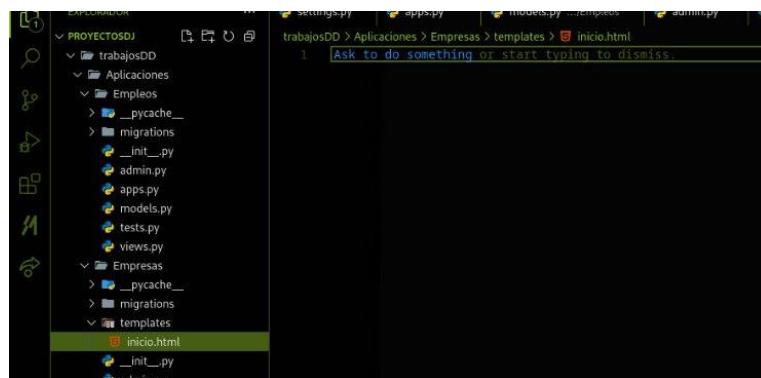
## 📁 UBICACIÓN DEL ARCHIVO HTML

Primero, asegúrate de tener estas carpetas creadas dentro de tu app Empresas:

Empresas/

```
|   └── templates/
|       └── inicio.html ← Aquí va el archivo
|   └── views.py
|   └── models.py
└── ...otros archivos
```

Dentro de Empresas/templates, crea un archivo llamado inicio.html.



Pega este código

```
plantilla.html  inicio.html 2 ×
Aplicaciones > Empresas > templates > inicio.html > table.table.table-bordered.table-striped.table-hover > thead > tr > th
1  {% extends "./plantilla.html" %} 
2  {% block contenido %} 
3
4  <h1>Listado de Cargos</h1>
5  {% if messages %}
6  {% for mensaje in messages %}
7  <script>
8      Swal.fire({
9          title: "¡Éxito!",
10         text: "{{ mensaje }}",
11         icon: "error"
12     });
13  </script>
14  {% endfor %}
15  {% endif %}
16
17  <div class="text-end">
18      <a class="" href="nuevoCargo" class="btn btn-primary"> Agregar Cargo </a>
19
20  </div>
21  <table class="table table-bordered table-striped table-hover">
22      <thead>
23          <tr>
24              <th>ID</th>
25              <th>NOMBRE</th>
26              <th>FUNCIONES</th>
27              <th>HORARIOS</th>
28              <th>SUELDO</th>
29              <th>ACCIONES</th>
30              <th>REQUISITOS</th>
31      </tr>
32  </thead>
33  <tbody>
34      {% for cargoTemporal in cargo %}
35      <tr>
36          <td>{{cargoTemporal.id}}</td>
37          <td>{{cargoTemporal.nombres}}</td>
```



```

{ % extends "./plantilla.html" % }
{ % block contenido % }

<h1>Listado de Cargos</h1>
{ % if messages % }
{ % for mensaje in messages % }
<script>
  Swal.fire({
    title: "¡Éxito!",
    text: "{ { mensaje } }",
    icon: "error"
  });
</script>
{ % endfor % }
{ % endif % }

<div class="text-end">
  <a href="{ % url 'nuevoCargo' % }">Nuevo Cargo</a>
</div>
<table class="table table-bordered table-striped table-hover">
  <thead>
    <tr>
      <th>ID</th>
      <th>NOMBRE</th>
      <th>FUNCIONES</th>
      <th>HORARIOS</th>
      <th>SUELDO</th>
      <th>ACCIONES</th>
      <th>REQUISITOS</th>
    </tr>
  </thead>
  <tbody>
    { % for cargoTemporal in cargo % }
    <tr>
      <td>{ { cargoTemporal.id } }</td>
      <td>{ { cargoTemporal.nombres } }</td>
      <td>{ { cargoTemporal.funciones } }</td>
      <td>{ { cargoTemporal.horario } }</td>
      <td>{ { cargoTemporal.requisitos } }</td>
      <td>{ { cargoTemporal.sueldo } }</td>
      <td>
        <a class="btn btn-warning" href="{ % url 'EditarCargo' cargoTemporal.id % }"><i class="fa fa-pen"></i> Editar</a>

        <a href="#" onclick="confirmarEliminacion('{ % url 'ECargo' cargoTemporal.id % }')">
          <i class="fa fa-trash"></i> Eliminar
        </a>
      </td>
    </tr>
  </tbody>
</table>

```



```

        </a>

        </td>
    </tr>
    { % endfor %}
</tbody>
</table>

<script>
    function confirmarEliminacion(url) {
        Swal.fire({
            title: "¿Estás seguro?",
            text: "¡No podrás revertir esto!",
            icon: "warning",
            showCancelButton: true,
            confirmButtonColor: "#3085d6",
            cancelButtonColor: "#d33",
            confirmButtonText: "Sí, eliminar"
        }).then((result) => {
            // Aquí se redirige después del botón "Confirmar"
            window.location.href = url;
        });
    }
</script>

{ % endblock %

```

 ¿Qué hace esto?

## Estructura base del formulario

```

{ % extends "./plantilla.html" %}
{ % block contenido %}
<!-- aquí va el contenido del formulario -->
{ % endblock %}

```

- { % extends "./plantilla.html" %}: Esto hace que este archivo herede la estructura base (encabezado, pie, estilos).
- { % block contenido %}: Dentro de este bloque se escribe el contenido único de esta página (el formulario)
- cargo: Es el **nombre del conjunto de datos** (una lista de objetos Cargo) que se envió desde views.py al HTML.
- { % for cargoTemporal in cargo %}: Aquí se **repite una vez por cada cargo** que venga en esa lista.
  - Cada vez que se repite, se guarda el dato actual en la variable cargoTemporal.



- Luego se muestran los datos de cada cargoTemporal dentro de <td> (columnas de la tabla):
    - {{cargoTemporal.id}} → muestra el ID.
    - {{cargoTemporal.nombres}} → muestra el nombre del cargo.
    - {{cargoTemporal.funciones}} → muestra las funciones del cargo.
    - {{cargoTemporal.horario}} → muestra el horario.
    - {{cargoTemporal.requisitos}} → muestra los requisitos.
    - {{cargoTemporal.sueldo}} → muestra el sueldo.
- 

## ¿Y los botones?

- **Botón de editar:**

Lleva al usuario a la página de edición del cargo que corresponde.

```
<a href="EditarCargo/{{cargoTemporal.id}}">
```

Usa href="EditarCargo/{{cargoTemporal.id}}", lo que crea un enlace como EditarCargo/1, EditarCargo/2, etc. Ese ID permite saber **qué cargo se va a modificar**.

- **Botón de eliminar:**

```
html
CopiarEditar
<a href="#" onclick="confirmarEliminacion('{{ url
'ECargo' cargoTemporal.id }}')">
```

Usa onclick="confirmarEliminacion(...)" para llamar una alerta antes de eliminar. Dentro, se usa {{ url 'ECargo' cargoTemporal.id }} para generar correctamente la URL de eliminación.



## Configuración de rutas específicas de la App Empresas

Cuando creamos una aplicación nueva en Django, como Empresas, también debemos crear un archivo llamado urls.py dentro de esa app. Este archivo nos permite definir las rutas internas (URLs) para esa app, es decir, decirle a Django qué hacer cuando alguien visita cierta dirección dentro de esa app.

📄 Código de urls.py dentro de la app Empresas:

Pega este contenido:	
<pre>  from django.urls import path from . import views  urlpatterns = [     path("", views.inicio, name='inicio'), ]</pre>	<pre>#Configuración de rutas específicas de la App Empresas from django.urls import path from . import views urlpatterns = []     path('', views.inicio, name='inicio'),</pre>
1	2
from django.urls import path	from . import views:
Con esto se importando la función path, que sirve para definir una URL. Viene del módulo django.urls que se crea por defecto.	Importa el archivo views.py que está en la misma carpeta (por eso se usa . ). Así podrás llamar a las funciones que escribas allí.
3	4
urlpatterns = []	path("", views.inicio, name='inicio'):
Es una lista donde escribes todas las rutas de esta aplicación. Django busca esta lista para saber qué hacer cuando el usuario entra a una dirección.	Aquí defines que cuando el usuario entra a la URL vacía "" (o sea <a href="http://localhost:8000/">http://localhost:8000/</a> ), se ejecutará la función inicio que está en views.py.
 Importante: path() siempre necesita:	
<ul style="list-style-type: none"><li>• El primer argumento es la ruta (ej. "", 'crear/', 'editar/5/') es un nombre que tu defines.</li><li>• El segundo argumento es la función que se va a ejecutar cuando el usuario entre a esa ruta.</li></ul>	



'cargo' no es una palabra mágica ni tiene que ver con el nombre del modelo o tabla.

Es simplemente un nombre nuevo que tú eliges para llamar a los datos dentro del archivo HTML.

Este nombre es la “etiqueta” o “clave” con la que la plantilla (el HTML) recibirá la información.

Es ESENCIAL que el nombre que pongas aquí (en el diccionario Python) sea exactamente el mismo que uses en el archivo HTML para mostrar o recorrer los datos.

```
C:\Users\ASUS\Videos\J2D\rene\Nueva carpeta\ProyectosD\trabajosDD\Aplicaciones\Empresas\templates\inicio.html (preview)
1 <table class="table table-bordered table-striped table-hover">
2   <thead>
3     <tr>
4       <th>ID</th>
5       <th>NOMBRE</th>
6       <th>FUNCIONES</th>
7       <th>HORARIOS</th>
8       <th>SUELDO</th>
9       <th>ACCIONES</th>
10      <th>REQUISITOS</th>
11    </tr>
12  </thead>
13  <tbody>
14    {% for cargoTemporal in cargo %}<tr>
15      <td>{{cargoTemporal.id}}</td>
16      <td>{{cargoTemporal.nombres}}</td>
17      <td>{{cargoTemporal.funciones}}</td>
18      <td>{{cargoTemporal.horario}}</td>
19      <td>{{cargoTemporal.requisitos}}</td>
20      <td>{{cargoTemporal.sueldo}}</td>
21      <td><a href="#" class="btn btn-primary">modelos.py </a></td>
22    </tr>
23  </tbody>
24 </table>
```

```
Aplicaciones > Empresas > views.py > ...
1 from django.shortcuts import render, redirect
2
3 #importar el modelo cargo
4 from .models import Cargo
5
6
7 #importar notificación de confirmación
8 from django.contrib import messages
9
10 # Create your views here.
11 # Esta función para renderizar el listado de cargos
12 def inicio(request):
13     listaCargos=Cargo.objects.all()
14     return render(request,"inicio.html",
15                  {'cargo':listaCargos})
```

```
Aplicaciones > Empresas > models.py > ...
1 from django.db import models
2
3 # Create your models here.
4 class Cargo(models.Model): # Define un modelo llamado Cargo (una tabla en la base de datos)
5     id = models.AutoField(primary_key=True) # Campo que se incrementa solo
6     nombres = models.CharField(max_length=100) # Campo para texto corto, máximo 100 caracteres
7     funciones = models.TextField() # Campo para texto largo, donde se describen las funciones
8     horario = models.CharField(max_length=500) # Campo para el horario del cargo
9     requisitos = models.TextField() # Campo para poner los requisitos del cargo
10    sueldo = models.DecimalField(decimal_places=2, max_digits=10) # Campo para el sueldo
11
12    def __str__(self):
13        fila="{0};{1};{2};{3}"
14        return fila.format(self.id, self.nombres, self.funciones, self.horario, self.requisitos, self.sueldo)
```

#### ¿Qué debes recordar?

Todo lo que pongas después del punto (.) tiene que ser un atributo real que exista en el modelo Cargo.

Si escribes mal un nombre, por ejemplo cargoTemporal.nombre (cuando el atributo real es nombres), no te mostrará nada en la tabla.

Si en la vista haces esto:

```
return render(request, "inicio.html", {'cargo': listaCargos})
```

Entonces en el HTML debes usar:

```
{% for cargoTemporal in cargo %}
    {{ cargoTemporal.nombres }}
{% endfor %}
```

Porque el HTML busca una variable llamada cargo que fue la que enviaste.

Si cambias el nombre en la vista, debes cambiarlo en el HTML.

Por ejemplo, si en la vista haces:

```
return render(request, "inicio.html", {'datosCargos':
    listaCargos})
```

Entonces en el HTML tienes que usar:

```
{% for cargoTemporal in datosCargos %}
    {{ cargoTemporal.nombres }}
{% endfor %}
```

Si no coinciden, el HTML no encontrará los datos y no mostrará nada y solo saldrá en pantalla los nombres de las columnas.



Para ver la pagina  
solo Ejecuta el  
servidor con python  
manage.py  
runserver

### ⚠ ¡Importante!

- Al principio, cuando solo haces esto y aún no escribes el código de la función inicio() ni la plantilla HTML, la página web se mostrará en blanco o incluso puede mostrar un error si no está bien enlazada.
- Esto es totalmente normal. Es porque todavía no has dicho qué debe aparecer en esa página.

## 11 Crear el Insertar para ingresar nuevos cargos (CREATE)

Este paso es para que los usuarios puedan registrar un nuevo cargo desde una página web usando un formulario.

### CREAR LA FUNCIÓN nuevoCargo EN views.py

En: Aplicaciones/Empresas/views.py agrega lo siguiente debajo de las funciones anteriores.

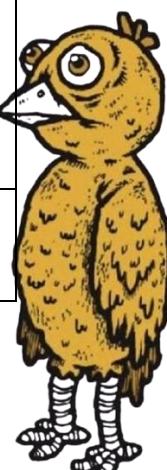
```
def nuevoCargo(request):
    return render(request, "nuevoCargo.html")
```

```
views.py
Aplicaciones > Empresas > views.py > nuevoCargo
1
2 from django.shortcuts import render, redirect
3
4 #importar el modelo cargo
5 from .models import Cargo
6
7
8 #importar la mensie d econfirmacion
9
10 from django.contrib import messages
11
12 # Create your views here.
13 # Esta función para renderizar el listado de cargos
14 def inicio (request):
15     listadoCargos=Cargo.objects.all()
16     return render(request,"inicio.html",
17     {'cargo':listadoCargos})
18
19
20 def nuevoCargo(request):
21     return render(request,"nuevoCargo.html")
22
```

Esta función se activa cuando alguien entra a la ruta /nuevoCargo y le muestra un formulario vacío (que crearás más adelante) para llenar los datos de un nuevo cargo.

1: def nuevoCargo(request):

2: return render(request,"nuevoCargo.html")

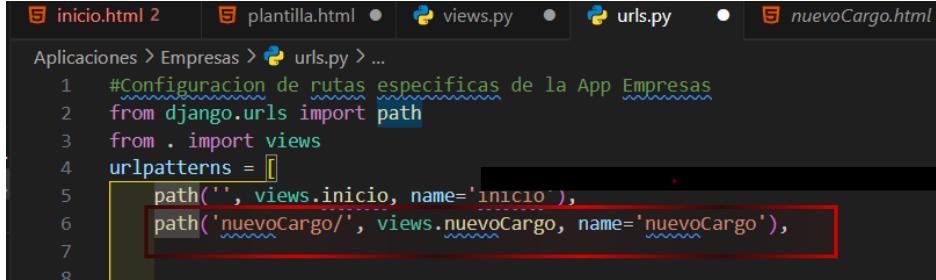


- |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>• <code>def</code> es una palabra reservada de Python para declarar funciones.</li> <li>• <code>nuevoCargo</code> es el nombre de la función (puedes poner otro, pero debe coincidir con la ruta después).</li> <li>• <code>(request)</code> es un parámetro obligatorio en Django. Django siempre manda ese objeto.</li> <li>• Este <code>request</code> contiene la petición del usuario: qué quiere hacer, qué método usó (GET o POST), etc.</li> <li>• Con esta función le vamos a mostrar una página (un formulario) al usuario.</li> </ul> | <ul style="list-style-type: none"> <li>• La función <code>render</code> es una herramienta de Django que "pinta" una plantilla HTML.</li> <li>• <code>request</code> se pasa como primer parámetro para saber quién está haciendo la petición.</li> <li>• "<code>nuevoCargo.html</code>" es el nombre del archivo HTML que vamos a mostrar (lo crearás en el siguiente paso).</li> <li>• El <code>render</code> devuelve una respuesta HTML, es decir: "muéstrale esta página al usuario".</li> </ul> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

 IMPORTANTE: Conectar esta función a una URL  
Para que esta función funcione, debes crear una ruta en tu archivo urls.py.  
Eso se hace así:

En Aplicaciones/Empresas/urls.py, agrega la siguiente línea en urlpatterns:

```
path('nuevoCargo', views.nuevoCargo),
```



```
Aplicaciones > Empresas > urls.py > ...
1 #Configuración de rutas específicas de la App Empresas
2 from django.urls import path
3 from . import views
4 urlpatterns = [
5     path('', views.inicio, name='inicio'),
6     path('nuevoCargo/', views.nuevoCargo, name='nuevoCargo'),
```

Cuando el usuario escriba /nuevoCargo en el navegador, Django va a ejecutar la función nuevoCargo que está en views.py.

 Recuerda:

- El archivo nuevoCargo.html aún no existe.
- Vamos a crearlo en pasos siguientes.
- Es muy importante que el nombre que pongas en el render coincida exactamente con el archivo HTML.



## Crear la función guardarCargo en views.py

Esta parte del sistema es la encargada de recibir los datos del formulario que llenó el usuario (nombre, funciones, horario, etc.), guardarlos en la base de datos, y luego mostrar un mensaje de confirmación en pantalla.

Abre el archivo **views.py** dentro de la carpeta Aplicaciones/Empresas/

Agrega la siguiente función:

```
def guardarCargo(request):
    # 1. Recoger los datos enviados por el formulario con POST
    nombres = request.POST["nombres"]
    funciones = request.POST["funciones"]
    horario = request.POST["horario"]
    requisitos = request.POST["requisitos"]
    sueldo = request.POST["sueldo"].replace(',', '.') # Aseguramos que el punto decimal sea correcto

    # 2. Crear y guardar el nuevo cargo usando el modelo Cargo
    nuevoCargo = Cargo.objects.create(
        nombres=nombres,
        funciones=funciones,
        horario=horario,
        requisitos=requisitos,
        sueldo=sueldo
    )

    # 3. Mostrar un mensaje de éxito en la página principal
    messages.success(request, "✓ Cargo creado exitosamente")

    # 4. Redirigir al usuario de vuelta al inicio
    return redirect('/')
```



```

4 #importar el modelo cargo
5 from .models import Cargo
6
7
8 #importarnto mensaje d eocnfirmaicon
9
10 from django.contrib import messages
11
12 # Create your views here.
13 # Esta función para renderizar el listado de cargos
14 Tabnine | Edit | Test | Explain | Document
15 def inicio (request):
16     listadoCargos=Cargo.objects.all()
17     return render(request,"inicio.html",
18     {'cargo':listadoCargos})
19
20
21
22
23
24
25 Tabnine | Edit | Test | Explain | Document
26 def nuevoCargo(request):
27     return render(request,"nuevoCargo.html")
28
29
30
31
32
33
34
35 Tabnine | Edit | Test | Explain | Document
36 def guardarCargo(request):
37     nombres=request.POST["nombres"]
38     funciones=request.POST["funciones"]
39     horario=request.POST["horario"]
40     requisitos=request.POST["requisitos"]
41     sueldo=request.POST["sueldo"].replace(',','.')
42     nuevoCargo=Cargo.objects.create(nombres=nombres,funciones=funciones, horario=horario, requisitos=requisitos, sueldo=sueldo)
43     #antes del redirec poner lso siguiente/mensaje de confirmacion/reas mos un mensaje d eocnfirmaicon con success
44     messages.success(request,"Cargo creado exitosamente")
45     return redirect('/')
46

```

**1:** from django.shortcuts import render, redirect  
 → Aquí estamos trayendo dos funciones desde Django:

render: la usamos para mostrar páginas HTML desde una vista.

redirect: la usamos para enviar al usuario a otra URL después de hacer una acción (por ejemplo, después de guardar un nuevo cargo, lo enviamos al inicio /).

**3 :**

```

nombres = request.POST["nombres"]
funciones = request.POST["funciones"]
horario = request.POST["horario"]
requisitos = request.POST["requisitos"]
sueldo = request.POST["sueldo"]

```

→ Aquí accedemos a los datos que el usuario envió desde el formulario HTML.  
 → request.POST es un diccionario que contiene todos los datos que llegaron usando el método POST.  
 → Las Nombres de variables dentro de los corchetes ["Nombre variable"] debe coincidir exactamente con el atributo name de cada campo en el formulario.

**2: def guardarCargo(request):**

→ Aquí creamos una nueva función en Python llamada guardarCargo.  
 → Django ejecutará esta función cuando se reciba una **petición POST desde el formulario HTML**.  
 → El parámetro request contiene toda la información que viene del navegador del usuario (como los datos que se llenaron en el formulario).

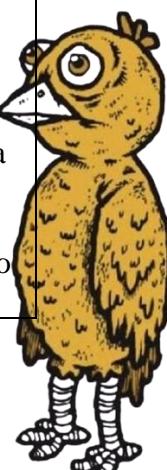
**4:**

```

Cargo.objects.create(
    nombres=nombres,
    funciones=funciones,
    horario=horario,
    requisitos=requisitos,
    sueldo=sueldo
)

```

→ Aquí usamos el modelo Cargo para crear un nuevo registro en la base de datos con los datos que recibimos de las variables que creamos para que guardar los datos que recibimos del formulario.  
 → Cargo.objects.create() crea y guarda el objeto automáticamente.



→ Tomamos el valor del campo de texto llamado nombres que el usuario escribió en el formulario.	→ Cada atributo debe coincidir con los campos definidos en el modelo Cargo (en models.py).
→ request.POST es como una caja que guarda todos los datos enviados desde el formulario con método POST.	
→ ["nombres"] debe coincidir exactamente con name="nombres" en el HTM	

## 5: return redirect('/')

→ Después de guardar el nuevo cargo, esta línea redirige (envía) al usuario a la página principal (/), que en nuestro caso mostrará el listado de cargos actualizado.  
 → Esto evita que si el usuario recargue la página, se vuelva a enviar el formulario (buena práctica).

### IMPORTANTE

- Esta función solo se ejecuta cuando el navegador envía un formulario usando el método POST (porque accede a request.POST).
- Si intentas acceder a esta función con GET, no tendría datos y causaría error, pero normalmente no se accede así porque el formulario usa POST.

Cuando creamos una función en views.py y la conectamos con una URL en urls.py, muchas veces necesitamos que el usuario pueda llegar a esa función usando un enlace o un formulario en la página web.

En este caso, para **guardar un nuevo cargo**, usamos un formulario con un botón de "Guardar" que envía los datos a la URL /guardarCargo.

Pero también, para mostrar la página donde el usuario puede llenar el formulario (que es otra función llamada nuevoCargo), debemos crear un enlace o link que lleve a esa página.

## Agregar la ruta guardarCargo en urls.py

Ahora que ya tenemos la función guardarCargo en views.py, necesitamos decirle a Django que esa función debe ejecutarse cuando el usuario haga clic en el botón "Guardar" del formulario.

Abre el archivo urls.py de tu aplicación Empresas (no el principal del proyecto, sino el que está en la carpeta Aplicaciones/Empresas/urls.py)

Dentro de urlpatterns, agrega la siguiente línea:

```
path('guardarCargo', views.guardarCargo)
```



```

EXPLORADOR          settings.py      apps.py      models.py      durin.py
PROYECTOSDJ
trabajosDD
  Aplicaciones
    Empleos
      __pycache__
      migrations
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
      urls.py

```

```

trabajosDD > Aplicaciones > Empresas > urls.py
1  #Configuracion de rutas específicas de la App Empresas
2  from django.urls import path
3  from . import views
4  urlpatterns=[
5      path('', views.inicio),
6      path('nuevoCargo', views.nuevoCargo),
7      path ('guardarCargo', views.guardarCargo)
8  ]
9
10

```

'guardarCargo': es la URL que el navegador pedirá cuando se envíe el formulario (esto se conecta con el atributo action del formulario en nuevoCargo.html).

views.guardarCargo: es la función que se va a ejecutar cuando alguien acceda a esa URL. Es decir, se conecta directamente con la función que creaste en views.py.

## Crear la página nuevoCargo.html

Ahora vamos a crear la página donde el usuario podrá llenar el formulario para ingresar un nuevo cargo.

Ve a la carpeta donde están tus plantillas HTML:

Ruta: Aplicaciones/Empresas/templates/

Dentro de esa carpeta, crea un archivo nuevo llamado:  
nuevoCargo.html

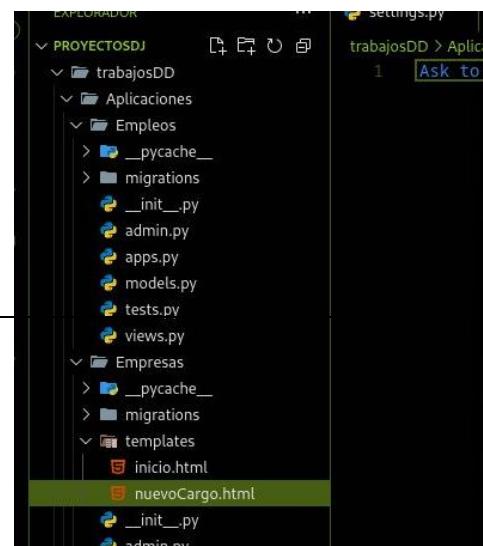
Escribe el siguiente código en ese archivo HTML:

```

{% extends "./plantilla.html" %}
{% block contenido %}
<div class="row">
  <div class="col-md-3"></div>
  <div class="col-md-6 bg-light">
    <h1>Nuevo Cargo</h1>
    <form action="{% url 'guardarCargo' %}" method="post">

      {% csrf_token %}
      <label for="">Nombre:</label><br>
      <input class="form-control" type="text" name="nombres" id="nombres">
      <br>
      <label for="">Requisitos:</label><br>
      <input class="form-control" type="text" name="requisitos" id="requisitos">
      <br>
      <label for="">Funciones:</label><br>
      <input class="form-control" type="text" name="funciones" id="funciones">

```



```

<br>
<label for="">Horario:</label><br>
<input class="form-control" type="text" name="horario" id="horario">
<br>
<label for="">Sueldo:</label><br>
<input class="form-control" type="text" name="sueldo" id="sueldo">

<br>
    <button class="form-control" type="submit" class="btn btn-success">Guardar</button>
    <a href="/" class="btn btn-outline-danger">Cancelar</a> <!-- invoca al inicio-->
</form>

</div>
<div class="col-m-3"></div>
</div>

{ % endblock %}

```

## Estructura base del formulario

{ % extends "./plantilla.html" % }

Esto indica que esta plantilla usa como base el archivo plantilla.html. Así heredamos el diseño general, como el encabezado, pie de página y estilos CSS.

{ % block contenido % }

Abre el bloque donde pondremos el contenido único de esta página, que en este caso es el formulario para crear un nuevo cargo.

<form action="{ % url 'guardarCargo' % }" method="post">

Empieza un formulario.

action="{ % url 'guardarCargo' % }" significa que cuando le des a guardar, se enviarán los datos a la función que tiene el nombre guardarCargo (definida en urls.py).

method="post" dice que los datos se envían con el método POST (para crear o guardar cosas).

{ % csrf\_token % }

Es una medida de seguridad de Django para evitar ataques de tipo CSRF (falsificación de solicitudes). Siempre se debe poner dentro de los formularios que usan POST.

Los siguientes campos (input) tienen un atributo name que debe coincidir con lo que esperas en request.POST dentro de tu función guardarCargo. Por ejemplo, el input con name="nombres" envía el dato que luego obtienes con request.POST["nombres"].

<button type="submit">Guardar</button>

Es el botón que envía el formulario.



```
<a href="/" class="btn btn-outline-danger">Cancelar</a>  
Es un enlace para cancelar y volver a la página principal (/).
```

```
{% endblock %}  
Cierra el bloque contenido.
```

### Paso para agregar el enlace “Nuevo Cargo” en la plantilla inicio.html

1. **Abre el archivo de la plantilla que usas para la página inicio** (por ejemplo, inicio.html).
2. **Dentro del bloque de contenido { % block contenido %} agrega este código para el enlace:**

```
<div class="text-end">  
    <a href="{% url 'nuevoCargo' %}">Nuevo Cargo</a>  
</div>
```

```
{% extends "./plantilla.html" %}  
{% block contenido %}  
  
<h1>Listado de Cargos</h1>  
{% if messages %}  
    {% for mensaje in messages %}  
        <script>  
            Swal.fire({  
                title: "¡Éxito!",  
                text: "{{ mensaje }}",  
                icon: "error"  
            });  
        </script>  
    {% endfor %}  
    {% endif %}  
  
<div class="text-end">  
    <a href="{% url 'nuevoCargo' %}">Nuevo Cargo</a>  
</div>
```

Se vera algo así

## Contenido del encabezado aquí

Así se ve el bloque de código, al ejecutar

## Listado de Cargos

[Nuevo Cargo](#)

ID NOMBRE FUNCIONES HORARIOS SUELDO ACCIONES REQUISITOS

## Contenido del pie de página aquí



## Contenido del encabezado aquí

### Listado de Cargos

Nuevo Cargo	ID	NOMBRE	FUNCIONES	HORARIOS	SUELDO	ACCIONES	REQUISITOS
-------------	----	--------	-----------	----------	--------	----------	------------

### Contenido del pie de página aquí

Es la línea de código que teníamos en inicio.html

```
<a href="{% url 'nuevoCargo' %}">Nuevo Cargo</a>
```

```
urls.py
Aplicaciones > Empresas > urls.py > ...
1 #Configuración de rutas específicas de la App Empresas
2 from django.urls import path
3 from . import views
4 urlpatterns = [
5     path('', views.inicio, name='inicio'),
6     path('nuevoCargo/', views.nuevoCargo, name='nuevoCargo')
```

```
#importando mensajes de confirmacion
from django.contrib import messages

# Create your views here.
# Esta función para renderizar el listado de cargos
def inicio(request):
    listaCargos=Cargo.objects.all()
    return render(request,"inicio.html",
    {'cargo':listaCargos})
```

```
def nuevoCargo(request):
    return render(request,"nuevoCargo.html")
```

Busca en base al name por eso se usa código jinja 2

Este abre la función que abre una nueva ventana

Con el render hace el truco y solo devuelve la pagina sin datos

## Contenido del encabezado aquí

### Nuevo Cargo

Nombre:

Requisitos:

Funciones:

Horario:

Sueldo:

### Contenido del pie de página aquí



## Contenido del encabezado aquí

### Nuevo Cargo

Nombre:

Requisitos:

Funciones:

Horario:

Sueldo:

<button class="form-control" type="submit" class="btn btn-success">Guardar</button>

El botón usa submit para cuando se de click guarda todos los datos de los cuadros de texto y los guardar con el nombre que se le da, eso se hace en el código con el atributo name, y se enviar con el método post, al link que se especifica con action en esta línea <form action="{% url 'guardarCargo' %}" método="post"> por eso busca ese link en url.py

## Contenido del pie de página aquí

```
#Configuración de rutas específicas de la App Empresas
from django.urls import path
from . import views
urlpatterns = [
    path('', views.inicio, name='inicio'),
    path('nuevoCargo/', views.nuevoCargo, name='nuevoCargo'),
    path('guardarCargo/', views.guardarCargo, name='guardarCargo'),
```

<button class="form-control" type="submit" class="btn btn-success">Guardar</button>

Igualmente busca el link y ejecuta la función, en esta la función reside el request donde están los datos por eso lo usaremos que es la petición que siempre va si o si, después se crean variables nuevas para poder guardar ahí los datos que viene del formulario trajimos del formulario, para luego usar el modelo y con la función objects que nos permite consultar, actualizar literal todo usamos el create para crear nuevo usuario

```
Tabnine | Edit | Test | Explain | Document
def nuevoCargo(request):
    return render(request, "nuevoCargo.html")
```

```
Tabnine | Edit | Test | Explain | Document
def guardarCargo(request):
    nombres=request.POST["nombres"]
    funciones=request.POST["funciones"]
    horario=request.POST["horario"]
    requisitos=request.POST["requisitos"]
    sueldo=request.POST["sueldo"].replace(',', '.')
    nuevoCargo=Cargo.objects.create(nombres=nombres, funciones=funciones, horario=horario, requisitos=requisitos, sueldo=sueldo)
    #antes del redirect poner lo siguiente/mensaje de confirmación/creas mos un mensaje de confirmación con success
    messages.success(request, "Cargo creado exitosamente")
    return redirect('/')
```

## Contenido del encabezado aquí

### Listado de Cargos

ID	NOMBRE	FUNCIONES	HORARIOS	SUELDO	ACCIONES
21	1	1	1	1.00	<a href="#">Editar</a> <a href="#">Eliminar</a>

## Contenido del pie de página aquí



### Mensaje

Cargo creado exitosamente

OK

<button class="form-control" type="submit" class="btn btn-success">Guardar</button>

Es muy importante los nombre abajo explico cual depende de cual, ya que a qui intervine el modelo que esta models.py, y los datos enviados desde el formulario



```

Aplicaciones > Empresas > templates > nuevoCargo.html > div.row > div.col-md-6.bg-light > h1
1  xtends "./plantilla.html %}
2  lock contenido %}
3  class="row">
4  <div class="col-md-3"></div>
5  <div class="col-md-6 bg-light">
6  <h1>Nuevo Cargo</h1>
7  <form action="{% url 'guardarCargo' %}" method="post">
8
9    {% csrf_token %}
10   <label for="Nombre"></label><br>
11   <input class="form-control" type="text" name="nombres" id="nombres">
12   <br>
13   <label for="Requisitos"></label><br>
14   <input class="form-control" type="text" name="requisitos" id="requisitos">
15   <br>
16   <label for="Funciones"></label><br>
17   <input class="form-control" type="text" name="funciones" id="funciones">
18   <br>
19   <label for="Horario"></label><br>
20   <input class="form-control" type="text" name="horario" id="horario">
21   <br>
22   <label for="Sueldo"></label><br>
23   <input class="form-control" type="text" name="sueldo" id="sueldo">
24
25   <br>
26   <button class="form-control" type="submit" class="btn btn-primary">Guardar</button>
27   <a href="/" class="btn btn-outline-danger">Cancelar</a>
28 </form>
29
30 </div>
31 <div class="col-m-3"></div>
32 v>
33
34
35 ndblock %}
36
37

```

```

nuevoCargo.html | views.py | models.py
Aplicaciones > Empresas > views.py > guardarCargo
19
20 def nuevoCargo(request):
21     return render(request, "nuevoCargo.html")
22
23
24 def guardarCargo(request):
25     nombres=request.POST["nombres"]
26     requisitos=request.POST["requisitos"]
27     funciones=request.POST["funciones"]
28     horario=request.POST["horario"]
29     sueldo=request.POST["sueldo"].replace(',', '.')
30     nuevoCargo.Cargo.objects.create(nombres=nombres, funciones=funciones, horario=horario)
31     #antes del redirect poner los siguientes mensajes de confirmación/creas
32     messages.success(request,"Cargo creado exitosamente")
33     return redirect('/')
34
35
36 def eliminarCargo(request,id):
37     CargoEliminar=Cargo.objects.get(id=id)
38     CargoEliminar.delete()
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
487
488
489
489
490
491
492
493
494
495
496
497
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
648
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
678
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
758
758
759
760
761
762
763
764
765
766
767
767
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
816
817
818
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
845
846
846
847
848
848
849
849
850
851
852
853
854
855
856
857
857
858
859
859
860
861
862
863
864
865
866
866
867
868
868
869
869
870
871
872
873
874
875
876
876
877
878
878
879
879
880
881
882
883
884
885
886
886
887
888
888
889
889
890
891
892
893
894
895
895
896
896
897
897
898
898
899
899
900
901
902
903
904
905
905
906
907
907
908
909
909
910
911
912
913
914
914
915
915
916
916
917
917
918
918
919
919
920
921
922
923
924
924
925
925
926
926
927
927
928
928
929
929
930
931
932
933
934
934
935
935
936
936
937
937
938
938
939
939
940
941
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
951
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
158
```

```

def Mformnualrio(request, id):
    # Busca el cargo que queremos editar, usando su ID
    cargoEditar = Cargo.objects.get(id=id)
    # Envía los datos del cargo a la plantilla Editar.html
    return render(request, "Editar.html", {'CargoEditar': cargoEditar})

```

```

models.py          views.py
Aplicaciones > Empresas > views.py > Mformnualrio
12   # Create your views here.
13   # Esta función para renderizar el listado de cargos
14   def inicio(request):
15       listadoCargos=Cargo.objects.all()
16       return render(request,"inicio.html",
17           {'cargo':listadoCargos})
18
19
20   def nuevoCargo(request):
21       return render(request,"nuevoCargo.html")
22
23
24   def guardarCargo(request):
25       nombres=request.POST["nombres"]
26       requisitos=request.POST["requisitos"]
27       funciones=request.POST["funciones"]
28       horario=request.POST["horario"]
29       sueldo=request.POST["sueldo"].replace(',','.')
30       nuevoCargo=Cargo.objects.create(nombres=nombres,funciones=funciones, horario=horario, requisitos=requisitos)
31       #antes del redirect poner lso siguiente/mensaje de confirmacion/creas mos un mensaje
32       messages.success(request,"Cargo creado exitosamente")
33       return redirect('/')
34
35
36   def Mformnualrio(request, id):
37       cargoEditar = Cargo.objects.get(id=id)
38       return render(request, "Editar.html", {'CargoEditar': cargoEditar})
39

```

## ¿Por qué?

Esta función es para mostrar un formulario con los datos del cargo que queremos actualizar. El usuario podrá ver los datos actuales para cambiar lo que necesite.

```
def Mformnualrio(request, id):
```

- Esta línea define una función en Django llamada Mformnualrio.
- ◆ request es el objeto que contiene la información de la petición del navegador.
- ◆ id es el número que viene desde la URL, y representa el ID del cargo que queremos editar.



```
cargoEditar = Cargo.objects.get(id=id)
```

- Busca en la base de datos el cargo que tenga ese id.
- ◆ `Cargo.objects.get(id=id)` significa: “tráeme el cargo cuyo campo id sea igual al id
- ◆ El resultado se guarda en la variable `cargoEditar`.

```
return render(request, "Editar.html", {'CargoEditar': cargoEditar})
```

- Devuelve una página HTML como respuesta.
- ◆ `"Editar.html"` es la plantilla que se va a mostrar en pantalla.
- ◆ Le enviamos a esa plantilla una variable llamada `CargoEditar` que contiene la información del cargo que vamos a editar.

## Paso 2: Crear la ruta para abrir el formulario de edición

### Archivo: urls.py

1. Abre `urls.py`.
2. Añade esta línea para que Django sepa a qué función ir cuando alguien visite `/EditarCargo/<id>`:

```
path('EditarCargo/<id>/', views.Mformnualrio, name='EditarCargo'),
```

```
models.py  views.py  urls.py  X
Aplicaciones > Empresas > urls.py > ...
1 #Configuración de rutas específicas de la App Empresas
2 from django.urls import path
3 from . import views
4 urlpatterns = [
5     path('', views.inicio, name='inicio'),
6     path('nuevoCargo/', views.nuevoCargo, name='nuevoCargo'),
7     path('guardarCargo/', views.guardarCargo, name='guardarCargo'),
8     path('EditarCargo/<id>/', views.Mformnualrio, name='EditarCargo')
```



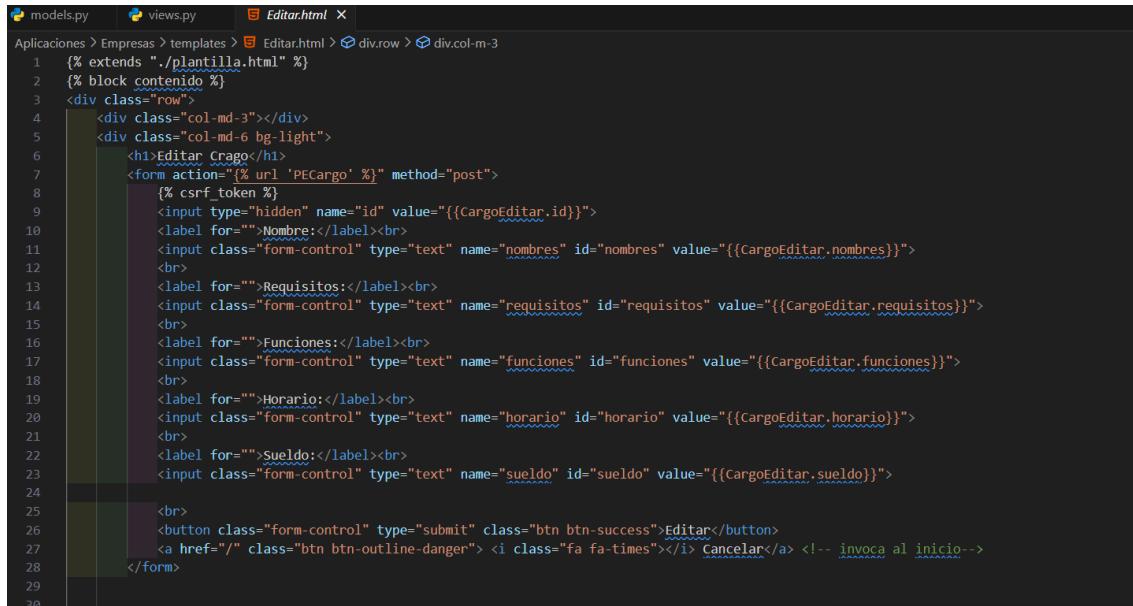
## ¿Por qué?

Esto crea la dirección web donde el usuario puede entrar para editar un cargo específico. El <id> es el número del cargo que queremos modificar.

### Paso 3: Crear la plantilla HTML del formulario de edición

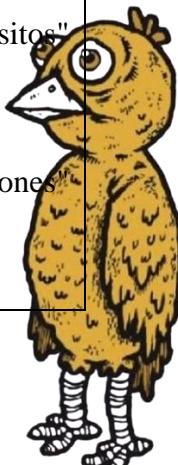
#### Archivo: Editar.html

1. Crea el archivo Editar.html en la carpeta templates.
2. Pega este código para que muestre los datos actuales y permita editarlos:



```
{% extends "./plantilla.html" %}  
{% block contenido %}  
<div class="row">  
    <div class="col-md-3"></div>  
    <div class="col-md-6 bg-light">  
        <h1>Editar Crago</h1>  
        <form action="{% url 'PECargo' %}" method="post">  
            {% csrf_token %}  
            <input type="hidden" name="id" value="{{CargoEditar.id}}>  
            <label for="">Nombre:</label><br>  
            <input class="form-control" type="text" name="nombres" id="nombres" value="{{CargoEditar.nombres}}>  
            <br>  
            <label for="">Requisitos:</label><br>  
            <input class="form-control" type="text" name="requisitos" id="requisitos" value="{{CargoEditar.requisitos}}>  
            <br>  
            <label for="">Funciones:</label><br>  
            <input class="form-control" type="text" name="funciones" id="funciones" value="{{CargoEditar.funciones}}>  
            <br>  
            <label for="">Horario:</label><br>  
            <input class="form-control" type="text" name="horario" id="horario" value="{{CargoEditar.horario}}>  
            <br>  
            <label for="">Sueldo:</label><br>  
            <input class="form-control" type="text" name="sueldo" id="sueldo" value="{{CargoEditar.sueldo}}>  
            <br>  
            <button class="form-control" type="submit" class="btn btn-success">Editar</button>  
            <a href="/" class="btn btn-outline-danger">  Cancelar</a> <!-- invoca al inicio-->  
        </form>  
    </div>  
</div>
```

```
{% extends "./plantilla.html" %}  
{% block contenido %}  
<div class="row">  
    <div class="col-md-3"></div>  
    <div class="col-md-6 bg-light">  
        <h1>Editar Crago</h1>  
        <form action="{% url 'PECargo' %}" method="post">  
            {% csrf_token %}  
            <input type="hidden" name="id" value="{{CargoEditar.id}}>  
            <label for="">Nombre:</label><br>  
                <input class="form-control" type="text" name="nombres" id="nombres" value="{{CargoEditar.nombres}}>  
            <br>  
            <label for="">Requisitos:</label><br>  
                <input class="form-control" type="text" name="requisitos" id="requisitos" value="{{CargoEditar.requisitos}}>  
            <br>  
            <label for="">Funciones:</label><br>  
                <input class="form-control" type="text" name="funciones" id="funciones" value="{{CargoEditar.funciones}}>  
            <br>  
            <label for="">Horario:</label><br>
```



```

<input class="form-control" type="text" name="horario" id="horario"
value="{{CargoEditar.horario}}>
<br>
<label for="">Sueldo:</label><br>
    <input class="form-control" type="text" name="sueldo" id="sueldo"
value="{{CargoEditar.sueldo}}>

<br>
<button class="form-control" type="submit" class="btn btn-success">Editar</button>
<a href="/" class="btn btn-outline-danger"><i class="fa fa-times"></i> Cancelar</a> <!--
- invoca al inicio-->
</form>

</div>
<div class="col-m-3"></div>
</div>

{ % endblock %}

```

## ¿Por qué?

- Los value="{{CargoEditar.campo}}" muestran la información actual.
- El campo oculto con name="id" es importante para saber cuál registro actualizar cuando enviamos el formulario.

## Paso 4: Crear la función para procesar y guardar la edición

### Archivo: views.py

1. En views.py, agrega esta función que recibe los datos del formulario y actualiza el cargo:

```

def ProcesarEdicionCargo(request):
    # Recibimos los datos enviados por POST
    id = request.POST["id"]
    nombres = request.POST["nombres"]
    funciones = request.POST["funciones"]
    horario = request.POST["horario"]
    requisitos = request.POST["requisitos"]
    sueldo = request.POST["sueldo"].replace(',', '.')

    # Buscamos el cargo original por ID
    cargo = Cargo.objects.get(id=id)

    # Actualizamos cada campo con los datos nuevos
    cargo.nombres = nombres
    cargo.funciones = funciones
    cargo.horario = horario

```



```
cargo.requisitos = requisitos  
cargo.sueldo = sueldo  
  
# Guardamos los cambios en la base de datos  
cargo.save()  
  
# Mensaje de confirmación para el usuario  
messages.success(request, "Cargo Actualizado exitosamente")  
  
# Redirigimos a la página principal  
return redirect('/')
```

### ¿Por qué?

Esta función recibe los datos del formulario, actualiza el registro y guarda los cambios.

### Paso 5: Crear la ruta para procesar la edición

#### Archivo: urls.py

1. Agrega esta línea en urls.py para conectar la URL con la función que guarda los cambios:

```
path('PECargo', views.ProcesarEdicionCargo),
```

### ¿Por qué?

Esta ruta recibe el formulario con el método POST y ejecuta la función que actualiza el cargo.

### Paso 6: Crear enlace para editar en la lista de cargos

#### Archivo que se modifica: inicio.html

### ¿Qué se va a hacer en este paso?

Vamos a agregar un **botón de editar** en cada fila de la tabla de cargos, para que el usuario pueda **abrir un formulario con los datos ya llenados** del cargo que desea modificar.

### ¿Por qué se hace esto?

Este botón es necesario para que el usuario pueda **ver los datos del cargo actual**, cambiar lo que necesite y **actualizarlo**. Al hacer clic en el botón, se abrirá una nueva página con un formulario que ya contiene los datos del cargo.



## Código que debes agregar o corregir:

1. Abre el archivo inicio.html.
2. Busca la parte donde dice {% for cargoTemporal in cargo %}. Eso es el ciclo que muestra todos los cargos.
3. Dentro de la columna <td> de acciones, reemplaza la línea del botón editar por esta:

```
<a class="btn btn-warning" href="{% url 'EditarCargo' cargoTemporal.id %}">
    <i class="fa fa-pen"></i>
</a>
```

```
models.py | views.py | inicio.html 2 x
Aplicaciones > Empresas > templates > inicio.html > table.table-bordered.table-striped.table-hover > tbody > tr > td > a
22     <table class="table table-bordered table-striped table-hover">
23         <thead>
24             <tr>
25                 <th>ID</th>
26                 <th>Nombre</th>
27                 <th>Funciones</th>
28                 <th>Horarios</th>
29                 <th>Suelo</th>
30                 <th>Acciones</th>
31             </tr>
32         </thead>
33         <tbody>
34             {% for cargoTemporal in cargo %}
35             <tr>
36                 <td>{{cargoTemporal.id}}</td>
37                 <td>{{cargoTemporal.nombres}}</td>
38                 <td>{{cargoTemporal.funciones}}</td>
39                 <td>{{cargoTemporal.horario}}</td>
40                 <td>{{cargoTemporal.requisitos}}</td>
41                 <td>{{cargoTemporal.sueldo}}</td>
42                 <td>
43                     <a class="btn btn-warning" href="{% url 'EditarCargo' cargoTemporal.id %}"><i class="fa fa-pen"></i>Editar</a>
44                     <a href="#" onclick="confirmarEliminacion('{% url 'ECargo' cargoTemporal.id %}')"><i class="fa fa-trash"></i>Eliminar</a>
45                 </td>
46             </tr>
47             {% endfor %}
48         </tbody>
49     </table>
```

Flujo

Es esta linea del código

```
<a class="btn btn-warning" href="{% url
'EditarCargo' cargoTemporal.id %}"><i class="fa fa-
pen"></i>Editar</a>
```

Al preionar el se lleva el valor que esta en al columna ID , en etse caso viaje el 21.

## Contenido del encabezado aquí

### Listado de Cargos

Nuevo Cargo	ID	NOMBRE	FUNCIONES	HORARIOS	SUELDO	ACCIONES	REQUISITOS
	21	1	1	1	1.00	<a href="#">Editar</a>	<a href="#">Eliminar</a>

## Contenido del pie de página aquí

```
models.py | views.py | urls.py | ...
Aplicaciones > Empresas > urls.py > ...
#Configuración de rutas específicas de la App Empresas
from django.urls import path
from . import views
urlpatterns = [
    path('', views.inicio, name='inicio'),
    path('nuevoCargo/', views.nuevoCargo, name='nuevoCargo'),
    path('guardarCargo/', views.guardarCargo, name='guardarCargo'),
    path('EditarCargo/<id>', views.formulario, name='EditarCargo'),
    path('RECargo/', views.ProcessEdicionCargo, name='RECargo'),
```



```

    Tabnine | Edit | Test | Explain | Document
def nuevoCargo(request):
    return render(request, "nuevoCargo.html")

    Tabnine | Edit | Test | Explain | Document
def guardarCargo(request):
    nombres=request.POST["nombres"]
    requisitos=request.POST["requisitos"]
    funciones=request.POST["funciones"]
    horario=request.POST["horario"]
    sueldo=request.POST["sueldo"].replace(',','.')
    nuevoCargo=Cargo.objects.create(nombres=nombres,funciones=funciones, ho
#antes del redirect poner lo siguiente/mensaje de confirmacion/creas m
    messages.success(request,"Cargo creado exitosamente")
    return redirect('/')

    Tabnine | Edit | Test | Explain | Document
def mformulario(request, id):
    cargoEditar = Cargo.objects.get(id=id)
    return render(request, "Editar.html", {'CargoEditar': cargoEditar})

```

Entonce se captura los datos en un variable, pero con el get=id capturamos específicamente datos de un usuario en específico eso se hico gracias al id que viajo

Asi los datos de ese suario viajan y podremos mostralos para editar, y usamos lógica mostrar los datos en el formulario de edición

Y por eso se carga solo

## Contenido del encabezado aquí

### Editar Crago

Nombre:

Requisitos:

Funciones:

Horario:

Sueldo:

[Editar](#) [Cancelar](#)

<button class="form-control" type="submit" class="btn btn-success">Editar</button>

Es Con esa línea se crea El botón editar , y esa com en la otra ejecuta el action que lleva los datos por la url

<form action="{% url 'PECargo' %}" method="post">

## Contenido del pie de página aquí



```
#comenzar la definicion de rutas especificadas de la app Empresas
from django.urls import path
from . import views
urlpatterns = [
    path('', views.inicio, name='inicio'),
    path('nuevoCargo/', views.nuevoCargo, name='nuevoCargo'),
    path('guardarCargo/', views.guardarCargo, name='guardarCargo'),
    path('EditarCargo/<id>', views.Mformnualrio, name='EditarCargo'),
    path('PECargo/', views.ProcesarEdicionCargo, name='PECargo'),
```

```
Tabnine | Edit | Test | Explain | Document
def ProcesarEdicionCargo(request):
    id=request.POST["id"]
    nombres=request.POST["nombres"]
    funciones=request.POST["funciones"]
    horario=request.POST["horario"]
    requisitos=request.POST["requisitos"]
    sueldo=request.POST["sueldo"].replace(',', '.')
    cargo=Cargo.objects.get(id=id) #buscra cargo a editar
    #antes del redirect poner lso siguiente/mensaje de confirmacion/cre

    cargo.nombres=nombres
    cargo.funciones=funciones
    cargo.horario=horario
    cargo.requisitos=requisitos
    cargo.sueldo=sueldo
    cargo.save()
    messages.success(request,"Cargo Actualizado exitosamente")
    return redirect('/')
```

Contenido del encabezado aquí

## Listado de Cargos

Nuevo Cargo	ID	NOMBRE	FUNCIONES	HORARIOS	SUELDO	ACCIONES	REQUISITOS
	21	1	1	12	1	1.00	<a href="#">Editar</a> <a href="#">Eliminar</a>

Contenido del pie de página aquí

Cargo Actualizado exitosamente

OK



Mensaje



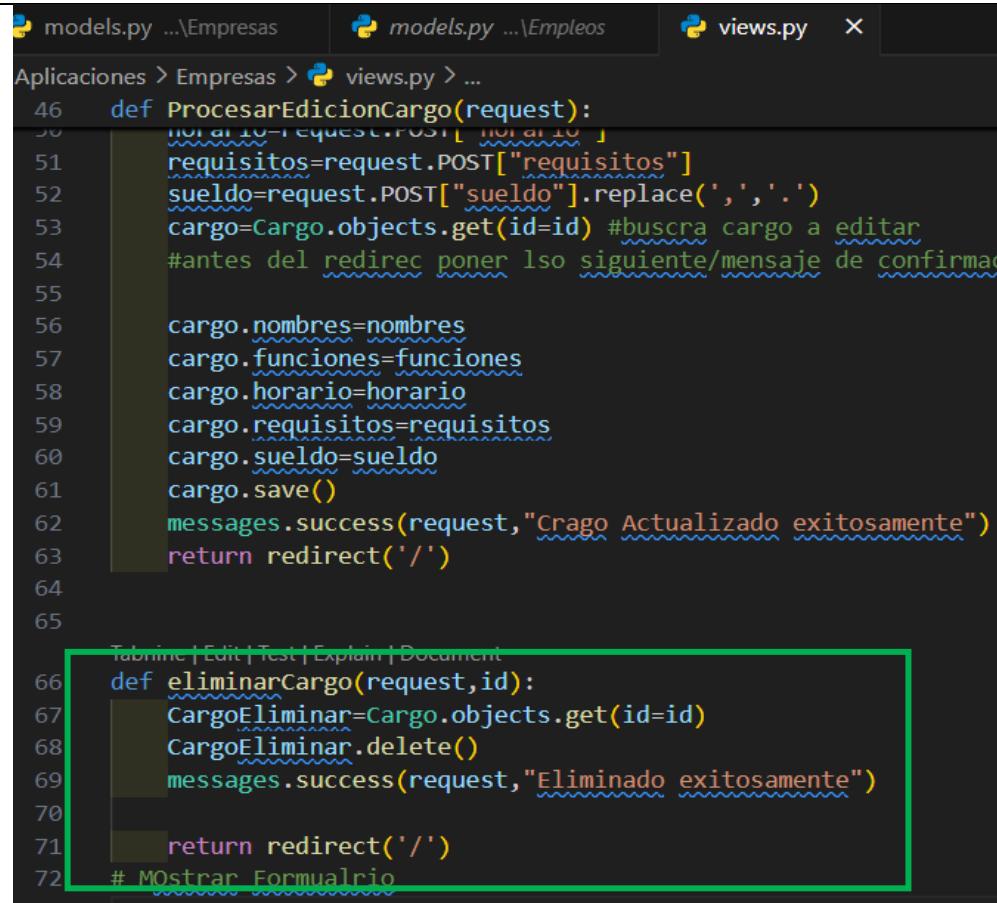
## 13 Crear delete para Eliminar un cargo

### Vista en views.py

Agrega esta función que se encarga de eliminar un cargo según su ID:

```
def eliminarCargo(request,id):
    CargoEliminar=Cargo.objects.get(id=id)
    CargoEliminar.delete()
    messages.success(request,"Eliminado exitosamente")

    return redirect('/')
```



```
models.py ...\\Empresas  models.py ...\\Empleos  views.py  X
Aplicaciones > Empresas > views.py > ...
46     def ProcesarEdicionCargo(request):
47         nombres=request.POST["nombres"]
48         requisitos=request.POST["requisitos"]
49         sueldo=request.POST["sueldo"].replace(',','.')
50         cargo=Cargo.objects.get(id=id) #busca cargo a editar
51         #antes del redirect poner lso siguiente/mensaje de confirmacion
52
53         cargo.nombres=nombres
54         cargo.funciones=funciones
55         cargo.horario=horario
56         cargo.requisitos=requisitos
57         cargo.sueldo=sueldo
58         cargo.save()
59         messages.success(request,"Cargo Actualizado exitosamente")
60         return redirect('/')
61
62
63
64
65
66     def eliminarCargo(request,id):
67         CargoEliminar=Cargo.objects.get(id=id)
68         CargoEliminar.delete()
69         messages.success(request,"Eliminado exitosamente")
70
71         return redirect('/')
72     # Mostrar Formulario
```

```
def eliminarCargo(request, id):
    CargoEliminar = Cargo.objects.get(id=id) # Busca el cargo con ese ID
    CargoEliminar.delete() # Elimina el registro de la base de datos
    messages.success(request, "Eliminado exitosamente") # Muestra mensaje en la plantilla
    return redirect('/') # Redirige a la página de inicio (listado)
```



## Ruta en urls.py

Agrega esta línea en tu lista de urlpatterns:

```
path('ECargo/<id>/', views.eliminarCargo, name='ECargo'),  
  
#Configuración de rutas específicas de la App Empresas  
from django.urls import path  
from . import views  
urlpatterns = [  
    path('', views.inicio, name='inicio'),  
    path('nuevoCargo/', views.nuevoCargo, name='nuevoCargo'),  
    path('guardarCargo/', views.guardarCargo, name='guardarCargo'),  
    path('EditarCargo/<id>/', views.Mformnualrio, name='EditarCargo'),  
    path('PECargo/', views.ProcesarEdicionCargo, name='PECargo'),  
    path('ECargo/<id>/', views.eliminarCargo, name='ECargo'),  
]  
|
```

- Esto hace que se pueda acceder a la función eliminarCargo enviando el ID del cargo en la URL.

### Enlace para eliminar en el HTML del listado (inicio.html o donde muestras la tabla)

Dentro del bucle for que muestra los cargos, pon este botón:

```
<a href="#" onclick="confirmarEliminacion('{% url 'ECargo' cargoTemporal.id %}')">  
    <i class="fa fa-trash"></i> Eliminar  
</a>
```

```
jones > Empresas > templates > inicio.html > table.table.table-bordered.table-striped.table-hover > tbody > tr > td  
<table class="table table-bordered table-striped table-hover">  
    <tbody>  
        {% for cargoTemporal in cargo %}  
            <tr>  
                <td>{{cargoTemporal.id}}</td>  
                <td>{{cargoTemporal.nombres}}</td>  
                <td>{{cargoTemporal.funciones}}</td>  
                <td>{{cargoTemporal.horario}}</td>  
                <td>{{cargoTemporal.requisitos}}</td>  
                <td>{{cargoTemporal.sueldo}}</td>  
                <td>  
                    <a class="btn btn-warning" href="{% url 'EditarCargo' cargoTemporal.id %}"><i class="fa fa-pencil"></i> Editar</a>  
                    <a href="#" onclick="confirmarEliminacion('{% url 'ECargo' cargoTemporal.id %}')"><i class="fa fa-trash"></i> Eliminar</a>  
                </td>  
            </tr>  
        {% endfor %}  
    </tbody>
```



- Esto crea un botón que al hacer clic lanza un mensaje de confirmación antes de eliminar.

### Script para confirmar eliminación con SweetAlert

Pon este código al final del HTML (dentro del bloque { % block contenido % }):

```
<script>

function confirmarEliminacion(url) {

  Swal.fire({
    title: "¿Estás seguro?",
    text: "¡No podrás revertir esto!",
    icon: "warning",
    showCancelButton: true,
    confirmButtonColor: "#3085d6",
    cancelButtonColor: "#d33",
    confirmButtonText: "Sí, eliminar"
  }).then((result) => {
    if (result.isConfirmed) {
      window.location.href = url; // Redirige a la URL de eliminación
    }
  });
}

</script>
```

Flujo



## Contenido del encabezado aquí

### Listado de Cargos

[Nuevo Cargo](#)

ID	NOMBRE	FUNCIONES	HORARIOS	SUELDO	ACCIONES	REQUISITOS
21	1	1	1	1.00	<a href="#">Editar</a>	<a href="#">Eliminar</a>

## Contenido del pie de página aquí

```
#Configuración de rutas específicas de la App Empresas
from django.urls import path
from . import views
urlpatterns = [
    path('', views.inicio, name='inicio'),
    path('nuevoCargo/', views.nuevoCargo, name='nuevoCargo'),
    path('guardarCargo/', views.guardarCargo, name='guardarCargo'),
    path('EditarCargo/<id>', views.Mformualario, name='EditarCargo'),
    path('PECargo/', views.ProcesarEdicionCargo, name='PECargo'),
    path('ECargo/<id>', views.eliminarCargo, name='ECargo'),
]
```

Tabnine | Edit | Test | Explain | Document

```
def eliminarCargo(request,id):
    CargoEliminar=Cargo.objects.get(id=id)
    CargoEliminar.delete()
    messages.success(request,"Eliminado exitosamente")

    return redirect('/')
# Mostrar Formualrio
```

```
<a href="#" onclick="confirmarEliminacion('{% url 'ECargo' cargoTemporal.id %}')">
>

<i class="fa fa-trash"></i>Eliminar
</a>
```

Con el onclick se hace una la opción de aceptar o cancelar

## Contenido del encabezado aquí

### Listado de Cargos

ID	NOMBRE	FUNCIONES	HORARIOS	SUELDO	ACCIONES	REQUISITOS
21	1	12	1	1.00	<a href="#">Editar</a>	<a href="#">Eliminar</a>

## Contenido del pie de página aquí

### Listado de Cargos

ID	NOMBRE	FUNCIONES	HORARIOS	SUELDO	ACCIONES	REQUISITOS
21	1	12	1	1.00	<a href="#">Editar</a>	<a href="#">Eliminar</a>

¿Estás seguro?

¡No podrás revertir esto!

[Sí, eliminar](#)

[Cancelar](#)



Mensaje

Eliminado exitosamen

OK



## 14 ¿Cómo relacionar dos tablas con ForeignKey?

Cuando trabajamos en proyectos Django grandes o divididos en varias aplicaciones (apps), cada app tiene sus propios modelos que representan tablas en la base de datos. Muchas veces, es necesario relacionar dos tablas que están definidas en apps distintas. Por ejemplo, el modelo Cargo está en la app Empleos, y queremos crear otro modelo llamado Empleado en otra app que esté vinculado al Cargo.

Para lograr esto, antes de crear la relación con ForeignKey debes importar el modelo que quieras usar como referencia, en este caso Cargo, para que Django sepa a qué tabla nos referimos.

### 💡 ¿Qué es ForeignKey?

ForeignKey es un tipo de campo en Django que sirve para **crear una relación entre dos modelos (tablas)**.

- Conecta una **tabla secundaria** con una **tabla principal**.
- Permite que una fila de la tabla secundaria **apunte** a una fila de la tabla principal.

## 2. Situación práctica

Tienes dos apps en tu proyecto Django:

- **App 1:** Empleos  
Aquí está el modelo Cargo que ya creaste.
- **App 2:** Personas  
Aquí quieres crear el modelo Empleado que se va a relacionar con Cargo.

## Modelo ya creado

Tú ya tienes este modelo creado:

```
from django.db import models

class Cargo(models.Model):
    nombres = models.CharField(max_length=100)
    funciones = models.TextField()
    horario = models.CharField(max_length=500)
    requisitos = models.TextField()
    sueldo = models.DecimalField(decimal_places=2, max_digits=10)

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

💡 Este modelo es la **tabla principal**. Guarda los datos de cada cargo (por ejemplo: "Contador", "Analista", etc.).



## **Crear el modelo Empleado en la app Personas**

Ahora creamos otro modelo en la app Personas, en este archivo:

Aplicaciones/Personas/models.py

---

## **Importar el modelo Cargo de la app Empleos**

Para usar el modelo Cargo en otro archivo, hay que importarlo primero:

```
from Aplicaciones.Empleos.models import Cargo
```

Esto es **muy importante**, porque Django necesita saber a qué modelo exacto te refieres.

---

## **Definir el modelo Empleado con ForeignKey a Cargo**

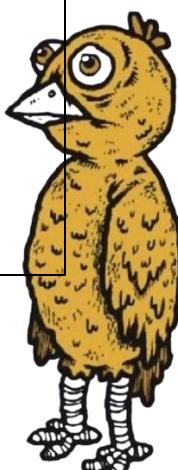
Después de la importación, creamos la clase Empleado y ponemos el campo cargo como ForeignKey a Cargo:

```
from Aplicaciones.Empresas.models import Cargo # Importamos Cargo desde otra app

# Create your models here.

class Empleado(models.Model):
    nombres = models.CharField(max_length=100)
    cedula = models.CharField(max_length=10)
    cargo = models.ForeignKey(Cargo, on_delete=models.CASCADE) # Relación con Cargo

    def __str__(self):
        return f'{self.nombres} - Cargo: {self.cargo.nombres}'
```



```

models.py ...\\Empresas tests.py models.py ...\\Empleos X views.py

Aplicaciones > Empleos > 🐍 models.py > ...
1  from django.db import models
2  from Aplicaciones.Empleos.models import Cargo # Importamos Cargo desde otra app
3
4  # Create your models here.
5
6  class Empleado(models.Model):
7      nombres = models.CharField(max_length=100)
8      cedula = models.CharField(max_length=10)
9      cargo = models.ForeignKey(Cargo, on_delete=models.CASCADE) # Relación con Cargo
10
11     Tabnine | Edit | Test | Explain | Document
12     def __str__(self):
13         return f"{self.nombres} - Cargo: {self.cargo.nombres}"
14

```

## Explicación del campo ForeignKey

`cargo = models.ForeignKey(Cargo, on_delete=models.CASCADE)`

- `Cargo`: es el modelo al que nos relacionamos (el modelo padre).
  - `on_delete=models.CASCADE`: significa que si borras un `Cargo`, se borran también todos los `Empleado` que tengan ese cargo.
- (Otras opciones para `on_delete` pueden ser: `SET_NULL`, `PROTECT`, etc.)

---

## Registrar las apps en settings.py

Para que Django sepa de tus apps, debes tenerlas registradas en el archivo `settings.py` de tu proyecto:

```

INSTALLED_APPS = [
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'Aplicaciones.Empleos',
]

```

```

31  # Application definition
32
33  INSTALLED_APPS = [
34      'django.contrib.admin',
35      'django.contrib.auth',
36      'django.contrib.contenttypes',
37      'django.contrib.sessions',
38      'django.contrib.messages',
39      'django.contrib.staticfiles',
40      'Aplicaciones.Empleos',
41      'Aplicaciones.Empresas'
42
43  ]
44

```

---

## Cómo Django crea la relación en la base de datos

- Django crea una columna cargo\_id en la tabla Empleado.
- Esa columna guarda el ID del Cargo relacionado.

Por ejemplo, si en la tabla Cargo tienes:

**id nombres**

1 Programador

Y en la tabla Empleado:

**id nombres cedula cargo\_id**

1 Ana 0123456789 1

Significa que Ana es Programadora.

---

## 5. Alternativa: usar ForeignKey con string en vez de importación directa

Si por alguna razón no quieres o no puedes hacer el import, puedes poner el nombre del modelo como string:

```
cargo = models.ForeignKey('Empleos.Cargo', on_delete=models.CASCADE)
```

- 'Empleos.Cargo' es el nombre de la app + modelo.
  - Es útil cuando hay relaciones circulares o importaciones complicadas.
- 

## 6. Pasos finales: Migraciones

Después de definir o modificar los modelos, en la terminal de tu proyecto corre estos comandos:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

Así Django crea las tablas y relaciones en la base de datos.

---



## 1 .Crea el archivo urls.py y la carpeta template en la aplicación

Después de definir o modificar los modelos, en la terminal de tu proyecto corre estos comandos:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

Así Django crea las tablas y relaciones en la base de datos.

### Antes de usar ForeignKey (Relaciones entre tablas)

#### Importante:

Antes de crear relaciones entre tablas con ForeignKey, se recomienda tener los CRUD básicos (Crear, Leer, Actualizar, Eliminar) funcionando para cada tabla por separado. Esto ayuda a que entiendas cómo funciona cada modelo individual y luego puedas unirlos sin problemas.

Por ejemplo:

- Primero crea el CRUD completo para la tabla Cargo (que sería como el catálogo o referencia de cargos).
- Luego crea el CRUD para la tabla Empleado, pero sin relación al cargo (solo con campos simples).

Una vez que tienes estos CRUD funcionando por separado, pasas a relacionar las tablas usando ForeignKey para que el modelo Empleado tenga un campo que apunte a un Cargo.

Cuando usamos una relación con ForeignKey para enlazar dos tablas, como por ejemplo que el modelo Empleado tenga un campo que apunte a un Cargo, necesitamos que esa relación se refleje también en la vista (la parte donde se muestran los datos y se controlan las peticiones).

¿Por qué?

1. La vista es la que conecta los datos con la interfaz: Aunque ya definimos la relación en el modelo, la vista es la encargada de obtener los datos relacionados (por ejemplo, obtener el cargo de cada empleado) y enviarlos al template para que el usuario pueda ver la información completa.
2. La vista controla qué datos llegan al usuario: Si no modificamos la vista para incluir la relación, solo veremos los datos simples del empleado, pero no su cargo. Por eso, en la vista debemos pedir que también se cargue la información del cargo relacionada.
3. La vista facilita el manejo de formularios con ForeignKey: Cuando creamos o editamos un empleado, en la vista debemos asegurarnos de que el formulario muestre las opciones de cargos disponibles para que el usuario pueda elegir uno. Esto se hace en la vista pasando esa lista de cargos al template.



4. Sin ajustar la vista, la relación no se puede mostrar ni modificar correctamente:  
Porque el modelo solo define la estructura de datos, pero la lógica para mostrar o interactuar con esa relación está en la vista

### Antes (sin relación entre Empleado y Cargo)

```
from django.shortcuts import render, redirect
from django.contrib import messages
from .models import Empleado # Solo Empleado, sin Cargo

# Mostrar todos los empleados
def listaEmpleados(request):
    empleados = Empleado.objects.all()
    return render(request, "Inicio.html", {'empleados': empleados})

# Formulario para crear un nuevo empleado (sin lista de cargos)
def nuevoEmpleado(request):
    return render(request, "Nuevo.html")

# Guardar nuevo empleado sin cargo
def guardarEmpleado(request):
    nombres = request.POST["nombres"]
    cedula = request.POST["cedula"]

    Empleado.objects.create(nombres=nombres, cedula=cedula)
    messages.success(request, "Empleado registrado correctamente")
    return redirect('listaEmpleados')

# Mostrar formulario de edición (sin lista de cargos)
def editarEmpleado(request, id):
    empleado = Empleado.objects.get(id=id)
    return render(request, "Editar.html", {'empleado': empleado})

# Procesar edición sin actualizar cargo
def actualizarEmpleado(request):
    id = request.POST["id"]
    empleado = Empleado.objects.get(id=id)

    empleado.nombres = request.POST["nombres"]
    empleado.cedula = request.POST["cedula"]

    empleado.save()
    messages.success(request, "Empleado actualizado correctamente")
    return redirect('listaEmpleados')

# Eliminar empleado
def eliminarEmpleado(request, id):
    empleado = Empleado.objects.get(id=id)
    empleado.delete()
    messages.success(request, "Empleado eliminado correctamente")
    return redirect('listaEmpleados')
```



## Después (con relación foreign key a Cargo)

```
from django.shortcuts import render, redirect
from django.contrib import messages

# IMPORTAMOS LOS DOS MODELOS PARA LA RELACIÓN
from .models import Empleado, Cargo # <-- NUEVO: Importar Cargo para la relación

# Mostrar todos los empleados
def listaEmpleados(request):
    empleados = Empleado.objects.all()
    return render(request, "Inicio.html", {'empleados': empleados})

# Formulario para crear un nuevo empleado
def nuevoEmpleado(request):
    cargos = Cargo.objects.all() # <-- NUEVO: Obtener lista de cargos para mostrar en el formulario
    return render(request, "Nuevo.html", {'cargos': cargos})

# Guardar nuevo empleado
def guardarEmpleado(request):
    nombres = request.POST["nombres"]
    cedula = request.POST["cedula"]

    # <-- NUEVO: Obtener el id del cargo seleccionado y traer el objeto Cargo
    id_cargo = request.POST["cargo"]
    cargo = Cargo.objects.get(id=id_cargo)

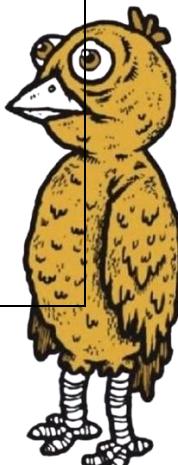
    # <-- NUEVO: Guardar el empleado con el cargo relacionado
    Empleado.objects.create(nombres=nombres, cedula=cedula, cargo=cargo)
    messages.success(request, "Empleado registrado correctamente")
    return redirect('listaEmpleados')

# Mostrar formulario de edición
def editarEmpleado(request, id):
    empleado = Empleado.objects.get(id=id)
    cargos = Cargo.objects.all() # <-- NUEVO: Traer cargos para mostrar en el select
    return render(request, "Editar.html", {'empleado': empleado, 'cargos': cargos})

# Procesar edición
def actualizarEmpleado(request):
    id = request.POST["id"]
    empleado = Empleado.objects.get(id=id)

    empleado.nombres = request.POST["nombres"]
    empleado.cedula = request.POST["cedula"]

    # <-- NUEVO: Actualizar la relación con el cargo seleccionado
    id_cargo = request.POST["cargo"]
    empleado.cargo = Cargo.objects.get(id=id_cargo)
```



```

    empleado.save()
    messages.success(request, "Empleado actualizado correctamente")
    return redirect('listaEmpleados')

# Eliminar empleado
def eliminarEmpleado(request, id):
    empleado = Empleado.objects.get(id=id)
    empleado.delete()
    messages.success(request, "Empleado eliminado correctamente")
    return redirect('listaEmpleados')

```

### Nuevo empleado Antes (sin relación, solo inputs):

```

{% extends "plantilla.html" %}
{% block contenido %}
<h2>Nuevo Empleado</h2>
<form action="{% url 'guardarEmpleado' %}" method="post">
    {% csrf_token %}
    <label>Nombres:</label>
    <input class="form-control" type="text" name="nombres" required>

    <label>Cédula:</label>
    <input class="form-control" type="text" name="cedula" required>

    <!-- ANTES: Sin relación, solo input simple para cargo -->
    <label>Cargo:</label>
    <input class="form-control" type="text" name="cargo" required>

    <br>
    <button class="btn btn-success" type="submit">Guardar</button>
    <a href="{% url 'listaEmpleados' %}" class="btn btn-secondary">Cancelar</a>
</form>
{% endblock %}

```

### Nuevo empleado Despues (con relación, usando select para elegir el cargo):

```

{% extends "plantilla.html" %}
{% block contenido %}
<h2>Nuevo Empleado</h2>
<form action="{% url 'guardarEmpleado' %}" method="post">
    {% csrf_token %}
    <label>Nombres:</label>
    <input class="form-control" type="text" name="nombres" required>

    <label>Cédula:</label>
    <input class="form-control" type="text" name="cedula" required>

    <!-- DESPUÉS: Se agrega select para elegir el cargo, mostrando los cargos disponibles -->
    <label>Cargo:</label>

```



```

<select class="form-control" name="cargo" required>
    { % for c in cargos % }
        <option value="{{ c.id }}>{{ c.nombres }}</option>
    { % endfor %}
</select>

<br>
<button class="btn btn-success" type="submit">Guardar</button>
<a href="{% url 'listaEmpleados' %}" class="btn btn-secondary">Cancelar</a>
</form>
{ % endblock %}

```

## 🎨 PASO: Transformar un campo sin relación a un campo con relación (ForeignKey)

Cuando estamos creando un formulario en Django para registrar empleados, al inicio podríamos tener algo muy básico, como esto:

### 📄 Código original (antes del cambio)

### En `views.py`:

```

Formulario para crear un nuevo empleado (sin lista de cargos)

def nuevoEmpleado(request):
    return render(request, "Enuevo.html")

```

Esta función simplemente carga el formulario vacío. No envía datos extras como listas, solo muestra el HTML

### En `Enuevo.html`:

```

<!-- ANTES: Sin relación, solo input simple para cargo -->

<label>Cargo:</label>
<input class="form-control" type="text" name="cargo" required>
<br>

```

Este campo es un \*\*input de texto simple\*\*. Eso significa que:

- El usuario tiene que escribir a mano el nombre del cargo.
- Es fácil equivocarse, escribir mal o poner un cargo que no existe.
- No hay una relación real con el modelo `Cargo`, solo se guarda un texto.



> **X** Este método \*\*no es bueno\*\* si ya tienes una tabla de cargos en la base de datos, porque no aprovechas esa relación. Además, no puedes hacer búsquedas ni validaciones fácilmente.

## **✓** Ahora, ¿qué se debe hacer?

Vamos a modificar tanto la \*\*vista\*\* como el \*\*formulario HTML\*\*, para que el campo de "cargo" esté relacionado directamente con los registros de la tabla `Cargo` en la base de datos. Esto hace que todo sea más ordenado, correcto y profesional.

## **✓** NUEVO CÓDIGO: Con relación (ForeignKey) al modelo `Cargo`

### En `views.py`, reemplaza la función `nuevoEmpleado` así:

```
# NUEVO: Formulario con lista de cargos relacionada al modelo

def nuevoEmpleado(request):
    cargos = Cargo.objects.all()
    return render(request, "Enuevo.html", {'cargos': cargos})
```

#### **def nuevoEmpleado(request):**

- 👉 Esta línea empieza una nueva función llamada nuevoEmpleado.
- Esta función se va a ejecutar cuando alguien quiera **ver la página para registrar un nuevo empleado**.
  - Django se encarga de llamar a esta función automáticamente cuando tú configuras la URL correspondiente.

#### **cargos = Cargo.objects.all()**

👉 Esta línea busca **todos los cargos** que hay guardados en la base de datos y los guarda en la variable cargos.

- Cargo es el modelo que contiene los nombres de los diferentes cargos disponibles en la empresa, como por ejemplo: "Administrador", "Secretaria", "Técnico", etc.
- .objects.all() es una función que **pide todos los registros** de ese modelo.
- El resultado será como una lista, donde cada elemento es un cargo con su id y su nombre.

💡 Esta lista es esencial porque será usada para llenar automáticamente el menú desplegable del formulario. Así no tienes que escribir los cargos a mano.

#### **return render(request, "Enuevo.html", {'cargos': cargos})**

👉 Esta línea dice:

"Muéstrale al usuario la plantilla Enuevo.html, y además envíale la lista de cargos con el nombre 'cargos'".

- render es una función que combina una plantilla HTML con datos que tú le pasas.
- "Enuevo.html" es el archivo que va a mostrar el formulario.
- {'cargos': cargos} es un diccionario que **envía la variable cargos al HTML**. En otras palabras:



- En Python se llama cargos.
- En el HTML se va a poder usar esa misma variable cargos gracias a esta línea.

### 🔎 ¿Qué hace esto?

- Recupera \*\*todos los cargos registrados\*\* en la base de datos usando `Cargo.objects.all()`.
- Envía esa lista de cargos al archivo HTML usando un diccionario: `{'cargos': cargos}`.
- Gracias a esto, el formulario podrá mostrar los cargos como un \*\*menú desplegable automático\*\*, en lugar de un campo de texto.

### En `Enuevo.html`, reemplaza el campo de texto por esto:

```
<!-- NUEVO: Select que muestra cargos reales de la base -->

<label>Cargo:</label>
<select class="form-control" name="cargo" required>
    { % for c in cargos %}
        <option value="{{ c.id }}>{{ c.nombres }}</option>
    { % endfor %}
</select>
<br>
```

**<label>Cargo:</label>**

- 👉 Muestra el texto "**Cargo:**" justo arriba del menú desplegable.
- Esto le dice al usuario que debe escoger el cargo del nuevo empleado.
  - No tiene lógica de programación, solo es una etiqueta visual.

**<select class="form-control" name="cargo" required>**

- 👉 Crea un menú desplegable que muestra todos los cargos que vienen desde la base de datos.
- select: Este es el nombre de la etiqueta HTML que se usa para crear un menú desplegable.
  - class="form-control": Esta clase viene de Bootstrap. Sirve para que el menú se vea bonito y más ancho. No es obligatorio, pero ayuda al diseño.
  - name="cargo": Este nombre es **clave**.
    - Cuando el formulario se envía, este será el **nombre del campo que llega al servidor**.
    - En Django, vas a recuperar este dato usando request.POST["cargo"].
  - required: Hace que el campo sea obligatorio. El usuario no puede enviar el formulario sin elegir un cargo.

**{% for c in cargos %}**

- 👉 Empieza un bucle que se repite una vez por cada cargo que haya en la lista cargos.
- c es una variable temporal. En cada vuelta del bucle, representa **un solo cargo**.



- Por ejemplo, si hay 3 cargos: Administrador, Contador y Técnico, este bloque se ejecutará 3 veces.
- Este bucle se ejecuta en el servidor (gracias a Django) y el resultado es HTML plano que llega al navegador.

```
<option value="{{ c.id }}" {% if c.id == empleado.cargo.id %}selected{% endif %}>{{ c.nombres }}</option>
```

👉 Esta línea crea una **opción dentro del menú desplegable**, o sea, una fila que el usuario podrá seleccionar.

Vamos por partes:

**value="{{ c.id }}"**

- Esto le dice al formulario que, si el usuario elige esta opción, se va a enviar el **ID del cargo** (un número).
- Este ID es lo que se usará en Django para buscar el cargo correcto y asignárselo al nuevo empleado.

🧠 Ejemplo: si el cargo “Administrador” tiene ID 3, entonces esta opción será:

```
<option value="3">Administrador</option>
```

**{% if c.id == empleado.cargo.id %}selected{% endif %}**

👉 Esto es un **condicional** que se usa cuando estás editando un empleado (no creando uno nuevo).

- Compara el ID del cargo del bucle (c.id) con el cargo actual del empleado (empleado.cargo.id).
- Si los dos son iguales, se agrega la palabra selected.
- selected hace que esa opción aparezca **marcada por defecto** en el menú.

🧠 Ejemplo:

Si el empleado que estás editando tiene el cargo “Contador”, y el ID de “Contador” es 5, entonces esta línea se convierte en:

```
<option value="5" selected>Contador</option>
```

Eso hace que aparezca marcada como predeterminada.

**{{ c.nombres }}**

👉 Aquí se muestra el **nombre del cargo** que el usuario va a ver en la pantalla.

- Aunque el valor que se envía sea el número (ID), lo que se ve es el texto descriptivo del cargo.
- Ejemplo:

```
<option value="3">Administrador</option>
```

**{% endfor %}**

👉 Esta línea cierra el bucle for.

Le dice a Django que ya terminaste de crear las opciones del menú.

⬅ **END** ¿Qué se logra con todo esto?

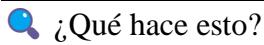
Gracias a este bloque:

- El formulario muestra **todos los cargos** de la base de datos.
- Se guarda el **ID del cargo seleccionado**, lo que permite relacionar un empleado con su cargo.



- Si el formulario es para **editar un empleado**, se marca automáticamente su cargo actual.

¿Quieres que ahora pase a explicar cómo se recibe este cargo en la vista guardarEmpleado y cómo se relaciona con el modelo Empleado? Puedo hacerlo igual de detallado, MBUE.



- Muestra un menú desplegable (`<select>`), no un campo de texto.
- La lista de opciones se llena automáticamente con los datos de la base gracias al bucle `'{% for c in cargos %}'`.
- Cada opción del menú tiene como valor el `id` del cargo, pero muestra el nombre (`c.nombres`).
- Cuando el formulario se envía, se manda el `id` del cargo que fue seleccionado.
- Este `id` luego se puede usar en la vista que guarda el empleado para hacer la relación correctamente.

#### Editar Empleado Antes (sin relación, solo inputs):

```
{% extends "plantilla.html" %}
{% block contenido %}
<h2>Editar Empleado</h2>
<form action="{% url 'actualizarEmpleado' %}" method="post">
  {% csrf_token %}
  <input type="hidden" name="id" value="{{ empleado.id }}>

  <label>Nombres:</label>
  <input class="form-control" type="text" name="nombres" value="{{ empleado.nombres }}" required>

  <label>Cédula:</label>
  <input class="form-control" type="text" name="cedula" value="{{ empleado.cedula }}" required>

  <!-- ANTES: Sin relación, solo input simple para cargo -->
  <label>Cargo:</label>
  <input class="form-control" type="text" name="cargo" value="{{ empleado.cargo }}" required>

  <br>
  <button class="btn btn-success" type="submit">Actualizar</button>
  <a href="{% url 'listaEmpleados' %}" class="btn btn-secondary">Cancelar</a>
</form>
{% endblock %}
```



## Editar Después (con relación, usando select para elegir y mostrar el cargo seleccionado):

```
{% extends "plantilla.html" %}  
{% block contenido %}  
<h2>Editar Empleado</h2>  
<form action="{% url 'actualizarEmpleado' %}" method="post">  
    {% csrf_token %}  
    <input type="hidden" name="id" value="{{ empleado.id }}>  
  
    <label>Nombres:</label>  
    <input class="form-control" type="text" name="nombres" value="{{ empleado.nombres }}" required>  
  
    <label>Cédula:</label>  
    <input class="form-control" type="text" name="cedula" value="{{ empleado.cedula }}" required>  
  
    <!-- DESPUÉS: Se cambia input por select, mostrando todos los cargos -->  
    <label>Cargo:</label>  
    <select class="form-control" name="cargo" required>  
        {% for c in cargos %}  
            <!-- Se marca 'selected' el cargo que tiene el empleado -->  
            <option value="{{ c.id }}" {% if c.id == empleado.cargo.id %}selected{% endif %}>{{ c.nombres }}</option>  
        {% endfor %}  
    </select>  
  
<br>  
    <button class="btn btn-success" type="submit">Actualizar</button>  
    <a href="{% url 'listaEmpleados' %}" class="btn btn-secondary">Cancelar</a>  
</form>  
{% endblock %}
```

### ● CÓDIGO ORIGINAL — Antes de usar relaciones

#### En Editar.html:

```
<!-- ANTES: Sin relación, solo input simple para cargo -->  
  
<label>Cargo:</label>  
<input class="form-control" type="text" name="cargo" value="{{ empleado.cargo }}" required>
```



## ?

### ¿Qué hace este campo?

- Muestra un campo de texto con el valor actual del cargo del empleado.
- El valor viene de empleado.cargo, que sí es un objeto relacionado, pero se muestra solo como texto.
- El problema es que al modificarlo, el usuario puede escribir mal el nombre del cargo o inventarse uno que no exista.
- Al guardar, no se hace una relación real con la tabla Cargo, sino que se intenta guardar como texto.

✗ Esto **no aprovecha la relación** que ya existe en el modelo entre Empleado y Cargo.

**NEW** En `Editar.html`, se reemplaza el `input` por esto:

```
<!-- DESPUÉS: Se cambia input por select, mostrando todos los cargos -->
<label>Cargo:</label>
<select class="form-control" name="cargo" required>
    { % for c in cargos %}
        <!-- Se marca 'selected' el cargo que tiene el empleado -->
        <option value="{{ c.id }}" { % if c.id == empleado.cargo.id % }selected{ % endif % }>{{ c.nombres }}</option>
    { % endfor %}
</select>
```

<label>Cargo:</label>

- Esto crea una etiqueta o texto que se muestra antes del campo para que el usuario sepa qué debe seleccionar.
- “Cargo” es el nombre que le damos al campo, indica que el usuario va a elegir un puesto o función dentro de la empresa.
- Es simplemente un texto descriptivo que se ve en la página.

<select class="form-control" name="cargo" required>

- `<select>` es un elemento HTML que crea un menú desplegable.
- `class="form-control"` aplica estilos (generalmente de Bootstrap) para que el campo se vea bien, con bordes, tamaño adecuado y espaciado.
- `name="cargo"` define el nombre con el que este dato se enviará al servidor cuando el formulario se envíe.
  - Esto es importante porque el servidor usará ese nombre para acceder al valor que eligió el usuario.



- Por ejemplo, cuando el formulario llegue al backend, podrás obtener el cargo con `request.POST["cargo"]` en Django.
- `required` obliga al usuario a seleccionar un cargo antes de enviar el formulario, no permite que quede vacío.

{% for c in cargos %}

- Esto es una estructura de Django llamada plantilla o template tag.
- Inicia un ciclo que repite el código que está dentro tantas veces como elementos tenga la variable `cargos`.
- `cargos` es una lista o conjunto de objetos que viene de la vista en Python (del archivo `views.py`).
  - En la vista, previamente, se hizo algo así: `cargos = Cargo.objects.all()` para traer todos los cargos desde la base de datos.
- Por cada cargo dentro de esta lista, Django crea una variable temporal llamada `c` que representa ese cargo individual.
- Así podemos usar `c` para mostrar sus datos dentro del ciclo, como su `id` o nombre.

```
<option value="{{ c.id }}" {% if c.id == empleado.cargo.id %}selected{% endif %}>{{ c.nombres }}</option>
```

Esta línea es la que genera cada opción dentro del menú desplegable y aquí hay varias cosas para entender:

- `<option>` es una etiqueta HTML que representa una opción que puede seleccionar el usuario dentro del `<select>`.
- `value="{{ c.id }}":`
  - Aquí `value` es el dato que se enviará al servidor si el usuario escoge esta opción.
  - Usamos `{{ c.id }}` para colocar el ID único del cargo actual que está en la base de datos.
  - Cada cargo tiene un ID (un número único que identifica ese cargo). Por ejemplo, “Gerente” puede tener ID 1, “Secretario” ID 2, y así sucesivamente.
  - Esto es muy importante porque el servidor necesita el ID para saber qué cargo específico se está seleccionando, no solo el nombre.
- `{% if c.id == empleado.cargo.id %}selected{% endif %}:`
  - Esto es una condición para marcar la opción que corresponde al cargo que ya tiene asignado el empleado.
  - `empleado` es el objeto que contiene la información del empleado que estamos editando.
  - `empleado.cargo` es la relación que indica el cargo asignado a ese empleado.
  - `empleado.cargo.id` es el ID del cargo asignado.
  - Comparamos el ID del cargo actual del ciclo (`c.id`) con el ID del cargo del empleado (`empleado.cargo.id`).
  - Si son iguales, significa que esta opción es la que debe estar seleccionada por defecto cuando se abre el formulario.
  - En ese caso, se agrega la palabra `selected`, que es un atributo HTML que hace que esa opción aparezca marcada.
- `{{ c.nombres }}:`



- Muestra el nombre del cargo en el menú para que el usuario pueda verlo y elegirlo.
- Por ejemplo: “Gerente”, “Secretario”, “Vendedor”.
- </option> cierra la opción.

¿De dónde viene empleado y cargos?

- En la vista Python que llama a este formulario, hay algo así:

```
def editarEmpleado(request, id):
    empleado = Empleado.objects.get(id=id) # Trae el empleado que queremos editar
    cargos = Cargo.objects.all() # Trae todos los cargos disponibles
    return render(request, "Editar.html", {'empleado': empleado, 'cargos': cargos})
```

- Aquí, empleado y cargos se pasan a la plantilla como variables para que puedan usarse en el HTML.
- Por eso podemos hacer empleado.cargo.id para acceder al cargo asignado al empleado y hacer la comparación.

{% endfor %}

- Esto indica que termina el ciclo for.
- Con esto se aseguran que se generen todas las opciones necesarias, una por cada cargo.

</select>

- Finalmente se cierra el menú desplegable, indicando que ya no hay más opciones.

Resumen funcional

- Este fragmento crea un menú donde el usuario puede elegir un cargo para el empleado.
- El menú muestra todos los cargos existentes.
- El cargo que el empleado ya tiene aparece preseleccionado.
- Cuando se envía el formulario, el servidor recibe el ID del cargo elegido para guardarlo correctamente en la base de datos.

Si quieras, te puedo explicar también cómo funciona la parte del backend para recibir ese ID y guardar o actualizar el empleado con ese cargo. ¿Quieres?

**NEW En views.py, modifica la función editarEmpleado para que envíe los cargos:**

```
# Mostrar formulario de edición
def editarEmpleado(request, id):
    empleado = Empleado.objects.get(id=id)
    cargos = Cargo.objects.all() # <-- NUEVO: Traer todos los cargos de la base
    return render(request, "Editar.html", {'empleado': empleado, 'cargos': cargos})
```



### Explicación de cada línea:

- empleado = Empleado.objects.get(id=id)  
→ Trae el empleado que se va a editar, usando el ID que llega por la URL.
- cargos = Cargo.objects.all()  
→ Obtiene todos los cargos de la tabla Cargo para mostrarlos en el <select>.
- return render(...)  
→ Envía dos cosas al HTML:
  - empleado: para llenar los campos actuales del formulario.
  - cargos: para crear el menú desplegable con todos los cargos.

## SECCIÓN DEL MANUAL – GUARDAR UN EMPLEADO CON SU CARGO

### CÓDIGO COMPLETO:

```
# Guardar nuevo empleado con cargo
def guardarEmpleado(request):
    nombres = request.POST["nombres"]
    cedula = request.POST["cedula"]

    id_cargo = request.POST["cargo"]
    cargo = Cargo.objects.get(id=id_cargo)

    Empleado.objects.create(nombres=nombres, cedula=cedula, cargo=cargo)

    messages.success(request, "Empleado registrado correctamente")
    return redirect('listaEmpleados')
```

### **def guardarEmpleado(request):**

- **def**: palabra clave para **definir una función**.



- **guardarEmpleado**: nombre personalizado que damos a esta función. Tú puedes ponerle cualquier nombre, pero en este caso se llama así porque su función es **guardar un empleado**.
- (**request**): los paréntesis indican que esta función recibe un argumento, que en este caso es **request**.
  - request representa todo lo que el usuario envió desde el navegador: los datos que escribió, los botones que presionó, etc.

**nombres = request.POST["nombres"]**

- **request.POST**: es un **diccionario** (una estructura que guarda datos con nombre y valor) que contiene toda la información enviada por el formulario HTML cuando el método es POST.
- **["nombres"]**: se está buscando el dato que se envió con el atributo `name="nombres"` del formulario.
- **nombres = ...**: este valor se guarda en la variable llamada nombres.

👉 **Ejemplo real:** si el formulario tiene esto:

```
<input name="nombres" value="Juan Pérez">
```

Entonces, esta línea va a guardar el texto "Juan Pérez" en la variable nombres.

**cedula = request.POST["cedula"]**

- Hace lo mismo que la línea anterior, pero esta vez con el valor del campo cedula.
- Busca lo que el usuario escribió en el input con `name="cedula"` y lo guarda en la variable cedula.

👉 **Ejemplo:** Si el input era:

```
<input name="cedula" value="1234567890">
```

Entonces la variable cedula ahora tiene ese valor.

**id\_cargo = request.POST["cargo"]**

- Aquí se obtiene el **ID del cargo que el usuario seleccionó** en el menú desplegable (<select>).
- En el formulario, hay un select con `name="cargo"`, y cada opción tiene como value el ID del cargo (por ejemplo: 1, 2, 3).
- Este número (en forma de texto) se guarda en la variable id\_cargo.



👉 Ejemplo real del HTML:

```
<select name="cargo">  
    <option value="1">Administrador</option>  
    <option value="2">Vendedor</option>  
</select>
```

Si el usuario elige “Vendedor”, se guarda “2” en la variable `id_cargo`.

**`cargo = Cargo.objects.get(id=id_cargo)`**

- **Cargo**: este es el **modelo** que representa la tabla de cargos en la base de datos.
- **.objects**: accede al administrador de objetos del modelo, que nos permite consultar la base de datos.
- **.get(id=id\_cargo)**: busca en la tabla un cargo que tenga ese id. Devuelve **el objeto completo**, no solo el número.

👉

Ejemplo:

Si `id_cargo = "2"`, esta línea busca en la base de datos el cargo con ID 2 y lo guarda en la variable `cargo`.

**`Empleado.objects.create(nombres=nombres, cedula=cedula, cargo=cargo)`**

- Esta línea **crea un nuevo empleado en la base de datos**.
- **Empleado**: es el modelo que representa la tabla de empleados.
- **.objects.create(...)**: crea y guarda directamente un nuevo registro con los datos que le das.
- **nombres=nombres**: usa el valor que estaba guardado en la variable `nombres` (por ejemplo: "Juan Pérez").
- **cedula=cedula**: usa la cédula que el usuario escribió (por ejemplo: "1234567890").
- **cargo=cargo**: aquí no se pone el número, sino el objeto cargo completo que se obtuvo antes (por ejemplo: el cargo "Vendedor").

**`messages.success(request, "Empleado registrado correctamente")`**

- **messages**: es un sistema de Django para enviar mensajes al usuario.
- **.success(...)**: indica que este es un mensaje de tipo “éxito” (puede ser también info, warning o error).



- **request**: se lo pasa porque el mensaje debe ir al mismo usuario que hizo la solicitud.
- "**Empleado registrado correctamente**

```
return redirect('listaEmpleados')
```

- Esta línea **redirige** al usuario a otra vista.
- **redirect(...)**: es una función de Django que envía al usuario a una URL definida.
- '**listaEmpleados**

## SECCIÓN DEL MANUAL – ACTUALIZAR UN EMPLEADO Y SU CARGO

### CÓDIGO COMPLETO:

```
# Procesar edición con cargo actualizado
def actualizarEmpleado(request):
    id = request.POST["id"]
    empleado = Empleado.objects.get(id=id)

    empleado.nombres = request.POST["nombres"]
    empleado.cedula = request.POST["cedula"]

    id_cargo = request.POST["cargo"]
    empleado.cargo = Cargo.objects.get(id=id_cargo)

    empleado.save()
    messages.success(request, "Empleado actualizado correctamente")
    return redirect('listaEmpleados')
```

**id = request.POST["id"]**

- Toma el ID del empleado que se está editando.



- Este valor llega desde un input oculto en el formulario (<input type="hidden" name="id" value="{{ empleado.id }}>).

**empleado = Empleado.objects.get(id=id)**

- Busca en la base de datos el empleado que tiene ese ID.
- El resultado es un objeto Empleado que podemos modificar.

**empleado.nombres = request.POST["nombres"]**

- Cambia el valor del campo nombres del empleado con el nuevo valor que escribió el usuario.

**empleado.cedula = request.POST["cedula"]**

- Cambia el valor de la cédula del empleado.

**id\_cargo = request.POST["cargo"]**

- Toma el nuevo ID del cargo que el usuario seleccionó.

**empleado.cargo = Cargo.objects.get(id=id\_cargo)**

- Busca el objeto Cargo con ese ID y lo asigna al campo cargo del empleado.

**empleado.save()**

- Guarda todos los cambios que hicimos en el objeto empleado en la base de datos.

**messages.success(request, "Empleado actualizado correctamente")**

- Muestra un mensaje en la web indicando que el empleado fue editado con éxito.

**return redirect('listaEmpleados')**

- Redirige al usuario a la lista de empleados.



## 15. SUBIR UN PROYECTO A GITLAB DESDE FEDORA

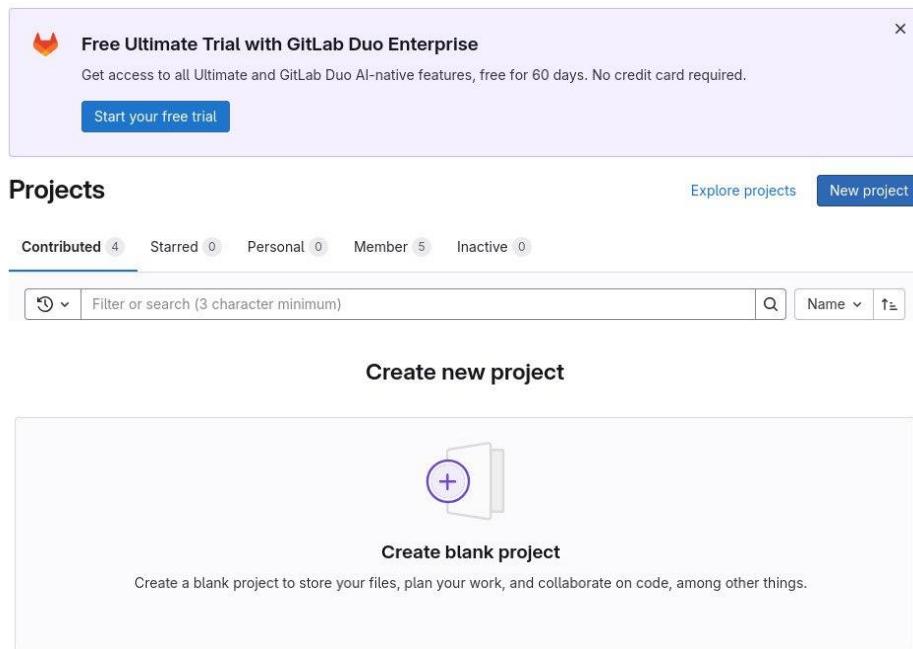
### 1 Crear un repositorio nuevo en GitLab

#### ¿Qué es un repositorio?

Un repositorio es como una carpeta especial en internet donde guardas todo tu proyecto para que esté seguro y puedas compartirlo con otros si quierés.

#### Pasos para crear el repositorio:

1. Abre tu navegador de internet (Firefox, Chrome, etc.) y entra a <https://gitlab.com>.
2. Si no tienes cuenta, primero regístrate, si ya tienes, inicia sesión con tu usuario y contraseña.
3. Una vez dentro, busca un botón que dice “**New project**” o “**Nuevo proyecto**”, normalmente está en la parte superior derecha o en el menú lateral.



The screenshot shows the GitLab homepage. At the top, there is a promotional banner for a free trial: "Free Ultimate Trial with GitLab Duo Enterprise". Below the banner, the main navigation bar includes links for "Explore projects" and "New project". Underneath the navigation, there are filters for "Contributed 4", "Starred 0", "Personal 0", "Member 5", and "Inactive 0". A search bar with placeholder text "Filter or search (3 character minimum)" is followed by a magnifying glass icon and sorting options "Name" and "↑". At the bottom of the page, a large button labeled "Create new project" is prominently displayed, accompanied by an icon of a document with a plus sign.

4. Se abrirá un formulario para crear tu proyecto.



## Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

Project name

trabajosDD\_conimages

Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

Project URL

https://gitlab.com/ mbue-j2d-group

Project slug

/ trabajosdd\_conimages

Project deployment target (optional)

Select the deployment target

Visibility Level ?

Private

Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.

Internal

The project can be accessed by any logged in user except external users.

Public

The project can be accessed without any authentication.

Project Configuration

Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Enable Static Application Security Testing (SAST)

Analyze your source code for known security vulnerabilities. [Learn more](#).

Enable Secret Detection

Scan your code for secrets and credentials to prevent unauthorized access. [Learn more](#).

› Experimental settings

[Create project](#)

[Cancel](#)

5. Ponle un nombre a tu proyecto, por ejemplo: trabajo o el nombre que tú quieras.

6. Elige si quieres que tu proyecto sea:

- **Privado:** Solo tú y las personas que invites pueden ver y modificar el proyecto.
- **Público:** Cualquiera puede verlo, pero solo tú y los que invites pueden cambiarlo.

7. **Importante:** Desmarca la opción que dice algo como “**Initialize repository with a README**” o “Iniciar repositorio con README”. Esto es para que el repositorio quede vacío y puedas subir tú todos los archivos desde tu computadora.

8. Haz clic en el botón “**Create project**” o “**Crear proyecto**”.

9. Ahora te llevará a una página con instrucciones para conectar tu proyecto local (tu PC) con este repositorio en GitLab.



## 2 Abrir la terminal y ubicarse en la carpeta de tu proyecto

### ¿Qué es la terminal?

La terminal es una ventana negra donde escribes comandos para que tu computadora haga cosas, es como hablarle directamente.

### ¿Dónde está tu proyecto?

Tienes que estar dentro de la carpeta donde tienes tu proyecto, porque ahí están todos los archivos que quieras subir.

Si tienes un proyecto Django, por ejemplo, en la carpeta donde está el archivo manage.py.

### Cómo moverte a esa carpeta:

- Abre la terminal (puedes buscar “Terminal” en tu sistema Fedora).
- Escribe el siguiente comando para entrar a la carpeta donde está tu proyecto (cambia la ruta si tu proyecto está en otro lugar):

```
cd ProyectosDJ/trabajosDD
```

- cd significa "change directory" (cambiar carpeta).
- ProyectosDJ/trabajosDD es la ruta de tu proyecto.
- Como es django para subir la carpeta completa siempre garantiza estar ubicado donde esta el archivo manage.py con el comando ls puede ver donde estas ubicado.

```
Th mbue@fedora:~/ProyectosDJ/trabajosDD$ ls
To Aplicaciones db.sqlite3 Img manage.py trabajosDD
mbue@fedora:~/ProyectosDJ/trabajosDD$ git init
```

## 3 Inicializar Git en la carpeta de tu proyecto

Git es la herramienta que usas para controlar versiones, es decir, para guardar los cambios de tu proyecto.

Para activar Git en tu carpeta (esto crea una carpeta oculta llamada .git), escribe:

```
git init
```

Si ves un mensaje que dice algo como "Initialized empty Git repository", todo bien.

```
Th mbue@fedora:~/ProyectosDJ/trabajosDD$ git init
hint: Usando 'master' como el nombre de la rama inicial. Este nombre de rama predeterminado
hint: està sujeto a cambios. Para configurar el nombre de la rama inicial para usar en todos
hint: de sus nuevos repositorios, reprimiendo esta advertencia, llama a:
hint:
You can hint: git config --global init.defaultBranch <nombre>
hint:
Corrige hint: Los nombres comúnmente elegidos en lugar de 'master' son 'main', 'trunk' y
hint: 'development'. Se puede cambiar el nombre de la rama recién creada mediante este comando:
Get s hint: git branch -m <nombre>
```



## 4 Configurar tu nombre y correo en Git

Para que Git sepa quién hace los cambios, tienes que poner tu nombre y correo. Esto solo se hace una vez en la computadora. En la terminal escribe estas dos líneas, cambiando por tu nombre y correo siquieres: Busca en la pagina que te da GitLab

```
git config --global user.name "MBUE-j2d"
```

```
git config --global user.email "jhefferson.daquilema8146@utc.edu.ec"
```

- --global significa que esta configuración se usa para todos tus proyectos.

```
create mode 100644 trabajosDD/urls.py
create mode 100644 trabajosDD/wsgi.py
mbue@fedora:~/ProyectosDJ/trabajosDD$ git config --local user.name "David Jhefferson"
mbue@fedora:~/ProyectosDJ/trabajosDD$ git config --local user.email "jhefferson.daquilema8146@utc.edu.ec"
mbue@fedora:~/ProyectosDJ/trabajosDD$
```

---

## 5 Preparar los archivos para guardar (hacer commit)

```
Loc mbue@fedora:~/ProyectosDJ/trabajosDD$ git add .
mbue@fedora:~/ProyectosDJ/trabajosDD$ git commit -m "Primer commit del proyecto trabajosDD"
Git [master (commit-raíz) 314e6be] Primer commit del proyecto trabajosDD
 86 files changed, 960 insertions(+)
```

Para decirle a Git que quieres guardar todos los archivos que están en la carpeta, escribes:

```
git add .
```

- El punto (.) significa “todos los archivos y carpetas aquí”.
  - Este comando prepara los archivos para guardarlos, pero todavía no los guarda.
- 

## 6 Guardar los archivos en Git (hacer commit)

Ahora tienes que guardar esos archivos con un mensaje que diga qué hiciste.

Ejecuta:

```
git commit -m "Primer commit del proyecto trabajosDD
```

```
"
```

- El texto entre comillas es el mensaje que describe qué guardaste.
- 

## 7 Conectar tu proyecto local con el repositorio de GitLab

Para subir tu proyecto a GitLab, tienes que decirle a Git dónde está el repositorio que creaste en GitLab.

- Primero, si ya hiciste esta conexión antes y te dice que el remoto “origin” ya existe, quítalo con:

```
git remote remove origin
```



- Ahora ve a GitLab, en la página de tu proyecto, busca la sección que dice “**Add files**” > **Escoje la opción HTTPS es la mejor.**

## Add files

Push files to this repository using SSH or HTTPS. If you're unsure, we recommend SSH.

SSH    **HTTPS**

Se recomienda escoger la opción HTTPS al subir un proyecto a GitLab porque es más fácil de usar y configurar, especialmente para principiantes. No necesitas generar claves ni hacer configuraciones adicionales; solo debes usar tu usuario y un token personal que puedes guardar para evitar ingresar cada vez. Además, HTTPS funciona en cualquier computadora de forma directa y es compatible con la autenticación moderna de GitLab, lo que lo hace seguro y conveniente. En cambio, usar SSH tiene varias desventajas: necesitas generar una clave SSH en tu máquina, agregarla manualmente a tu cuenta de GitLab, y si cambias de computadora o sistema operativo, debes volver a crear y registrar una nueva clave. También puede presentar problemas de conexión en redes restringidas donde el puerto SSH está bloqueado. Por eso, para la mayoría de usuarios y situaciones, HTTPS es la opción más práctica y rápida.

- Ahora ve al apartado “**Push an existing folder**” o algo similar, Y en **Configures the Git repository**
- Ahí en la 2 linea verás una línea que empieza con:

git remote add origin https://gitlab.com/usuario/nombre\_proyecto.git

- Copia esa línea completa y pégala en tu terminal. Esto conecta tu carpeta local con GitLab.

SSH    **HTTPS**

## Create a new repository

```
git clone https://gitlab.com/mbue-j2d-group/trabajosdd_conimages.git
cd trabajosdd_conimages
git switch --create main
touch README.md
git add README.md
git commit -m "add README"
git push --set-upstream origin main
```

## Push an existing folder

[Go to your folder](#)

```
cd existing_folder
```

## Configure the Git repository

```
git init --initial-branch=main
git remote add origin https://gitlab.com/mbue-j2d-group/trabajosdd_conimages.git
git add .
git commit -m "Initial commit"
git push --set-upstream origin main
```



---

## 8 Cambiar el nombre de la rama principal a main

Antes la rama principal se llamaba master, pero ahora se usa main.

Para cambiar el nombre ejecuta:

```
git branch -M main
```

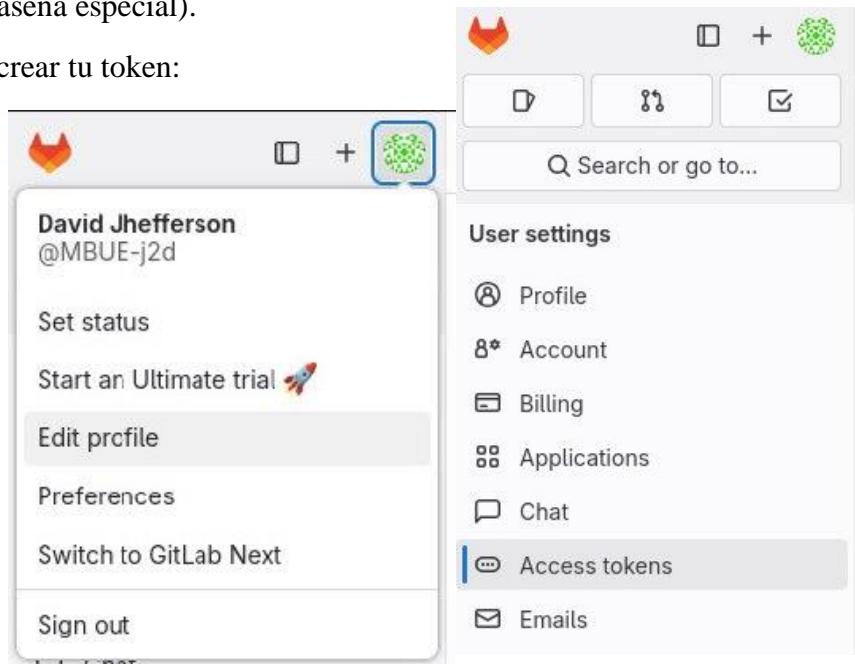
```
mbue@fedora:~/ProyectosDJ/trabajosDD$ git branch -M main
mbue@fedora:~/ProyectosDJ/trabajosDD$
```

---

## 9 Crear un token personal para subir archivos (autenticación segura)

GitLab ya no usa la contraseña normal para subir archivos, necesitas un "token" (una contraseña especial).

Para crear tu token:



1. En GitLab, ve a tu foto de perfil (arriba a la derecha) y elige “Edit profile” o “Editar perfil”.
2. En el menú de la izquierda, busca “Access Tokens” o “Tokens de acceso”.



## Personal access tokens

**Add a personal access token**

**Token name**  
admin

For example, the application using the token or the purpose of the token.

**Token description**  
lo que guste

**Expiration date**  
2025-07-01

An administrator has set the maximum expiration date to 2026-06-01. Learn more.

**Select scopes**  
Scopes set the permission levels granted to the token. Learn more.

- api**  
Grants complete read/write access to the API, including all groups and projects, the container registry, the dependency proxy, and the package registry.
- read\_api**  
Grants read access to the API, including all groups and projects, the container registry, and the package registry.
- read\_user**  
Grants read-only access to your profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.
- create\_runner**  
Grants create access to the runners.
- manage\_runner**  
Grants access to manage the runners.
- k8s\_proxy**  
Grants permission to perform Kubernetes API calls using the agent for Kubernetes.
- self\_rotate**  
Grants permission for token to rotate itself.
- read\_repository**  
Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
- write\_repository**  
Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API)

3. Ponle un nombre, por ejemplo token-mbue.
4. Marca estos permisos:
  - o  **read\_repository** (para leer repositorios)
  - o  **write\_repository** (para escribir o subir cosas)
5. Haz clic en “**Create personal access token**”.
6. Copia el código que te da (es como tu nueva contraseña secreta).

---

## 10. Subir tus archivos a GitLab (hacer push)

Ahora sí, sube tus archivos con este comando:

**git push -u origin main**

Cuando te pida:

- **Usuario:** pon tu usuario de GitLab (por ejemplo: MBUE-j2d).
- **Contraseña:** pega el token que copiaste (no uses tu contraseña normal).



## **[11] Guardar tu token para no escribirlo siempre**

Si quieras que Git recuerde tu token y no te lo pida cada vez, escribe:

```
git config --global credential.helper store
```

Así, la próxima vez que subas cambios no te pedirá usuario ni token.

Código Completo

```
mbue@fedora:~/ProyectosDJ/trabajosDD$ git config --local user.name "David Jhefferson"
mbue@fedora:~/ProyectosDJ/trabajosDD$ git config --local user.email "jhefferson.daquilema8146@utc.edu.ec"
mbue@fedora:~/ProyectosDJ/trabajosDD$ git remote add origin https://gitlab.com/mbue-j2d-group/trabajosdd_conimages.git
mbue@fedora:~/ProyectosDJ/trabajosDD$ git branch -M main
mbue@fedora:~/ProyectosDJ/trabajosDD$ git push -u origin main
Enumerando objetos: 93, listo.
Contando objetos: 100% (93/93), listo.
Compresión delta usando hasta 8 hilos
Comprimiendo objetos: 100% (91/91), listo.
Escribiendo objetos: 100% (93/93), 1.23 MiB | 28.10 MiB/s, listo.
Total 93 (delta 21), reused 0 (delta 0), pack-reused 0 (from 0)
To https://gitlab.com/mbue-j2d-group/trabajosdd_conimages.git
 * [new branch]      main -> main
rama 'main' configurada para rastrear 'origin/main'.
mbue@fedora:~/ProyectosDJ/trabajosDD$
```

## **Comprobar que todo salió bien**

Si todo funcionó, vuelve a la página de tu proyecto en GitLab y recarga.

Ahí deberías ver todos tus archivos subidos.

Name	Last commit	Last update
Aplicaciones	Primer commit del proyecto trabajosDD	7 minutes ago
img	Primer commit del proyecto trabajosDD	7 minutes ago
trabajosDD	Primer commit del proyecto trabajosDD	7 minutes ago
db.sqlite3	Primer commit del proyecto trabajosDD	7 minutes ago
manage.py	Primer commit del proyecto trabajosDD	7 minutes ago



## Manual detallado para usar una plantilla HTML descargada en un proyecto Django

### Objetivo:

Tomar una plantilla descargada (por ejemplo, de Tewngo), organizar sus archivos correctamente en Django y crear una **plantilla padre** (base.html) que se pueda reutilizar para todas las demás páginas del proyecto.

---

### Paso 1: Descargar la plantilla

1. Ve a una página como tewngo.com.
2. Descarga la plantilla que te guste. Te bajará un archivo comprimido, normalmente con nombre parecido a:

miPlantilla-v1.1.zip

 Este nombre puede variar, es solo un ejemplo.

---

### Paso 2: Crear carpetas en el proyecto Django para guardar los archivos

### Objetivo de este paso:

Colocar los archivos de la plantilla HTML en el lugar correcto para que Django los pueda usar como archivos estáticos.

### Ruta base del proyecto

Supongamos que tu proyecto Django está ubicado aquí:



Ese trabajosDD/ (el segundo) es la **carpeta raíz del proyecto**, donde están archivos como:

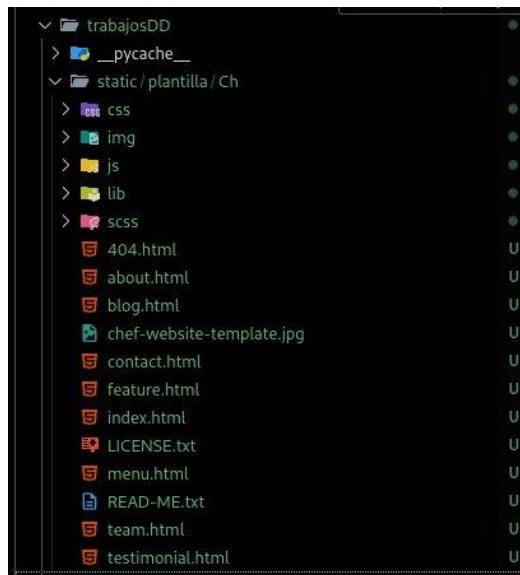
- settings.py
- urls.py
- wsgi.py
- \_\_init\_\_.py

### Ahora sigue estos pasos:

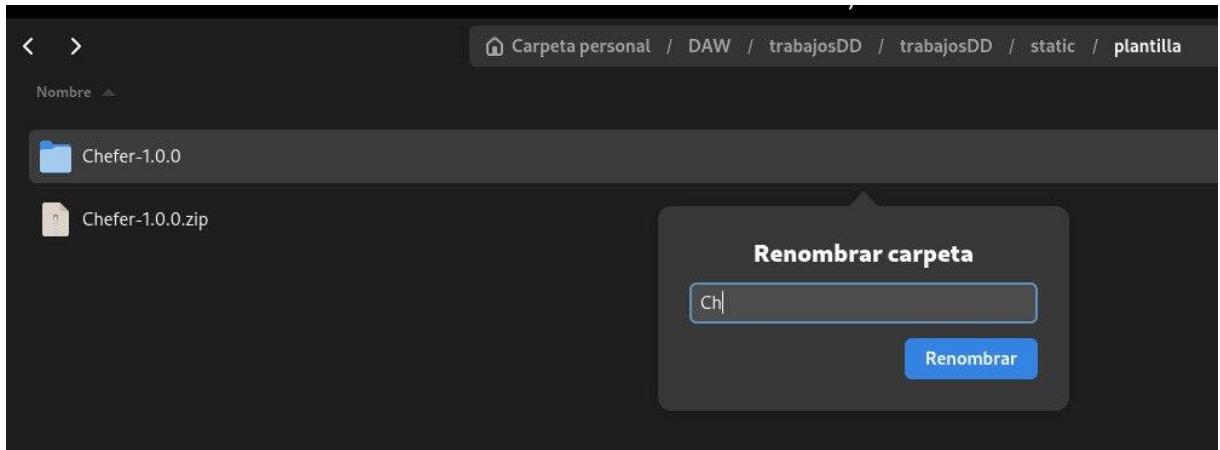
1. Dentro de esa carpeta (trabajosDD/), crea una nueva carpeta llamada: **static**
2. Dentro de static, crea otra carpeta llamada: **plantilla**
3. **Descomprime el archivo .zip que descargaste dentro de la carpeta plantilla.**

Te quedará algo como esto:

```
trabajosDD/  
|   static/  
|   |   plantilla/  
|   |   |   miPlantilla-v1.1/  
|   |   |   |   css/  
|   |   |   |   js/  
|   |   |   |   img/  
|   |   |   |   index.html
```



- Para que sea más fácil escribir las rutas, **renombrá la carpeta larga** (como miPlantilla-v1.1) a algo más corto, por ejemplo: **Ch**



Entonces, la estructura final sería:

trabajosDD/

```
|   └── static/
|       └── plantilla/
|           └── Ch/
|               ├── css/
|               ├── js/
|               ├── img/
|               └── index.html
```

💡 Esto es solo un ejemplo. En tu caso puedes usar otros nombres, lo importante es mantener el orden.



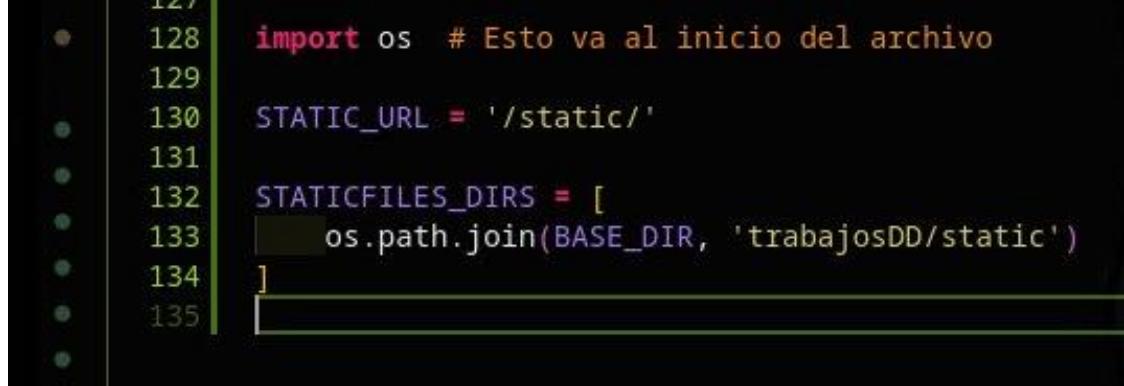
### Paso 3: Configurar settings.py para que Django encuentre los archivos

Abre el archivo **settings.py** y busca la parte donde se encuentra STATIC\_URL. Asegúrate de tener lo siguiente:

```
import os

STATIC_URL = '/static/'

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



#### ¿Qué significa todo esto?

- import os: Esto es necesario para trabajar con rutas de carpetas de forma correcta en Linux, Windows o Mac.
- STATIC\_URL = '/static/': Le dice a Django que los archivos estáticos se mostrarán usando /static/ en la URL del navegador.
- STATICFILES\_DIRS: Aquí le decimos a Django en qué carpeta **buscar** los archivos estáticos. En este caso, la carpeta trabajosDD/static.



## Paso 4: Crear la plantilla padre plantilla.html

### Objetivo:

Crear un archivo base que tendrá todo el diseño visual (como estilos, menú, pie de página) y que se pueda reutilizar en todas las páginas.

#### 1. Crea una carpeta llamada templates/ si aún no la tienes.

Dentro de ella, crea un archivo llamado: plantilla.html

#### 2. Abre el plantilla.html que vino con la plantilla y copia todo el contenido que tu gustes.

Pégalo dentro de plantilla.html.



```
index.html U
trabajosDD > static > plantilla > Ch > index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  ...
4  <head>
5      <meta charset="utf-8">
6      <title>CHEFER - Chef Website Template</title>
7      <meta content="width=device-width, initial-scale=1.0" name="viewport">
8      <meta content="Free HTML Templates" name="keywords">
9      <meta content="Free HTML Templates" name="description">
10
11     <!-- Favicon -->
12     <link href="img/favicon.ico" rel="icon">
13
14     <!-- Google Web Fonts -->
15     <link rel="preconnect" href="https://fonts.googleapis.com">
16     <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
17     <link href="https://fonts.googleapis.com/css2?family=Emblema+One&family=Poppins:wght@400;600&display=swap" rel="stylesheet">
18
19
20     <!-- Icon Font Stylesheet -->
21     <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.0/css/all.min.css" rel="stylesheet">
22     <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons.css" rel="stylesheet">
23
24     <!-- Libraries Stylesheet -->
25     <link href="lib/animate/animate.min.css" rel="stylesheet">
26     <link href="lib/owlcarousel/assets/owl.carousel.min.css" rel="stylesheet">
27
28     <!-- Customized Bootstrap Stylesheet -->
29     <link href="css/bootstrap.min.css" rel="stylesheet">
30
31     <!-- Template Stylesheet -->
32     <link href="css/style.css" rel="stylesheet">
33 </head>

plantilla.html M • index.html U
Aplicaciones > Empresas > templates > plantilla.html
1  {% load static %}
2
3
4  <!DOCTYPE html>
5  <html lang="en">
6
7  <head>
8      <meta charset="utf-8">
9      <title>CHEFER - Chef Website Template</title>
10     <meta content="width=device-width, initial-scale=1.0" name="viewport">
11     <meta content="Free HTML Templates" name="keywords">
12     <meta content="Free HTML Templates" name="description">
13
14     <!-- Favicon -->
15     <link href="img/favicon.ico" rel="icon">
16
17     <!-- Google Web Fonts -->
18     <link rel="preconnect" href="https://fonts.googleapis.com">
19     <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
20     <link href="https://fonts.googleapis.com/css2?family=Emblema+One&family=Poppins:wght@400;600&display=swap" rel="stylesheet">
21
22
23     <!-- Icon Font Stylesheet -->
24     <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.0/css/all.min.css" rel="stylesheet">
25     <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons.css" rel="stylesheet">
26
27     <!-- Libraries Stylesheet -->
28     <link href="lib/animate/animate.min.css" rel="stylesheet">
29     <link href="lib/owlcarousel/assets/owl.carousel.min.css" rel="stylesheet">
30
31     <!-- Customized Bootstrap Stylesheet -->
32     <link href="css/bootstrap.min.css" rel="stylesheet">
33
34     <!-- Template Stylesheet -->
```



## Paso 5: Hacer que la plantilla funcione con Django

### 1. Cargar los archivos estáticos

Arriba de todo, justo antes del <html> o al inicio del <head>, escribe esta línea:

```
{% load static %}
```

#### ? ¿Qué hace {% load static %}?

Django tiene su propio lenguaje de plantillas, y esta línea **le dice que vas a usar archivos estáticos** (como imágenes, CSS, JS). Si no pones esto, no se cargarán los estilos ni los scripts correctamente.

---

### 2. Cambiar las rutas de los archivos estáticos

En la plantilla, verás rutas así:

```
<link href="css/estilos.css" rel="stylesheet">  
<script src="js/funciones.js"></script>  
  
30 | <!-- Customized Bootstrap Stylesheet -->  
31 | <link href="css/bootstrap.min.css" rel="stylesheet">  
32 | <!-- Template Stylesheet -->  
  
<link href="{% static 'plantilla/Ch/css/estilos.css' %}" rel="stylesheet">  
<script src="{% static 'plantilla/Ch/js/funciones.js' %}"></script>  
  
30 | <!-- Customized Bootstrap Stylesheet -->  
31 | <link href="{% static 'css/bootstrap.min.css' %}" rel="stylesheet">  
32 | <!-- Template Stylesheet -->
```

#### ? ¿Qué significa esta línea?

```
{% static 'plantilla/Ch/css/estilos.css' %}
```

Esto le dice a Django:

"Busca un archivo que está dentro de la carpeta /static/plantilla/Ch/css/ y se llama estilos.css".



Por eso pusimos toda la plantilla dentro de static/plantilla/Ch/.

Haz esto con **todas las rutas** de imágenes, hojas de estilo (CSS) y scripts (JS).

### 3. Insertar el bloque donde irá el contenido de cada página

En el <body>, busca el lugar donde debería ir el contenido principal (por ejemplo, debajo del menú o del banner), y reemplaza eso por:

```
<div class="container-fluid">

    { % block contenido % }

    { % endblock % }

</div>
```



```
plantilla.html M index.html U
Aplicaciones > Empresas > templates > plantilla.html > html > head
3 | 
4 |     <!DOCTYPE html>
5 |     <html lang="en">
6 | 
7 |         <head>
8 |             <meta charset="utf-8">
9 |             <title>CHEFER - Chef Website Template</title>
10 |            <meta content="width=device-width, initial-scale=1.0" name="viewport">
11 |            <meta content="Free HTML Templates" name="keywords">
12 |            <meta content="Free HTML Templates" name="description">
13 | 
14 |            <!-- Favicon -->
15 |            <link href="img/favicon.ico" rel="icon">
16 | 
17 |            <!-- Google Web Fonts -->
18 |            <link rel="preconnect" href="https://fonts.googleapis.com">
19 |            <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
20 |            <link href="https://fonts.googleapis.com/css2?family=Emblema+One&family=Poppins:wght@400;600&display=swap" rel="stylesheet">
21 | 
22 |            <!-- Icon Font Stylesheet -->
23 |            <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.0/css/all.min.css" rel="stylesheet">
24 |            <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.1/font/bootstrap-icons.css" rel="stylesheet">
25 | 
26 |            <!-- Libraries Stylesheet -->
27 |            <link href="lib/animate/animate.min.css" rel="stylesheet">
28 |            <link href="lib/owlcarousel/assets/owl.carousel.min.css" rel="stylesheet">
29 | 
30 |            <!-- Customized Bootstrap Stylesheet -->
31 |            <link href="css/bootstrap.min.css" rel="stylesheet">
32 | 
33 |            <!-- Template Stylesheet -->
34 |            <link href="css/style.css" rel="stylesheet">
35 | 
36 | 
37 |         <div class="container-fluid">
38 | 
39 |             { % block contenido % }
40 |             { % endblock % }
41 | 
42 |         </div>
43 | 
44 |         <!-- aqui culaquier cosa igual eredara -->
45 | 
46 | 
```

Puedes dejar el menú arriba, el footer abajo y este bloque justo en el centro.

- ▲ Arriba de esto puedes dejar cualquier otra cosa como sliders, banners, menús... lo que tú quieras. El bloque contenido es solo para que las otras páginas puedan insertar su propio contenido ahí.

## Paso 6: Crear una página que use esta plantilla

Crea otro archivo en templates/, por ejemplo:

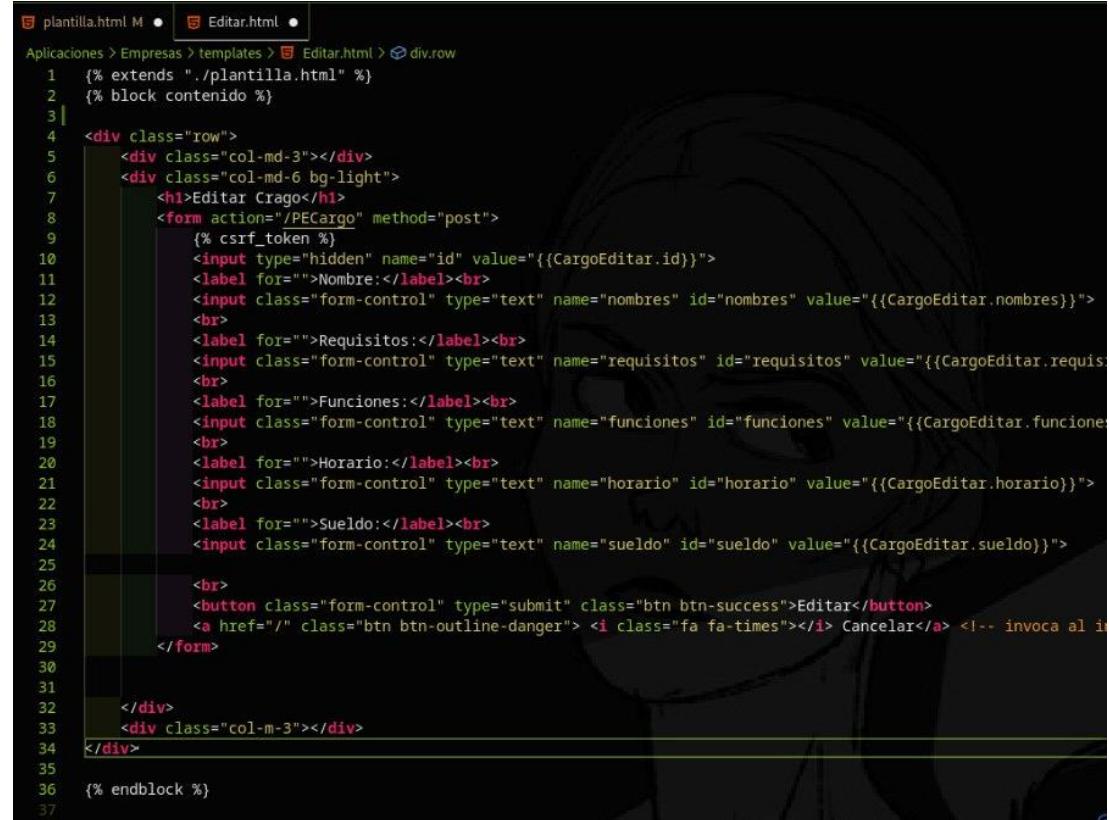
```
{% extends 'base.html' %}

{% block contenido %}

<h1>Bienvenido a mi página</h1>

<p>Este contenido está dentro del bloque contenido.</p>

{% endblock %}
```



```
plantilla.html M • Editar.html •
Aplicaciones > Empresas > templates > Editar.html > div.row
1  {% extends "./plantilla.html" %}
2  {% block contenido %}
3  |
4  <div class="row">
5  <div class="col-md-3"></div>
6  <div class="col-md-6 bg-light">
7    <h1>Editar Crago</h1>
8    <form action="/PECargo" method="post">
9      {% csrf_token %}
10     <input type="hidden" name="id" value="{{CargoEditar.id}}">
11     <label for="">>Nombre:</label><br>
12     <input class="form-control" type="text" name="nombres" id="nombres" value="{{CargoEditar.nombres}}">
13     <br>
14     <label for="">>Requisitos:</label><br>
15     <input class="form-control" type="text" name="requisitos" id="requisitos" value="{{CargoEditar.requisitos}}">
16     <br>
17     <label for="">>Funciones:</label><br>
18     <input class="form-control" type="text" name="funciones" id="funciones" value="{{CargoEditar.funciones}}">
19     <br>
20     <label for="">>Horario:</label><br>
21     <input class="form-control" type="text" name="horario" id="horario" value="{{CargoEditar.horario}}">
22     <br>
23     <label for="">>Sueldo:</label><br>
24     <input class="form-control" type="text" name="sueldo" id="sueldo" value="{{CargoEditar.sueldo}}">
25     <br>
26     <button class="form-control" type="submit" class="btn btn-success">Editar</button>
27     <a href="/" class="btn btn-outline-danger">  Cancelar</a> <!-- invoca al i
28   </form>
29
30
31
32   </div>
33   <div class="col-m-3"></div>
34
35
36  {% endblock %}
```

Este archivo está **heredando la estructura de base.html** y rellenando la parte que tiene el bloque contenido.



## 16 Manual paso a paso para validar formularios con jQuery Validation

### 1. Entender las librerías que necesitas

Para validar el formulario con jQuery Validation, necesitas estas librerías (archivos externos) que debes cargar en el <head> de tu HTML:

- **Bootstrap CSS:** para los estilos bonitos de la página y los formularios.
  - **Font Awesome:** para iconos (opcional).
  - **SweetAlert2:** para mostrar alertas bonitas cuando el formulario es válido o inválido.
  - **jQuery:** librería principal que necesita jQuery Validation para funcionar.
  - **jQuery Validation:** librería que hace la validación fácil.
- 

### 2. Cómo poner las librerías en el <head>

Este código lo tienes que poner dentro de la etiqueta <head>...</head>. Ponlas en este orden para que no haya errores en especial el 4,5,6 los otros son otra cosa, esto muestra para que sepas como pegar:

```
<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1" />

<title>Validación de formulario</title>

<!-- 1. Bootstrap CSS para estilos --&gt;

&lt;link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.3/css/bootstrap.min.css"
crossorigin="anonymous" /&gt;

<!-- 2. Font Awesome para iconos (opcional) --&gt;

&lt;link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.7.2/css/all.min.css"
crossorigin="anonymous" /&gt;

<!-- 3. SweetAlert2 para alertas --&gt;

&lt;script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"&gt;&lt;/script&gt;</pre>
```



```

<!-- 4. jQuery: es muy importante que solo se importe una vez -->
<script src="https://code.jquery.com/jquery-3.7.1.js" crossorigin="anonymous"></script>

<!-- 5. jQuery Validation para validar formularios -->
<script src="https://cdn.jsdelivr.net/npm/jquery-validation@1.19.5/dist/jquery.validate.js"></script>

<!-- 6. Estilos para errores -->
<style>

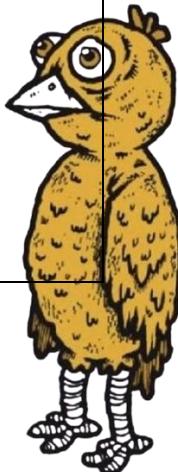
.error {
    color: red;          /* Color rojo para mensajes de error */
    font-weight: bold;   /* Texto en negrita */
}

.form-control.error {
    border: 3px solid red; /* Borde rojo en los inputs con error */
}

</style>

```

</head>



plantilla.html M • nuevoCargo.html Editar.html •

acciones > Empresas > templates > plantilla.html > html > head > style

```

<html lang="es">
<head>
    <!-- JS de Bootstrap con bundle que incluye Popper -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.3/js/bootstrap.bundle.min.js"
        crossorigin="anonymous" referrerpolicy="no-referrer"></script>

    <!-- Importacion link -->
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.7.2/css/all.min.css"
        integrity="sha512-Evv84Mr4kqVGRNSgIGL/F/aIdqQb7xQ2VcrdIwxjfThSH8CSR7PBEakCi51Ck+w+/U6swU2Im1vVX0SVk9ABhg=="
        crossorigin="anonymous" referrerpolicy="no-referrer" />

    <!-- Importacion link de sweetalert2-->
    <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>

    <!-- 4. jQuery: es muy importante que solo se importe una vez -->
    <script src="https://code.jquery.com/jquery-3.7.1.js" crossorigin="anonymous"></script>

    <!-- 5. jQuery Validation para validar formularios -->
    <script src="https://cdn.jsdelivr.net/npm/jquery-validation@1.19.5/dist/jquery.validate.js"></script>

    <!-- 6. Estilos para errores -->
    <style>
        .error {
            color: red;          /* Color rojo para mensajes de error */
            font-weight: bold;   /* Texto en negrita */
        }

        .form-control.error {
            border: 3px solid red; /* Borde rojo en los inputs con error */
        }
    </style>

```

### 3. Crear el formulario

En el <body> debes poner un formulario que cumpla con estas reglas para que jQuery Validation lo reconozca:

- El formulario debe tener un **id único** (en este caso frm\_nuevoCargo)
- Cada campo debe tener un atributo name (igual que usarás en las reglas de validación)
- Opcionalmente, pon id a cada input para que el label funcione bien

```
<body>
  <div class="container mt-5">
    <h3>Formulario para Nuevo Cargo</h3>

    <!-- 1. Formulario con id -->
    <form id="frm_nuevoCargo" action="{% url 'guardarCargo' %}"
method="post">
      {% csrf_token %} <!-- Si usas Django -->

      <!-- 2. Campo Nombres -->
      <div class="mb-3">
        <label for="nombres" class="form-label">Nombres:</label>
        <input type="text" class="form-control" id="nombres" name="nombres" />
      </div>

      <!-- 3. Campo Requisitos -->
      <div class="mb-3">
        <label for="requisitos" class="form-label">Requisitos:</label>
        <input type="text" class="form-control" id="requisitos" name="requisitos" />
      </div>

      <!-- 4. Campo Funciones -->
      <div class="mb-3">
        <label for="funciones" class="form-label">Funciones:</label>
        <input type="text" class="form-control" id="funciones" name="funciones" />
      </div>

      <!-- 5. Campo Horario -->
      <div class="mb-3">
        <label for="horario" class="form-label">Horario:</label>
        <input type="text" class="form-control" id="horario" name="horario" />
      </div>

      <!-- 6. Campo Sueldo -->
      <div class="mb-3">
        <label for="sueldo" class="form-label">Sueldo:</label>
        <input type="number" class="form-control" id="sueldo" name="sueldo" />
      </div>
```



```

<!-- 7. Botón para enviar -->
<button type="submit" class="btn btn-primary">Guardar</button>
</form>
</div>

```

```

antilla.html M • nuevoCargo.html • Editar.html •
taciones > Empresas > templates > nuevoCargo.html > div.row > div.col-md-6.bg-light > form#frm_nuevoCargo
{% extends ".plantilla.html" %}
{% block contenido %}


# Nuevo Cargo


<form action="guardarCargo" method="post" id="frm_nuevoCargo">
    {% csrf_token %}
    <label for="">Nombre:</label><br>
    <input class="form-control" type="text" name="nombres" id="nombres">
    <br>
    <label for="">Requisitos:</label><br>
    <input class="form-control" type="text" name="requisitos" id="requisitos">
    <br>
    <label for="">Funciones:</label><br>
    <input class="form-control" type="text" name="funciones" id="funciones">
    <br>
    <label for="">Horario:</label><br>
    <input class="form-control" type="text" name="horario" id="horario">
    <br>
    <label for="">Sueldo:</label><br>
    <input class="form-control" type="text" name="sueldo" id="sueldo">
    <br>
    <button class="form-control" type="submit" class="btn btn-success">Guardar</button>
    <a href="/" class="btn btn-outline-danger">Cancelar</a> <!-- invoca al inicio-->
</form>

</div>
<div class="col-m-3"></div>
</div>


```

#### 4. Código JavaScript para validar el formulario

Después de tu formulario, antes de cerrar el `</body>`, pon este código dentro de una etiqueta `<script>`. Esto es lo que hace que el formulario se valide:

```

<script>

    // Aquí le decimos a jQuery Validation que valide el formulario con id
    frm_nuevoCargo

    $("#frm_nuevoCargo").validate({
        // Reglas para cada campo (usa el atributo 'name' para referirte)
        rules: {
            nombres: {
                required: true, // campo obligatorio
                minlength: 5, // mínimo 5 caracteres
                maxlength: 10 // máximo 10 caracteres
            },
        }
    });

```



```

requisitos: {
    required: true
},
funciones: {
    required: true
},
horario: {
    required: true
},
sueldo: {
    required: true,
    number: true, // debe ser número
    min: 200,    // mínimo 200
    max: 5000    // máximo 5000
}
},
// Mensajes de error personalizados
messages: {
    nombres: {
        required: "Campo obligatorio",
        minlength: "Debe tener al menos 5 caracteres",
        maxlength: "Máximo 10 caracteres"
    },
    requisitos: {
        required: "Ingrese los requisitos"
    },
    funciones: {
        required: "Ingrese las funciones"
    }
},

```



```

horario: {
    required: "Ingrese el horario"
},
sueldo: {
    required: "Ingrese el sueldo",
    number: "Debe ser un número",
    min: "El mínimo permitido es 200",
    max: "El máximo permitido es 5000"
}
},

// Clase que se usará para mostrar errores (con el CSS que pusimos)
errorClass: "error",
errorElement: "span",

// Esto pinta el borde rojo cuando hay error
highlight: function(element) {
    $(element).addClass("error");
},
// Quita el borde rojo cuando ya no hay error
unhighlight: function(element) {
    $(element).removeClass("error");
},

// Esta función se ejecuta cuando el formulario es válido y se va a enviar
submitHandler: function(form) {
    // Mostrar alerta con SweetAlert2 (puedes cambiar esto si quieres)
    Swal.fire({
        icon: 'success',
        title: '¡Formulario válido!',

```



```

text: 'El formulario está listo para enviar.'

}).then(() => {

    form.submit(); // Aquí se envía el formulario

});

});

});

</script>

```

```

html M nuevoCargo.html M Editar.html M
s > Empresas > templates > nuevoCargo.html > div.row > script > rules


<form action="guardarCargo" method="post" id="frm_nuevoCargo">
    <input class="form-control" type="text" name="requisitos" id="requisitos">
    <br>
    <label for="">Funciones:</label><br>
    <input class="form-control" type="text" name="funciones" id="funciones">
    <br>
    <label for="">Horario:</label><br>
    <input class="form-control" type="text" name="horario" id="horario">
    <br>
    <label for="">Sueldo:</label><br>
    <input class="form-control" type="text" name="sueldo" id="sueldo">

    <br>
    <button class="form-control" type="submit" class="btn btn-success">Guardar</button>
    <a href="/" class="btn btn-outline-danger">Cancelar</a> <!-- invoca al inicio-->
</form>

</div>
<div class="col-m-3"></div>

<script>
    // Aquí le decimos a jQuery Validation que valide el formulario con id frm_nuevoCargo
    $("#frm_nuevoCargo").validate({
        // Reglas para cada campo (usa el atributo 'name' para referirte)
        rules: {
            nombres: {
                required: true,      // campo obligatorio
                minlength: 5,       // mínimo 5 caracteres
                maxlength: 10        // máximo 10 caracteres
            },
            requisitos: {
                required: true
            },
            funciones: {
                required: true
            },
            horario: {
                required: true
            },
            sueldo: {
                required: true,
                number: true,        // debe ser número
                min: 200,             // mínimo 200
                max: 5000             // máximo 5000
            }
        },
        // Mensajes de error personalizados
        messages: {
            nombres: {
                required: "Campo obligatorio",
                minlength: "Debe tener al menos 5 caracteres",
                maxlength: "Máximo 10 caracteres"
            },
            requisitos: {
                required: "Ingrese los requisitos"
            },
            funciones: {
                required: "Ingrese las funciones"
            },
            horario: {
                required: "Ingrese el horario"
            },
            sueldo: {
                required: "Ingrese el sueldo",
                number: "Debe ser un número",
                min: "El mínimo permitido es 200",
                max: "El máximo permitido es 5000"
            }
        }
    });


```



---

## 5. Qué hace cada parte importante

- `$("#frm_nuevoCargo").validate({...})`  
Le dice a jQuery Validation que valide el formulario con id frm\_nuevoCargo.
- rules  
Aquí defines las reglas de validación para cada campo. Por ejemplo, required: true dice que el campo es obligatorio.
- messages  
Son los mensajes personalizados que verá el usuario cuando el campo no pase la validación.
- errorClass y errorElement  
Define cómo se muestran los mensajes de error. Usamos la clase CSS .error que definimos arriba para que el texto sea rojo y el borde del input también.
- highlight y unhighlight  
Añaden o quitan la clase de error para mostrar o quitar el borde rojo en los campos que tengan errores.
- submitHandler  
Se ejecuta cuando el formulario pasa todas las validaciones. Aquí puedes mostrar un mensaje bonito con SweetAlert2 antes de enviarlo o hacer otra cosa.

---

## 6. Puntos importantes para que funcione bien

- **El formulario debe tener un id único** y el mismo que usas en el código jQuery.
- **Cada campo debe tener un name** (igual que usas en rules y messages).
- No importes jQuery más de una vez (solo una vez).
- Debes cargar todas las librerías en el orden correcto (jQuery antes que jQuery Validation).
- Los mensajes y reglas se escriben según el name del input.



## 17 implementar subida y edición de archivos (imágenes y PDFs) en Django

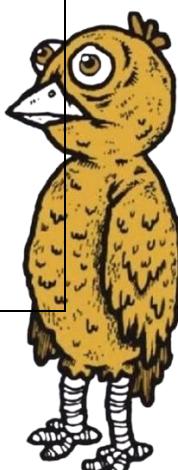
### 1. Modelo Cargo en models.py

```
 3 class Cargo(models.Model):
 4     id = models.AutoField(primary_key=True) # ID único que se crea solo
 5     nombres = models.CharField(max_length=100) # Nombre del cargo
 6     funciones = models.TextField() # Descripción de funciones
 7     horario = models.CharField(max_length=500) # Horario del cargo
 8     requisitos = models.TextField() # Requisitos del cargo
 9     sueldo = models.TextField() # Sueldo, texto para guardar decimales con coma o punto
10
11     # Campo para guardar el logotipo (imagen o archivo)
12     logo = models.FileField(
13         upload_to='cargos', # Carpeta dentro de media/ donde se guarda el logo
14         null=True,          # Permite que el campo quede vacío (nulo)
15         blank=True          # Permite que el formulario lo deje vacío
16     )
17
18     # Campo nuevo para guardar archivos PDF
19     pdf = models.FileField(
20         upload_to='pdfs', # Carpeta dentro de media/ donde se guarda el PDF
21         null=True,          # Permite campo vacío
22         blank=True
23     )
24
25     def __str__(self):
26         return f'{self.id} - {self.nombres} | Funciones: {self.funciones} | Horario: {self.horario}'
27
```

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

```
class Cargo(models.Model):
    id = models.AutoField(primary_key=True) # ID único que se crea solo
    nombres = models.CharField(max_length=100) # Nombre del cargo
    funciones = models.TextField() # Descripción de funciones
    horario = models.CharField(max_length=500) # Horario del cargo
    requisitos = models.TextField() # Requisitos del cargo
    sueldo = models.TextField() # Sueldo, texto para guardar decimales con coma o punto
```

```
# Campo para guardar el logotipo (imagen o archivo)
logo = models.FileField(
    upload_to='cargos', # Carpeta dentro de media/ donde se guarda el logo
    null=True,          # Permite que el campo quede vacío (nulo)
    blank=True          # Permite que el formulario lo deje vacío
```



```
)
```

```
# Campo nuevo para guardar archivos PDF
pdf = models.FileField(
    upload_to='pdfs', # Carpeta dentro de media/ donde se guarda el PDF
    null=True,      # Permite campo vacío
    blank=True
)

def __str__(self):
    # Esta función sirve para mostrar el contenido del cargo de forma legible
    return f'{self.id} - {self.nombres} | Funciones: {self.funciones} | Horario: {self.horario} |
Requisitos: {self.requisitos} | Sueldo: ${self.sueldo}"
```

- ◆ 1. Campo para guardar el logotipo (imagen)

```
logo = models.FileField(
    upload_to='cargos',
    null=True,
    blank=True
)
```

💡 Explicación:

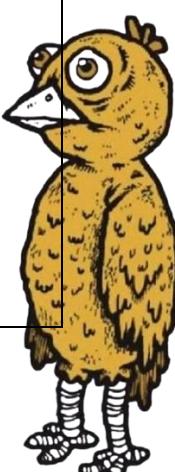
- FileField: Este tipo de campo se usa en Django para guardar archivos, ya sean imágenes, PDFs, documentos Word, etc.
- upload\_to='cargos': Esto indica que todos los archivos que se suban en este campo se guardarán dentro de la carpeta media/cargos/.
- null=True: Permite que este campo quede vacío en la base de datos. Es decir, no es obligatorio subir una imagen.
- blank=True: Permite que el formulario no exija este campo. Puedes dejarlo sin seleccionar nada al guardar o editar.
- Aunque FileField acepta cualquier archivo, en el formulario HTML tú limitas este campo a solo imágenes con accept="image/\*".

💡 Ejemplo de ruta donde se guarda:

Si subes un archivo llamado logo\_empresa.png, se guardará en:  
media/cargos/logo\_empresa.png

- 
- ◆ 2. Campo para guardar un archivo PDF

```
pdf = models.FileField(
```



```
upload_to='pdfs',
null=True,
blank=True
)
```

#### 💡 Explicación:

- Este campo funciona igual que el de logo, pero los archivos se guardan en otra carpeta.
- upload\_to='pdfs': Los archivos PDF se guardarán en media/pdfes/.
- null=True y blank=True: Permiten que este campo sea opcional tanto en la base como en el formulario.
- En el formulario HTML tú lo limitas con accept="application/pdf", así que aunque aquí no se especifica el tipo, el usuario solo podrá subir archivos PDF desde el navegador.

#### 💡 Ejemplo de ruta donde se guarda:

Si subes un archivo llamado cargo\_doc.pdf, se guardará en:  
media/pdfes/cargo\_doc.pdf

## 2. Vistas en views.py

### a) Guardar un nuevo cargo (crear)

```
def nuevoCargo(request):
    return render(request,"nuevoCargo.html")

def guardarCargo(request):
    nombres=request.POST["nombres"]
    requisitos=request.POST["requisitos"]
    funciones=request.POST["funciones"]
    horario=request.POST["horario"]
    sueldo=request.POST["sueldo"].replace(',','.')
    #subiendo archivo, ene ste caos se usa la propiedad files, que esta implementada esa toma el archivo y sube a la carpeta correspondiente
    # se usa los parentesis a qui va parentesis ()
    logo=request.FILES.get("logo")
    # Aqui tomamos el archivo PDF subido (si el usuario subio alguno)
    pdf = request.FILES.get("pdf")

    nuevoCargo=Cargo.objects.create(
        nombres=nombres,
        funciones=funciones,
        horario=horario,
        requisitos=requisitos,
        sueldo=sueldo,
        logo=logo,
        pdf=pdf
    )
    #antes del redirect poner lso siguiente/mensaje de confirmacion/creas mos un mensaje de confirmacion con success
    messages.success(request,"Cargo creado exitosamente")
    return redirect('/')
```

### def guardarCargo(request):

```
# Recibimos los datos del formulario por POST
nombres = request.POST["nombres"]
requisitos = request.POST["requisitos"]
funciones = request.POST["funciones"]
horario = request.POST["horario"]
sueldo = request.POST["sueldo"].replace(',', '.') # Reemplaza coma por punto para decimales
```

```
# Recibimos los archivos subidos (logo y pdf)
logo = request.FILES.get("logo") # Puede ser None si no se sube nada
pdf = request.FILES.get("pdf") # Igual que el logo
```

```
# Creamos el nuevo registro en la base de datos
```



```

nuevoCargo = Cargo.objects.create(
    nombres=nombres,
    funciones=funciones,
    horario=horario,
    requisitos=requisitos,
    sueldo=sueldo,
    logo=logo,
    pdf=pdf
)

# Mensaje de éxito que se muestra al usuario
messages.success(request, "Cargo creado exitosamente")

# Redirigimos al inicio o donde quieras
return redirect('/')

```

Claro, aquí tienes la explicación completa, detallada y enfocada únicamente en las líneas que manejan la subida de archivos (imagen y PDF) dentro de la función guardarCargo(request), ideal para un manual para programadores principiantes:

■ Explicación técnica – Subida de imagen (logo) y PDF al crear un cargo

Función: guardarCargo(request)

Enfocado solo en estas líneas:

### **Recibimos los archivos subidos (logo y pdf)**

```

logo = request.FILES.get("logo") # Puede ser None si no se sube nada
pdf = request.FILES.get("pdf") # Igual que el logo

```

...

```

logo=logo,
pdf=pdf

```

- ◆ ¿Qué hacen estas líneas?

Estas instrucciones sirven para recibir archivos que el usuario haya subido desde el formulario HTML y luego guardarlos directamente en la base de datos como parte del nuevo registro de cargo.

#### **✿ Línea 1:**

```
logo = request.FILES.get("logo")
```

- request.FILES es un diccionario que contiene todos los archivos que se suben en el formulario.
- .get("logo") busca si hay un archivo con el atributo name="logo".
- Si el usuario no seleccionó ningún archivo, el valor será None.
- Esto es útil porque no da error si no se sube nada.

#### **✿ Línea 2:**

```
pdf = request.FILES.get("pdf")
```

- Lo mismo que la línea anterior, pero esta vez busca el archivo PDF.



- Busca un archivo en el campo con name="pdf" en el formulario HTML.
- También puede ser None si el usuario no subió nada.

✿ Línea 3 (cuando se crea el cargo):

```
nuevoCargo = Cargo.objects.create(
```

```
...
```

```
logo=logo,
```

```
pdf=pdf
```

```
)
```

- Aquí se crean las columnas logo y pdf en el nuevo registro.
- Si el usuario subió los archivos, se guardan junto con el resto del cargo.
- Si no subió nada, esos campos quedan vacíos.

⚠ Importante:

- Para que esto funcione, el formulario HTML debe tener enctype="multipart/form-data", o Django no podrá recibir archivos.
- El modelo Cargo debe tener los campos definidos así:

```
logo = models.ImageField(upload_to='logos/', null=True, blank=True)
```

```
pdf = models.FileField(upload_to='documentos/', null=True, blank=True)
```

## b) Guardar cambios en un cargo existente (editar)



```
# Actualizando cargos

def ProcesarEdicionCargo(request):
    id = request.POST["id"]
    nombres = request.POST["nombres"]
    funciones = request.POST["funciones"]
    horario = request.POST["horario"]
    requisitos = request.POST["requisitos"]
    sueldo = request.POST["sueldo"].replace(',', '.')

    # Buscar el cargo a editar
    cargo = Cargo.objects.get(id=id)

    # Obtener la imagen nueva (si el usuario sube una)
    nuevo_logo = request.FILES.get("logo")
    # Obtener el nuevo archivo PDF (si se sube)
    nuevo_pdf = request.FILES.get("pdf")

    # Asignar los campos editados
    cargo.nombres = nombres
    cargo.funciones = funciones
    cargo.horario = horario
    cargo.requisitos = requisitos
    cargo.sueldo = sueldo

    # Solo reemplazar logo si se subió uno nuevo
    if nuevo_logo:
        cargo.logo = nuevo_logo

    # Solo reemplazar pdf si se subió uno nuevo
    if nuevo_pdf:
        cargo.pdf = nuevo_pdf

    cargo.save()
    messages.success(request, "Cargo actualizado exitosamente")
    return redirect('/')
```



```

def ProcesarEdicionCargo(request):
    # Recibimos los datos del formulario por POST
    id = request.POST["id"]
    nombres = request.POST["nombres"]
    funciones = request.POST["funciones"]
    horario = request.POST["horario"]
    requisitos = request.POST["requisitos"]
    sueldo = request.POST["sueldo"].replace(',', '.')

    # Buscamos el cargo que queremos editar
    cargo = Cargo.objects.get(id=id)

    # Recibimos los nuevos archivos, si es que se subieron
    nuevo_logo = request.FILES.get("logo")
    nuevo_pdf = request.FILES.get("pdf")

    # Actualizamos los campos del cargo con los datos nuevos
    cargo.nombres = nombres
    cargo.funciones = funciones
    cargo.horario = horario
    cargo.requisitos = requisitos
    cargo.sueldo = sueldo

    # Si subieron un logo nuevo, reemplazamos el anterior
    if nuevo_logo:
        cargo.logo = nuevo_logo

    # Si subieron un PDF nuevo, reemplazamos el anterior
    if nuevo_pdf:
        cargo.pdf = nuevo_pdf

    # Guardamos los cambios en la base de datos
    cargo.save()

    # Mensaje de éxito
    messages.success(request, "Cargo actualizado exitosamente")

    # Redirigimos al inicio
    return redirect('/')

```

```

# Recibimos los nuevos archivos, si es que se subieron
nuevo_logo = request.FILES.get("logo")
nuevo_pdf = request.FILES.get("pdf")

# Si subieron un logo nuevo, reemplazamos el anterior
if nuevo_logo:
    cargo.logo = nuevo_logo

# Si subieron un PDF nuevo, reemplazamos el anterior
if nuevo_pdf:

```



```
cargo.pdf = nuevo_pdf
```

🧠 Explicación línea por línea

◆ Línea 1:

```
nuevo_logo = request.FILES.get("logo")
```

🔍 ¿Qué hace?

- Busca si el formulario envió un archivo con el campo name="logo".
- Si el usuario subió una nueva imagen, esa imagen se guarda dentro de la variable nuevo\_logo como un archivo temporal.
- Si no subió nada, nuevo\_logo será None (nulo).

🧱 ¿Qué es request.FILES?

- Es un diccionario especial que Django usa para manejar los archivos enviados por el formulario con método POST y atributo enctype="multipart/form-data".
- En vez de request.POST (que solo contiene texto), los archivos deben capturarse con request.FILES.

🧱 ¿Qué es .get("logo")?

- Busca dentro del diccionario request.FILES un archivo con el nombre "logo".
- Este nombre debe coincidir con el name="logo" del input del formulario HTML.

◆ Línea 2:

```
nuevo_pdf = request.FILES.get("pdf")
```

🔍 ¿Qué hace?

- Exactamente lo mismo que la línea anterior, pero con el archivo PDF.
- Busca si el usuario subió un archivo con name="pdf".
- Si existe, lo guarda en la variable nuevo\_pdf.

◆ Línea 4 a 5:

```
if nuevo_logo:
```

```
    cargo.logo = nuevo_logo
```

🔍 ¿Qué hace?

- Verifica si la variable nuevo\_logo no está vacía.
- Si el usuario subió una imagen, reemplaza el archivo de imagen actual (que estaba guardado en cargo.logo) por el nuevo archivo que subió el usuario.
- Si no subió nada (nuevo\_logo es None), entonces no hace nada y se conserva el logo anterior.

🧱 cargo.logo = nuevo\_logo

- Aquí accedemos al campo logo del objeto cargo, que es un campo FileField o ImageField definido en el modelo.
- Asignamos directamente el nuevo archivo para que, cuando se haga cargo.save(), Django lo almacene en el servidor en la carpeta correspondiente (por ejemplo: /media/cargos/ o la que hayas definido en settings.py con MEDIA\_ROOT).

◆ Línea 7 a 8:

```
if nuevo_pdf:
```

```
    cargo.pdf = nuevo_pdf
```

🔍 ¿Qué hace?



- Igual que antes, pero para el archivo PDF.
- Si se subió un nuevo archivo PDF, reemplaza el que ya estaba guardado.
- Si no se subió nada, no se hace ningún cambio.

 Importante:

Si el usuario no cambia ni el logo ni el PDF, entonces esas líneas no hacen nada, y los archivos anteriores se mantienen.

### 3. Formulario en HTML para crear cargo (Ejemplo para crear)

```
<h1>Nuevo Cargo</h1>
<!-- 1. Primero se debe que todo lo componente tenga id, name ma simportante id -->

<form action="{% url 'guardarCargo' %}" method="post" id="frm_nuevoCargo" enctype="multipart/form-data">
<!-- 2. Agregar id al formualrio nec esario para sabe a quin validar -->

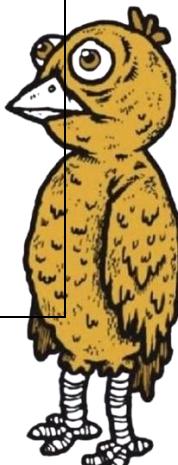
    {% csrf_token %}
    <label for="nombres">Nombre:</label><br>
    <input class="form-control" type="text" name="nombres" id="nombres" placeholder="Ingresar Nombre">
    <br>
    <label for="requisitos">Requisitos:</label><br>
    <input class="form-control" type="text" name="requisitos" id="requisitos" placeholder="Ingresar requisitos como experiencia,etc">
    <br>
    <label for="funciones">Funciones:</label><br>
    <input class="form-control" type="text" name="funciones" id="funciones" placeholder="Ingresar funciones como desarrollador,etc">
    <br>
    <label for="horario">Horario:</label><br>
    <input class="form-control" type="text" name="horario" id="horario" placeholder="Ingresar horario como de 12 a 17 pm">
    <br>
    <label for="sueldo">Sueldo:</label><br>
    <input class="form-control" type="number" name="sueldo" id="sueldo" placeholder="Ingresar sueldo como 20">
    <br>
    <label for="logotipo">Logotipo:</label><br>
    <input class="form-control" type="file" name="logo" id="logo" accept="image/*" placeholder="Selecione una imagen" style="width: 100%; height: 100%; opacity: 0;"/>
    <!-- Subir archivo PDF -->
    <label for="pdf">Archivo PDF:</label><br>
    <input class="form-control" type="file" name="pdf" id="pdf" accept="application/pdf"><br>
    <br>
    <button class="form-control" type="submit" class="btn btn-success">Guardar</button>
    <a href="/" class="btn btn-outline-danger">Cancelar</a> <!-- invoca al inicio-->
</form>
```

```
<form action="{% url 'guardarCargo' %}" method="post" id="frm_nuevoCargo"
enctype="multipart/form-data">
    {% csrf_token %}

    <label for="nombres">Nombre:</label><br>
    <input class="form-control" type="text" name="nombres" id="nombres" placeholder="Ingresar Nombre"><br>

    <label for="requisitos">Requisitos:</label><br>
    <input class="form-control" type="text" name="requisitos" id="requisitos" placeholder="Ingresar requisitos como experiencia"><br>

    <label for="funciones">Funciones:</label><br>
    <input class="form-control" type="text" name="funciones" id="funciones" placeholder="Ingresar funciones como desarrollador"><br>
```



```

<label for="horario">Horario:</label><br>
<input class="form-control" type="text" name="horario" id="horario" placeholder="Ingrese horario como de 12 a 17 pm"><br>

<label for="sueldo">Sueldo:</label><br>
<input class="form-control" type="number" name="sueldo" id="sueldo" placeholder="Ingrese sueldo como 20"><br>

<label for="logo">Logotipo:</label><br>
<input class="form-control" type="file" name="logo" id="logo" accept="image/*" placeholder="Seleccione una imagen"><br>

<label for="pdf">Archivo PDF:</label><br>
<input class="form-control" type="file" name="pdf" id="pdf" accept="application/pdf"><br><br>

<button class="form-control btn btn-success" type="submit">Guardar</button>
<a href="/" class="btn btn-outline-danger">Cancelar</a>
</form>

```

#### █ Explicación del formulario HTML

✖ Enfocado solo en la subida de archivos (imagen y PDF)

Fragmento del formulario:

```

<form action="{% url 'guardarCargo' %}" method="post" id="frm_nuevoCargo"
enctype="multipart/form-data">
...
<input class="form-control" type="file" name="logo" id="logo" accept="image/*"> <input
class="form-control" type="file" name="pdf" id="pdf" accept="application/pdf">

```

- ◆ ¿Qué hace esta parte del formulario?

Estas líneas permiten que el usuario pueda subir dos archivos: una imagen (logo) y un archivo PDF.

Estos archivos serán enviados al servidor junto con los demás campos del formulario.

Vamos parte por parte:

- ◆ 1. enctype="multipart/form-data"

```
<form ... enctype="multipart/form-data">
```

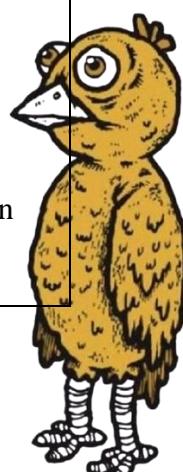
- Esta parte es obligatoria si quieres enviar archivos desde un formulario.
- multipart/form-data le dice al navegador que debe dividir los datos en partes separadas (texto + archivos) antes de enviarlos.
- Si no pones esto, Django no podrá leer los archivos que se suben, y request.FILES estará vacío.

- ◆ 2. Subida de imagen

```
<label for="logo">Logotipo:</label><br>
```

```
<input class="form-control" type="file" name="logo" id="logo" accept="image/*">
```

- type="file" crea un botón que permite seleccionar un archivo desde la computadora.
- name="logo": este nombre es clave, porque es el mismo que Django usará en views.py con request.FILES.get("logo").
- accept="image/\*" limita la selección solo a imágenes (JPG, PNG, etc.), para evitar que suban otro tipo de archivo por error.



### ◆ 3. Subida de PDF

- ```
<label for="pdf">Archivo PDF:</label><br>
<input class="form-control" type="file" name="pdf" id="pdf" accept="application/pdf">
    • type="file" igual que antes, pero para subir archivos PDF.
    • name="pdf": este nombre debe coincidir con el que se usa en Django: request.FILES.get("pdf").
    • accept="application/pdf" limita la selección a solo archivos con extensión .pdf. Es útil para que el usuario no se equivoque.
```
- ◆ ¿Qué pasa cuando se envía el formulario?
    - Si el usuario seleccionó archivos, el navegador los envía junto con el resto de los campos.
    - Django los recibe automáticamente en request.FILES.
    - Luego, en views.py los guarda como cualquier otro dato (como vimos antes con logo=logo, pdf=pdf).

## 3.b Formulario para editar (actualizar) un cargo — Editar.html

```
<form action="{% url 'PECargo' %}" method="post" enctype="multipart/form-data">
    value="{{ CargoEditar.requisitos }}"
<br>

    <!-- Funciones -->
    <label for="funciones">Funciones:</label>
    <input class="form-control" type="text" name="funciones" id="funciones" value="{{ CargoEditar.funciones }}"
<br>

    <!-- Horario -->
    <label for="horario">Horario:</label>
    <input class="form-control" type="text" name="horario" id="horario" value="{{ CargoEditar.horario }}"
<br>

    <!-- Sueldo -->
    <label for="sueldo">Sueldo:</label>
    <input class="form-control" type="text" name="sueldo" id="sueldo" value="{{ CargoEditar.sueldo }}"
<br>

    <!-- Logotipo actual -->
    <label for="logo">Logotipo:</label><br>
    {% if CargoEditar.logo %}
        <br><br>
    {% else %}
        <small>No hay logotipo cargado</small><br><br>
    {% endif %}
    <!-- Subir nuevo logotipo -->
    <input class="form-control" type="file" name="logo" id="logo"
<br>

    <!-- Archivo PDF actual -->
    <label for="pdf">Archivo PDF:</label><br>
    {% if CargoEditar.pdf %}
        <a href="{{ CargoEditar.pdf.url }}" target="_blank">Ver PDF actual</a><br><br>
    {% else %}
        <small>No hay archivo PDF cargado</small><br><br>
    {% endif %}
    <!-- Subir nuevo archivo PDF -->
    <input class="form-control" type="file" name="pdf" id="pdf" accept="application/pdf"
<br>

    <!-- Botones -->
    <button class="btn btn-success w-100" type="submit">Editar</button>
    <br><br>
    <a href="/" class="btn btn-outline-danger w-100">Cancelar</a>
</form>
```

Than  
a Sup  
use ti  
  
Origen: S



```

<form action="<% url 'PECargo' %>" method="post" enctype="multipart/form-data">
    {% csrf_token %}

    <!-- ID oculto para saber qué cargo editar -->
    <input type="hidden" name="id" value="{{ CargoEditar.id }}">

    <!-- Nombre -->
    <label for="nombres">Nombre:</label>
    <input class="form-control" type="text" name="nombres" id="nombres" value="{{ CargoEditar.nombres }}>

    <br>

    <!-- Requisitos -->
    <label for="requisitos">Requisitos:</label>
    <input class="form-control" type="text" name="requisitos" id="requisitos" value="{{ CargoEditar.requisitos }}>

    <br>

    <!-- Funciones -->
    <label for="funciones">Funciones:</label>
    <input class="form-control" type="text" name="funciones" id="funciones" value="{{ CargoEditar.funciones }}>

    <br>

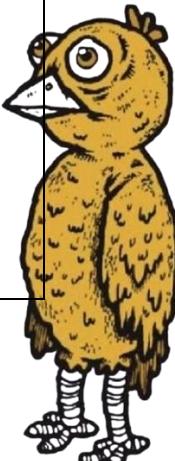
    <!-- Horario -->
    <label for="horario">Horario:</label>
    <input class="form-control" type="text" name="horario" id="horario" value="{{ CargoEditar.horario }}>

    <br>

    <!-- Sueldo -->
    <label for="sueldo">Sueldo:</label>
    <input class="form-control" type="text" name="sueldo" id="sueldo" value="{{ CargoEditar.sueldo }}>

    <br>

```



```

<!-- Mostrar logotipo actual -->
<label for="logo">Logotipo:</label><br>
{ % if CargoEditar.logo % }

    <br><br>

{ % else % }

    <small>No hay logotipo cargado</small><br><br>

{ % endif %}

<!-- Subir nuevo logotipo -->
<input class="form-control" type="file" name="logo" id="logo">
<br>

<!-- Mostrar archivo PDF actual -->
<label for="pdf">Archivo PDF:</label><br>
{ % if CargoEditar.pdf % }

    <a href="{{ CargoEditar.pdf.url }}" target="_blank">Ver PDF actual</a><br><br>

{ % else % }

    <small>No hay archivo PDF cargado</small><br><br>

{ % endif %}

<!-- Subir nuevo archivo PDF -->
<input class="form-control" type="file" name="pdf" id="pdf" accept="application/pdf">
<br>

<!-- Botones -->
<button class="btn btn-success w-100" type="submit">Editar</button>
<br><br>
<a href="/" class="btn btn-outline-danger w-100">Cancelar</a>
</form>

```



## ■ Sección: Formulario de edición de cargo

### ⌚ Enfoque: Subir y mostrar imagen (logo) y archivo PDF

#### 📌 1. enctype="multipart/form-data"

```
<form action="{% url 'PECargo' %}" method="post" enctype="multipart/form-data">
```

- Esta parte es esencial cuando quieras subir archivos desde un formulario.
- multipart/form-data le indica al navegador que envíe los datos en varias partes (campos + archivos).
- Sin esto, los archivos no llegarán a Django y request.FILES no tendrá nada.

#### 📌 2. Mostrar el logotipo actual

```
<label for="logo">Logotipo:</label><br>
{% if CargoEditar.logo %}
<br><br>
{% else %}
<small>No hay logotipo cargado</small><br><br>
{% endif %}
```

- Esta parte verifica si el cargo ya tiene una imagen (logo) guardada.
- Si existe, muestra la imagen actual usando la ruta {{ CargoEditar.logo.url }}.
- Si no hay imagen, muestra un pequeño mensaje diciendo “No hay logotipo cargado”.
- Esto es útil para que el usuario sepa qué imagen está guardada actualmente.

#### 📌 3. Subir un nuevo logotipo

```
<input class="form-control" type="file" name="logo" id="logo">
```

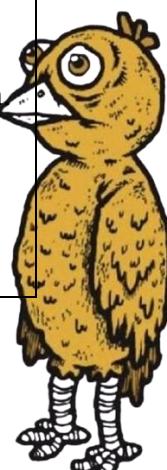
- Este input permite subir una nueva imagen para reemplazar el logo actual.
- type="file" crea el botón para seleccionar archivo.
- name="logo" es el nombre que Django usará para acceder al archivo con request.FILES.get("logo").
- A diferencia del formulario de creación, aquí no se usa accept="image/\*", pero se puede agregar si quieres restringir solo a imágenes.

#### 📌 4. Mostrar el PDF actual

```
<label for="pdf">Archivo PDF:</label><br>
```

```
{% if CargoEditar.pdf %}
<a href="{{ CargoEditar.pdf.url }}" target="_blank">Ver PDF actual</a><br><br>
{% else %}
<small>No hay archivo PDF cargado</small><br><br>
{% endif %}
```

- Aquí se verifica si el cargo ya tiene un archivo PDF.
- Si existe, se muestra un enlace para que el usuario pueda ver el PDF actual en una nueva pestaña.
- Si no hay archivo, se muestra un texto diciendo que no hay PDF cargado.
- Esto da contexto al usuario antes de subir un nuevo archivo.



## 5. Subir un nuevo archivo PDF

```
<input class="form-control" type="file" name="pdf" id="pdf" accept="application/pdf">
```

- Este input permite subir un nuevo archivo PDF.
- name="pdf": es el nombre que Django usará para recibir el archivo.
- accept="application/pdf": restringe que solo se pueda elegir archivos con extensión .pdf.

## Explicación:

### 4. Configuración en el archivo urls.py principal (del proyecto)

```
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))  
"""  
from django.contrib import admin  
from django.urls import path,include  
  
#para mostar los archivos en lso templates, esta dna el perimos para mostar iamgen  
from django.conf import settings  
from django.conf.urls.static import static  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path ('',include('Aplicaciones.Empresas.urls'))  
  
]  
  
if settings.DEBUG:  
    urlpatterns+=static(settings.MEDIA_URL,document_root=settings.MEDIA_ROOT)
```

```
from django.contrib import admin  
from django.urls import path, include  
from django.conf import settings  
from django.conf.urls.static import static  
  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path ("", include('Aplicaciones.Empresas.urls')),  
]  
  
if settings.DEBUG:  
    urlpatterns += static(settings.MEDIA_URL,  
    document_root=settings.MEDIA_ROOT)
```

## 6. Configuración importante en settings.py



```

# https://docs.djangoproject.com/en/5.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
# configurando acceso a la carpeta media (que dr usa para subida de archivos, )
# sudo chmod 777 -R nombreCarpeta o chown
#755 para un servidor
#1-7 lectura
#2-7escritura
#3-ejecucion

#agregar variable global, siempre en mayuscula

MEDIA_URL='/media/'

#va la sistema operativo y luego entra a la carpeta media, Basedir=ruta fisica donde se esta guardando el proyecto
MEDIA_ROOT=[os.path.join(BASE_DIR,'trabajosDD/media/')]

```

```

# Esto es para que Django use el tipo de campo ID correcto
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

import os

# Para archivos que se suben por el usuario (imágenes, PDFs, etc)
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'trabajosDD/media/')

```

## 7. Permisos para la carpeta media

Para que Django pueda guardar archivos en la carpeta media, asegúrate que el sistema operativo tenga permisos de escritura en esa carpeta.

```

sudo chmod 755 -R trabajosDD/media/
O para desarrollo:
sudo chmod 777 -R trabajosDD/media/

```

## Resumen final

- El modelo Cargo tiene campos para subir imagen y PDF.
- En la vista guardarCargo y ProcesarEdicionCargo se reciben los archivos con request.FILES.get().
- En el formulario HTML debes usar enctype="multipart/form-data".
- En urls.py se deben definir las rutas para crear, editar, actualizar y eliminar cargos.
- En settings.py se configura MEDIA\_URL y MEDIA\_ROOT.



- En el archivo principal de urls se agregan las rutas para servir archivos en desarrollo.

#### 4. Tabla para mostrar la lista de cargos

```

<tr>
    <th>ACCIONES</th>
</tr>
</thead>
<tbody>
    {% for cargoTemporal in cargo %}
    <tr>
        <td>{{cargoTemporal.id}}</td>
        <td>{{cargoTemporal.nombres}}</td>
        <td>{{cargoTemporal.funciones}}</td>
        <td>{{cargoTemporal.horario}}</td>
        <td>{{cargoTemporal.requisitos}}</td>
        <td>{{cargoTemporal.sueldo}}</td>
        <td>
            {% if cargoTemporal.logo %}
            
            {% else %}
            Ninguna
            {% endif %}
        </td>
        <td>
            {% if cargoTemporal.pdf %}
            <a href="{{ cargoTemporal.pdf.url }}" target="_blank">Ver PDF</a>
            {% else %}
            ninguno
            {% endif %}
        </td>
        <td>
            <a class="btn btn-warning" href="{% url 'EditarCargo' cargoTemporal.id %}"><i
                class="fa fa-pencil"></i>Editar</a>
            <br>
            <a href="#" onclick="confirmarEliminacion('{{ url 'ECargo' cargoTemporal.id }}')">
                <i class="fa fa-trash"></i>Eliminar
            </a>
        </td>
    </tr>
    {% endfor %}
</tbody>
</table>
```

```
<table class="table table-bordered table-striped table-hover">
```

```

<thead>
<tr>
    <th>ID</th>
    <th>NOMBRE</th>
    <th>FUNCIONES</th>
    <th>HORARIOS</th>
    <th>REQUISITOS</th>
    <th>SUELDO</th>
</tr>
</thead>
```



```

<th>LOGO</th>
<th>PDF</th>
<th>ACCIONES</th>
</tr>
</thead>
<tbody>
{ % for cargoTemporal in cargo %}
<tr>
<td>{{ cargoTemporal.id }}</td>
<td>{{ cargoTemporal.nombres }}</td>
<td>{{ cargoTemporal.funciones }}</td>
<td>{{ cargoTemporal.horario }}</td>
<td>{{ cargoTemporal.requisitos }}</td>
<td>{{ cargoTemporal.sueldo }}</td>
<td>
{ % if cargoTemporal.logo %}

{ % else %}
    Ninguno
{ % endif %}
</td>
<td>
{ % if cargoTemporal.pdf %}
<a href="{{ cargoTemporal.pdf.url }}" target="_blank">Ver PDF</a>
{ % else %}
    Ninguno
{ % endif %}
</td>
<td>
<a class="btn btn-warning" href="{ % url 'EditarCargo' cargoTemporal.id % }">

```



```

<i class="fa fa-pen"></i> Editar
</a>
<a href="#" onclick="confirmarEliminacion('{% url 'ECargo' cargoTemporal.id %}')">
    <i class="fa fa-trash"></i> Eliminar
</a>
</td>
</tr>
{% endfor %}
</tbody>
</table>

```

### Explicación:

- La tabla tiene encabezados para cada columna con los datos que quieras mostrar: ID, nombre, funciones, horarios, requisitos, sueldo, logo, PDF y acciones.
- Dentro del `<tbody>`, se usa un ciclo `{% for cargoTemporal in cargo %}` para recorrer la lista que envía la vista con todos los cargos y mostrar sus datos.
- Para el logo, si existe, se muestra la imagen; si no, muestra “Ninguno”. Lo mismo con el PDF, pero mostrando un enlace para abrirlo si está cargado.
- En la columna de acciones hay dos botones:
  - **Editar:** que lleva a la URL llamada 'EditarCargo' pasando el ID del cargo, para abrir el formulario de edición.
  - **Eliminar:** que llama a una función JavaScript `confirmarEliminacion(url)` para confirmar antes de eliminar el registro (esta función la debes implementar para que muestre un mensaje de confirmación y si acepta, haga la petición de borrado).

