



PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL  
FORMATO GUÍA DE APRENDIZAJE – MATERIAL DE APOYO  
DESARROLLO APLICACIONES WEB EN PYTHON

CREACIÓN DE APLICACIONES WEB EN PYTHON CON EL FRAMEWORK DJANGO



Para iniciar primero crear una carpeta donde se va a crear el proyecto. Después abrir visual studio code y abrir dicha carpeta. En mi caso la carpeta la he llamado **Proyecto Inicial Django**.

**Pasos:**

Crear un entorno virtual de trabajo para el proyecto

```
C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django>python -m venv entorno
```

Activar el entorno

```
C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django>activate
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django>
```

Ahora nos ubicamos en la raíz de la carpeta para instalar django así:



```
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\
Abril 17 de 2023\Proyecto Inicial Django>
```

```
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Pr
Abril 17 de 2023\Proyecto Inicial Django>pip install Django==4.2
Collecting Django==4.2
  Downloading Django-4.2-py3-none-any.whl (8.0 MB)
    8.0/8.0 MB 28.4 MB/s eta 0:00:00
Collecting asgiref<4,>=3.6.0
  Downloading asgiref-3.6.0-py3-none-any.whl (23 kB)
Collecting sqlparse>=0.3.1
  Downloading sqlparse-0.4.3-py3-none-any.whl (42 kB)
    42.8/42.8 kB ? eta 0:00:00
Collecting tzdata
  Downloading tzdata-2023.3-py2.py3-none-any.whl (341 kB)
    341.8/341.8 kB ? eta 0:00:00
Installing collected packages: tzdata, sqlparse, asgiref, Django
Successfully installed Django-4.2 asgiref-3.6.0 sqlparse-0.4.3 tzdata-2023.3

[notice] A new release of pip available: 22.3.1 -> 23.1
[notice] To update, run: python.exe -m pip install --upgrade pip

(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Pr
oyecto Inicial Django>
```

Tomado de la página oficial:

## Option 1: Get the latest official version

The latest official version is 4.2 (LTS). Read the [4.2 release notes](#), then install it with `pip`:

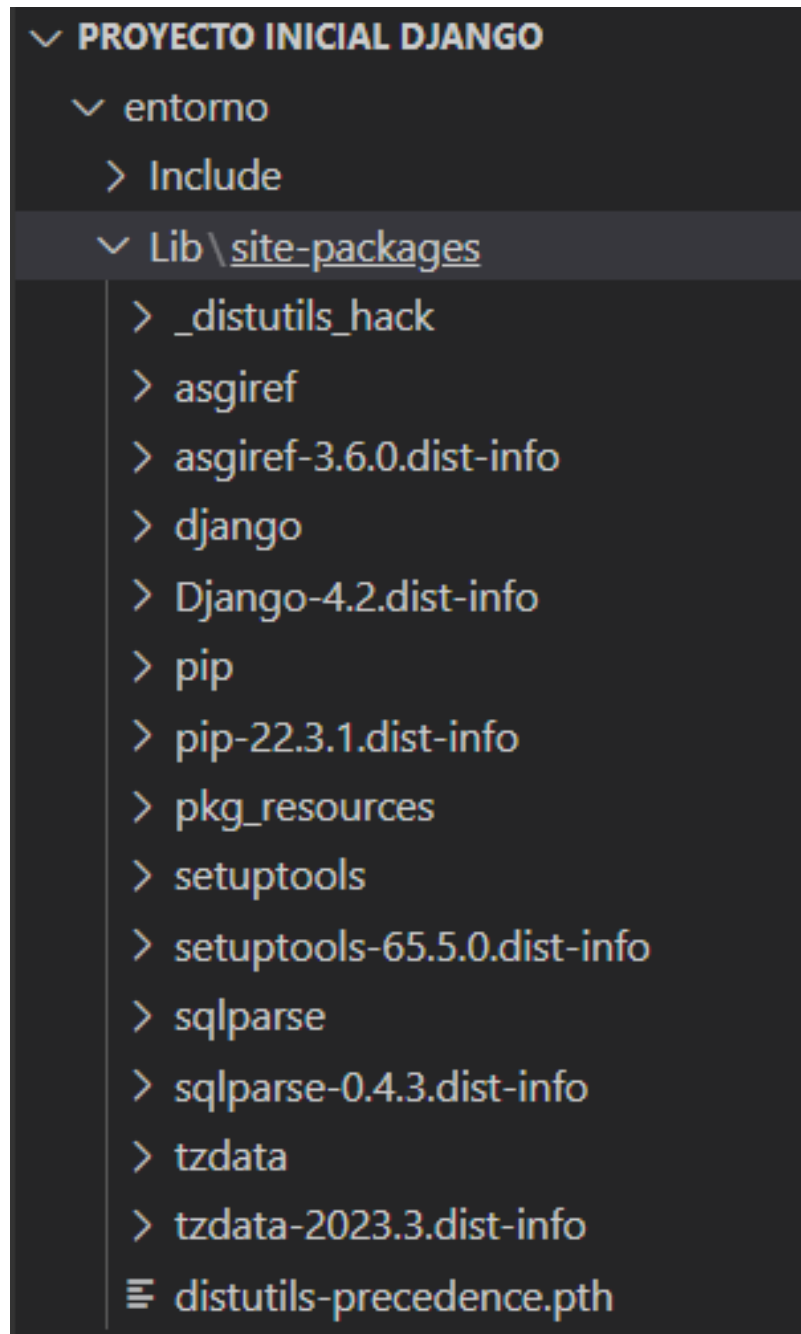
```
pip install Django==4.2
```

Si solicita actualizar pip hacer el siguiente comando:

```
python -m pip install --user --upgrade pip
```



Podemos verificar en la lib dentro de la carpeta del entorno para verificar si aparece django.

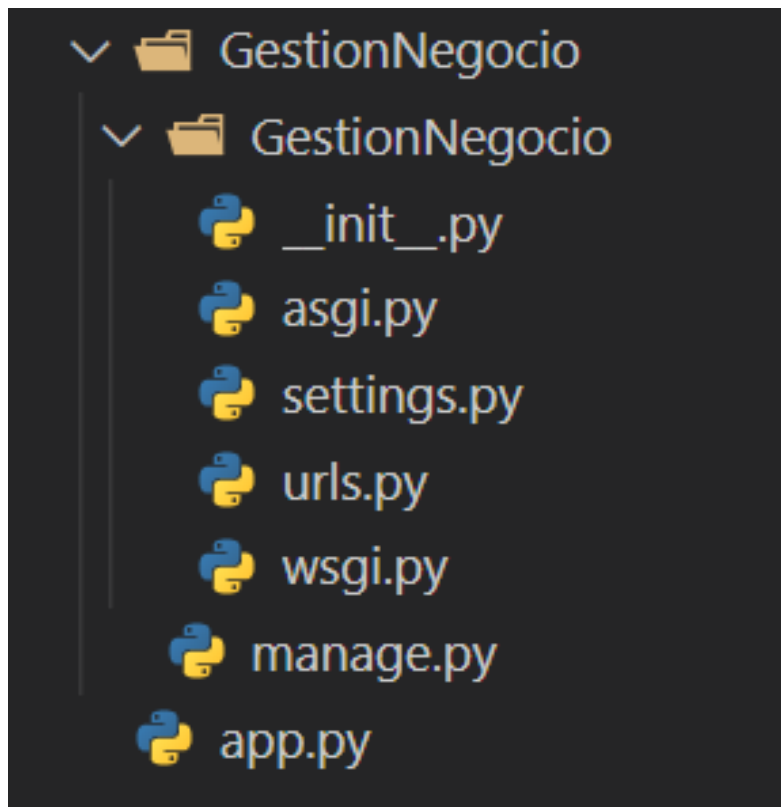


Ahora vamos a crear el proyecto ubicados en la carpeta raíz, ejecutando el siguiente comando.

```
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django>django-admin startproject GestionNegocio ←
```



Al ejecutarse se crea una Carpeta con ese Nombre y dentro de ella también se crea una carpeta con el mismo nombre y un archivo **manage.py**. Dentro de la subcarpeta se crean varios archivos como se muestra en la siguiente imagen.



Después vamos ir conociendo para que sirve cada uno de los archivos.

Ahora vamos a crear una aplicación en el proyecto, para ello vamos a ejecutar el siguiente comando verificando que estemos en la carpeta raíz **GestiónNegocio**.

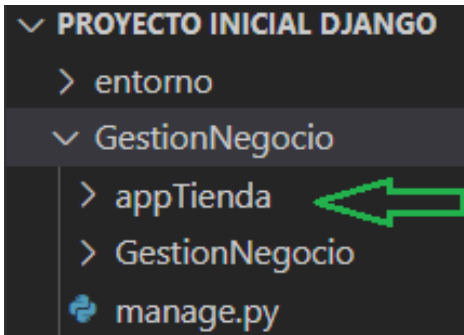
```
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django\GestionNegocio>
```

Ahora escribimos el siguiente comando para crear una aplicación llamada **appTienda**:

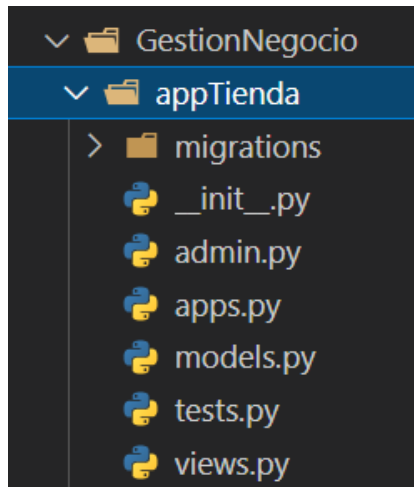
```
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django\GestionNegocio>python manage.py startapp appTienda
```



Al ejecutar dicho comando se creó una carpeta con el nombre de la aplicación appTienda así:

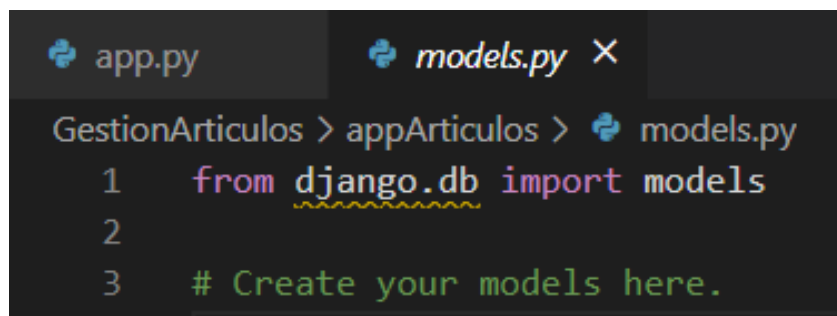


La carpeta appTienda contiene una carpeta llamada **migrations** y unos archivos como se muestra en la siguiente imagen:



Vamos ahora a proceder a crear las clases en el modelo para con ellas después crear la base de datos.

Abrimos el archivo **models.py**:

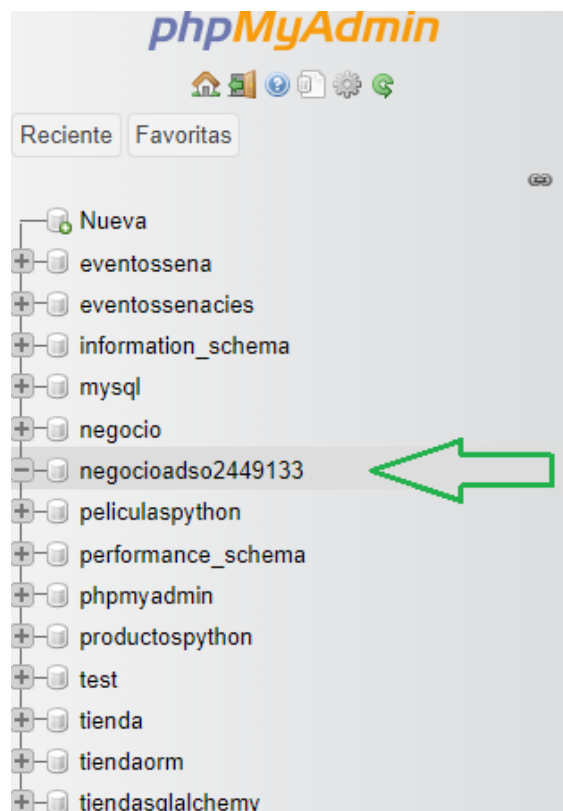




Nuestro modelo con las clases quedaría así:

```
GestionNegocio > appTienda > models.py > ...
1  from django.db import models
2
3  # Create your models here.
4
5  class Categoria(models.Model):
6      catNombre = models.CharField(max_length=50, unique=True)
7
8  class Producto(models.Model):
9      proCodigo = models.IntegerField(unique=True)
10     proNombre = models.CharField(max_length=50)
11     proPrecio = models.IntegerField()
12     proCategoria = models.ForeignKey(Categoria, on_delete=models.PROTECT)
13     proFoto = models.FileField(upload_to=f"fotos/", null=True, blank=True)
14
```

Como se requiere que desde django se cree las tablas en la base de datos, entonces primero vamos a ir a mysql y creamos una base de datos llamada **negocioadso2449133** sin tablas.





Ahora abrimos el archivo llamado **settings.py** y modificamos lo relacionado con lo de la base de datos así:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql',  
        'NAME': 'negocioadso2449133',  
        'USER': 'root',  
        'PASSWORD': '',  
        'HOST': 'localhost',  
        'PORT': '3306'  
    }  
}
```

Y en el mismo archivo registramos la aplicación creada llamada appTienda en el bloque **INSTALLED\_APPS** como se muestra en la imagen.

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

Ahora se debe instalar **mysqlclient** en el entorno así:

```
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django\GestionNegocio>pip install mysqlclient  
Collecting mysqlclient  
  Downloading mysqlclient-2.1.1-cp311-cp311-win_amd64.whl (178 kB)  
    178.4/178.4 kB 3.6 MB/s eta 0:00:00  
Installing collected packages: mysqlclient  
Successfully installed mysqlclient-2.1.1  
  
[notice] A new release of pip available: 22.3.1 -> 23.1  
[notice] To update, run: python.exe -m pip install --upgrade pip  
  
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django\GestionNegocio>
```



El siguiente paso sería crear las migraciones para poder después crear las tablas en la base de datos. Para crear las migraciones ejecutar el siguiente comando:

```
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django\GestionNegocio>python manage.py makemigrations ←
Migrations for 'appTienda':
  appTienda\migrations\0001_initial.py
    - Create model Categoria
    - Create model Producto

(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django\GestionNegocio>
```

Con el comando anterior **django** creó una migración con el código 0001. Dentro de la carpeta de la aplicación hay una carpeta llamada **migrations** y ahí podemos ver el archivo generado **0001\_initial.py** así:

```
GestionNegocio > appTienda > migrations > 0001_initial.py > ...
1  # Generated by Django 4.2 on 2023-04-16 15:30
2
3  from django.db import migrations, models
4  import django.db.models.deletion
5
6  class Migration(migrations.Migration):
7      initial = True
8      dependencies = [
9
10     ]
11     operations = [
12         migrations.CreateModel(
13             name='Categoria',
14             fields=[
15                 ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
16                 ('catNombre', models.CharField(max_length=50, unique=True)),
17             ],
18         ),
19         migrations.CreateModel(
20             name='Producto',
21             fields=[
22                 ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
23                 ('proCodigo', models.IntegerField(unique=True)),
24                 ('proNombre', models.CharField(max_length=50)),
25                 ('proPrecio', models.IntegerField()),
26                 ('proFoto', models.FileField(blank=True, null=True, upload_to='fotos/')),
27                 ('proCategoria', models.ForeignKey(on_delete=django.db.models.deletion.PROTECT, to='appTienda.categoria')),
28             ],
29         ),
30     ]
```

El archivo de la migración si contiene como va a llamar los campos llave primaria que por defecto los llamada id. A medida que se vayan realizando cambios y se vuelva a ejecutar makemigrations se van creand archivos nuevos que incluyen los cambios.

Y para terminar de crear las tablas en la base de datos ejecutamos el siguiente comando:





```
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django\GestionNegocio>python manage.py migrate
```

System check identified some issues:

WARNINGS:

?: (mysql.W002) MariaDB Strict Mode is not set for database connection 'default'

HINT: MariaDB's Strict Mode fixes many data integrity problems in MariaDB, such as data truncation upon insertion, by escalating warnings into errors. It is strongly recommended you activate it. See: <http://s://docs.djangoproject.com/en/4.2/ref/databases/#mysql-sql-mode>

Operations to perform:

Apply all migrations: admin, appTienda, auth, contenttypes, sessions

Running migrations:

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 appTienda.0001\_initial... 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

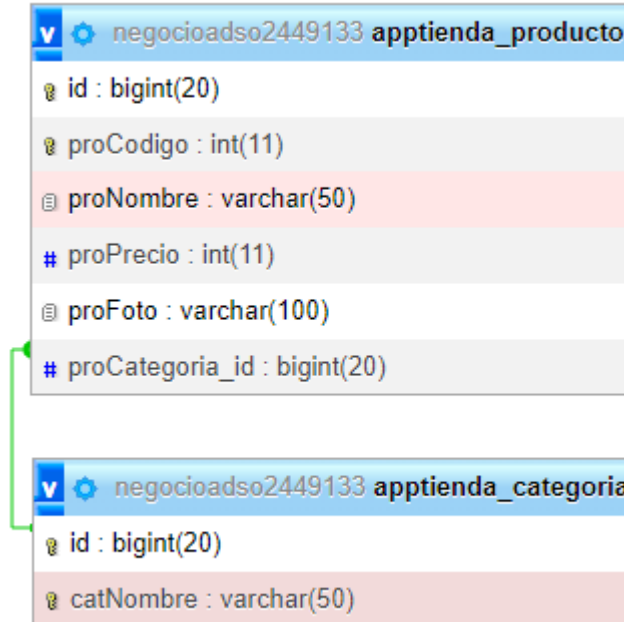
Applying auth.0011\_update\_proxy\_permissions... OK

Applying auth.0012\_alter\_user\_first\_name\_max\_length... OK

Applying sessions.0001\_initial... OK

Si vamos a mysql podemos ver que se crearon las dos tablas y adicionalmente otras tablas que utiliza **django** para administrar la aplicación:

Tabla	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
<input type="checkbox"/> apptienda_categoria	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_spanish_ci	32.0 KB	-
<input type="checkbox"/> apptienda_producto	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_spanish_ci	48.0 KB	-
<input type="checkbox"/> auth_group	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_spanish_ci	32.0 KB	-
<input type="checkbox"/> auth_group_permissions	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_spanish_ci	48.0 KB	-
<input type="checkbox"/> auth_permission	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	32	InnoDB	utf8_spanish_ci	32.0 KB	-
<input type="checkbox"/> auth_user	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_spanish_ci	32.0 KB	-
<input type="checkbox"/> auth_user_groups	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_spanish_ci	48.0 KB	-
<input type="checkbox"/> auth_user_user_permissions	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_spanish_ci	48.0 KB	-
<input type="checkbox"/> django_admin_log	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_spanish_ci	48.0 KB	-
<input type="checkbox"/> django_content_type	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	8	InnoDB	utf8_spanish_ci	48.0 KB	-
<input type="checkbox"/> django_migrations	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	19	InnoDB	utf8_spanish_ci	16.0 KB	-
<input type="checkbox"/> django_session	★ Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_spanish_ci	32.0 KB	-
12 tablas	Número de filas	59	InnoDB	utf8_spanish_ci	464.0 KB	0 B



Ahora vamos a crear dentro de la carpeta **appTienda** una carpeta llamada *templates* para colocar aquí los archivos html que se vayan a crear.

La ruta de la carpeta se debe registrar en el archivo **settings.py** en el bloque **TEMPLATES** como se muestra en la siguiente imagen:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['./appTienda/templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```



## Configuración de los templates y rutas

Aquí utilizar los htmls creados en proyecto anterior de la tienda utilizando Flask SQLAlchemy.

Se recomienda también crear la carpeta **static** dentro de la aplicación **appTienda**, y dentro de ella crear las otras carpetas como **css**, **js**, **imagenes** entre otros. Estas carpetas y archivos también se pueden copiar del proyecto creado con flask SQLAlchemy.

Debemos configurar en el archivo settings.py las siguientes variables que me permiten indicar cuál es la carpeta donde se van a ubicar los archivos estáticos.

Importar la librería os

```
from pathlib import Path
import os
```

Crear la variable **STATIC\_URL**

```
STATIC_URL = os.path.join(BASE_DIR, '/static/')
```

Para empezar vamos a crear una función que me represente la vista inicial en el archivo **views.py** que se encuentra dentro de la carpeta **appTienda**.

```
GestionNegocio > appTienda > views.py > inicio
1  from django.shortcuts import render
2
3  # Create your views here.
4
5  def inicio(request):
6      return render(request, "inicio.html")
```

Siempre que se vaya a crear una vista como la anterior se crea una función que recibe como parámetro un **request**. En el ejemplo anterior la vista muestra el html del archivo inicio.html.



Ahora debemos registrar la vista en el archivo llamado **urls.py** así:

```
17 from django.contrib import admin
18 from django.urls import path
19 from appTienda import views
20
21 urlpatterns = [
22     path('admin/', admin.site.urls),
23     path('inicio/', views.inicio),
24 ]
```

El archivo **urls.py** ya tiene una vista registrada para el admin y ahora agregamos la del inicio. La primera parte del path no necesariamente debe ser igual a la función, pero el segundo parámetro **views.inicio** si hace referencia al nombre de la función. Recordemos que en **Flask** la primera parte es como el decorador.

No olvidar hacemos el **import** señalado en la imagen.

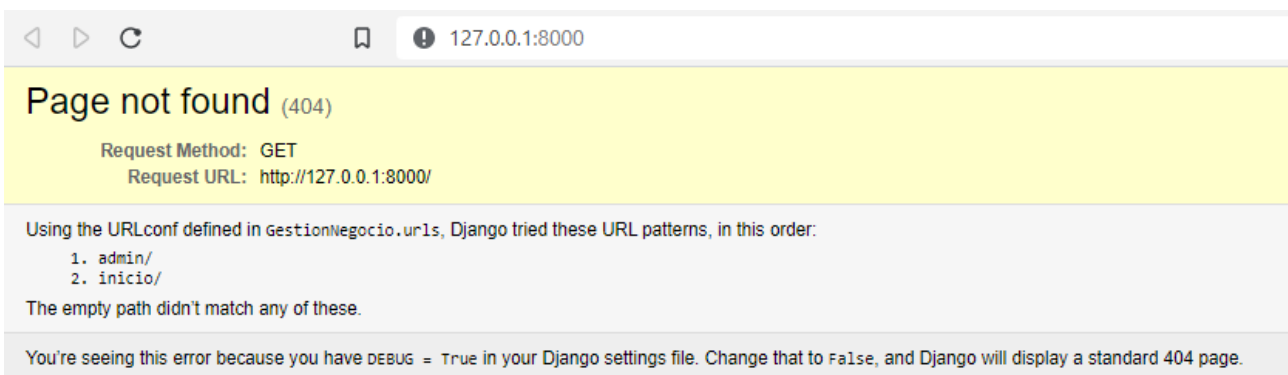
Ahora podemos probar ejecutando el servidor para ver si está funcionando.

En la consola ejecutamos el siguiente comando verificando que estemos en la carpeta donde se encuentra el archivo **manage.py**

```
(entorno) C:\Users\César Cuéllar\Documents\SENA\2023\FICHAS\Ficha 2449133\Actividades\2023\Abril 17 de 2023\Proyecto Inicial Django\GestionNegocio>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 16, 2023 - 10:59:36
Django version 4.2, using settings 'GestionNegocio.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

La respuesta nos dice que se ha lanzado el servidor web en el **puerto 8000**





Ahora procedemos a colocar la ruta **inicio/**



Continuamos con el html que nos permite mostrar la vista del formulario categorías así:

- Creamos una función en el archivo **views.py** para mostrar formulario categorías y que retorne al html de **frmCategoria.html**
- Registramos la función en el archivo **urls.py**
- Modificamos el menú de opciones para que al dar clic en categorías llame a la ruta creada.

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

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('inicio/', views.inicio),  
    path('vistaCategorias/', views.vistaCategorias),  
]
```



Ahora vamos a realizar el proceso de agregar una categoría así:

A continuación se muestra el código html del formulario que tiene una acción con una ruta `/agregarCategoria` y método `post`. También vemos una línea señalada que se utiliza para cuando se envían datos por método `post` por seguridad. `{% csrf_token %}`

```
<form action="/agregarCategoria/" method="post">
    {% csrf_token %}
    <div>
        <h3 class="text-center bg-danger text-white">AGREGAR CATEGORÍA</h3>
    </div>
    <div class="form-floating">
        <input type="text" name="txtNombre" id="txtNombre"
            class="form-control" required>
        <label for="txtNombre">Nombre Categoría:</label>
    </div>
    <div>
        <button class="btn btn-success mt-3">Agregar</button>
    </div>
</form>
```

Crear la función en **views.py** que permita hacer el proceso de agregar la categoría.

Primero importar el modelo `Categoria` así:

```
from appTienda.models import Categoria
```

La función **agregarCategoria** en **views.py**

```
def agregarCategoria(request):
    nombre = request.POST["txtNombre"]
    try:
        categoria = Categoria(catNombre=nombre)
        categoria.save()
        mensaje="Categoria agregada correctamente"
    except:
        mensaje="Problemas a la hora de agregar la categoría"
    retorno = {"mensaje":mensaje}
    return render(request,"frmCategoria.html",retorno)
```



El paso siguiente es registrar la función en el archivo urls.py.

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('inicio/', views.inicio),  
    path('vistaCategorias/', views.vistaCategorias),  
    path('agregarCategoria/', views.agregarCategoria),  
]
```

### Proceso Listar Productos

- Crear una función en el archivo views.py para mostrar los productos registrados.
- Registrar la función en el archivo urls.py
- Modificar los html necesarios

```
def listarProductos(request):  
    try:  
        productos = Producto.objects.all()  
        mensaje=""  
        print(productos)  
    except:  
        mensaje="Problemas al obtener los productos"  
    retorno = {"mensaje":mensaje, "listaProductos":productos}  
    return render(request,"listarProductos.html",retorno)
```

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('inicio/', views.inicio),  
    path('vistaCategorias/', views.vistaCategorias),  
    path('agregarCategoria/', views.agregarCategoria),  
    path('listarProductos/', views.listarProductos),  
]
```



```
<tbody>
  {%for producto in listaProductos %}
    <tr>
      <td class="text-center">{{producto.proCodigo}}</td>
      <td>{{producto.proNombre}}</td>
      <td class="text-end">${{producto.proPrecio}}</td>
      <td>{{producto.proCategoria.catNombre}}</td>
      <td class="text-center">
        
      </td>
      <td class="text-center" style="font-size:4vh">
        <a href="/consultarProducto/{{producto.id}}">
          <i class="fa fa-edit text-warning"></i>
        </a>
        <i class="fa fa-trash text-danger"
          onclick="abrirModalEliminar('{{producto.id}}')">
        </i>
      </td>
    </tr>
  {% endfor %}
</tbody>
```

Para poder que se muestren las imágenes debemos tener acceso a la carpeta **media** donde se van a guardar las fotos de los productos en una carpeta llamada **fotos**.

Inicialmente hay que crear las siguientes variables en el archivo **settings.py**

```
#para almacenar archivos multimedia. Fotos
MEDIA_URL = '/media/'

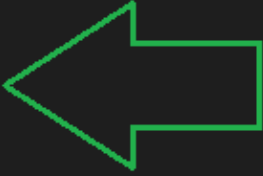

#Para recuperar los archivos multimedia
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```





Modificar el archivo **urls.py** para tener acceso a la carpeta donde se encuentran las imágenes.

```
17  from django.contrib import admin
18  from django.urls import path
19  from appTienda import views
20  from django.conf import settings
21  from django.conf.urls.static import static
22
23  urlpatterns = [
24      path('admin/', admin.site.urls),
25      path('inicio/', views.inicio),
26      path('vistaCategorias/', views.vistaCategorias),
27      path('agregarCategoria/', views.agregarCategoria),
28      path('listarProductos/', views.listarProductos),
29      path('vistaProducto/', views.vistaProducto),
30      path('agregarProducto/', views.agregarProducto),
31  ]
32
33  #para poder tener acceso a la carpeta media y poder ver las fotos
34  #de los productos
35  if settings.DEBUG:
36      urlpatterns += static(
37          settings.MEDIA_URL,
38          document_root = settings.MEDIA_ROOT
39      )
```





## Proceso para registrar un Producto:

- Crear una función en el archivo views.py para mostrar la vista del formulario.
- Crear la función que permita agregar un producto a la base de datos
- Registrar las funciones en el archivo urls.py
- Modificar los html necesarios

```
def vistaProducto(request):
    try:
        categorias = Categoria.objects.all()
        mensaje=""
    except:
        mensaje="Problemas al obtener las categorias"
    retorno = {"mensaje":mensaje, "listaCategorias":categorias, "producto":None}
    return render(request,"frmRegistrarProducto.html",retorno)
```


Ahora miremos en el código html como se reciben los datos del producto si tiene datos y de las categorías ya existentes:


```
<form action="/agregarProducto/" method="post" enctype="multipart/form-data" class="was-validated">
  <div class="form-floating mb-3">
    <input type="number" name="txtCodigo" id="txtCodigo"
    class="form-control" value="{{producto.proCodigo}}" required>
    <label for="txtCodigo">Código:</label>
  </div>
  <div class="form-floating mb-3">
    <input type="text" name="txtNombre" id="txtNombre"
    class="form-control" value="{{producto.proNombre}}" required>
    <label for="txtNombre">Nombre:</label>
  </div>
  <div class="form-floating mb-3">
    <input type="number" name="txtPrecio" id="txtPrecio"
    class="form-control" value="{{producto.proPrecio}}" required>
    <label for="txtPrecio">Precio:</label>
  </div>
  <div class="form-floating mb-3">
    <select name="cbCategoria" id="cbCategoria" class="form-select" required>
      <option value="">Selecione</option>
      {% for categoria in listaCategorias %}
        <option value="{{categoria.id}}">{{categoria.catNombre}}</option>
      {% endfor %}
    </select>
    <label for="cbCategoria">Categoria:</label>
  </div>
</form>
```






Y en la vista se puede ya seleccionar la categoría:

**REGISTRAR PRODUCTO**

Código: 

Nombre: 

Precio: 

Categoría:  

Seleccione

Seleccione

Calzado

muebles

otraaaaaaaaaa

Ropa

Registrar

Cancelar

Derechos Reservados Aprendices ficha ADSO 2449133

Tecnólogo en Análisis y Desarrollo de Software



Ahora vamos a crear función que nos permite agregar un producto a la base de datos:

```
def agregarProducto(request):
    nombre = request.POST["txtNombre"]
    codigo = int(request.POST["txtCodigo"])
    precio = int(request.POST["txtPrecio"])
    idCategoria = int(request.POST["cbCategoria"])
    archivo = request.FILES["fileFoto"]
    try:
        #obtener la categoria de acuerdo a su id
        categoria = Categoria.objects.get(id=idCategoria)
        #crear el producto
        producto = Producto(proNombre = nombre,proCodigo=codigo,
                             proPrecio=precio, proCategoria=categoria,
                             proFoto = archivo)
        #registrarlo en la base de datos
        producto.save()
        mensaje="Producto Agregado Correctamente"
        return redirect("/listarProductos/")
    except Error as error:
        mensaje=f"Problemas al realizar el proceso de agregar un producto. {error}"

    #obtener las categorias
    categorias = Categoria.objects.all()
    retorno = {"mensaje":mensaje, "listaCategorias":categorias, "producto":producto}
    return render(request,"frmRegistrarProducto.html",retorno)
```

En la función anterior ya se está validando las excepciones que puedan ocurrir al realizar tareas en la base de datos para ello se debe importar como se muestra en la imagen. Adicionalmente también se utiliza la función redirect que permite redireccionar a otra vista.

```
GestionNegocio > appTienda > views.py > inicio
1  from django.shortcuts import render, redirect
2  from appTienda.models import Categoria,Producto
3  from django.db import Error
4
```

Y después registramos la vista en el archivo **urls.py**:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('inicio/',views.inicio),
    path('vistaCategorias/',views.vistaCategorias),
    path('agregarCategoria/',views.agregarCategoria),
    path('listarProductos/',views.listarProductos),
    path('vistaProducto/',views.vistaProducto),
    path('agregarProducto/',views.agregarProducto),
]
```



Modificar el archivo **index.html** para cargar los archivos estáticos así:

```
{% load static%}  
<link rel="stylesheet" href="{% static '../static/css/app.css' %}">  
<script src="{% static '../static/js/app.js' %}"></script>
```

Terminar las otras tareas como actualizar y eliminar.

**Nota:** Realizar todas las validaciones necesarias para el buen funcionamiento.

Recomiendo ingresar a la documentación y revisar todas los tipos de consulta posibles. El material de apoyo es un ejercicio muy básico que espero les aclare un poco como crear un proyecto y una aplicación web en Python utilizando el framework Django.

### Referencias:

1. Sitio oficial Framework Django: <https://www.djangoproject.com/>
2. Documentación oficial Django: <https://docs.djangoproject.com/es/4.1/>
3. Tutorial en la web: <https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introduction>
4. Qué es django: <https://aws.amazon.com/es/what-is/django/>
5. Tutorial Django: <https://www.w3schools.com/django/index.php>
6. Material Django <https://runebook.dev/es/docs/django/-index->
7. EDT SENA. Fernando Galindo: <https://siomi.datasena.com/EDTFullstack/django.html>

### CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
<b>Autor (es)</b>	César Marino Cuéllar Chacón	Instructor	CIES-NEIVA	17-04-2023