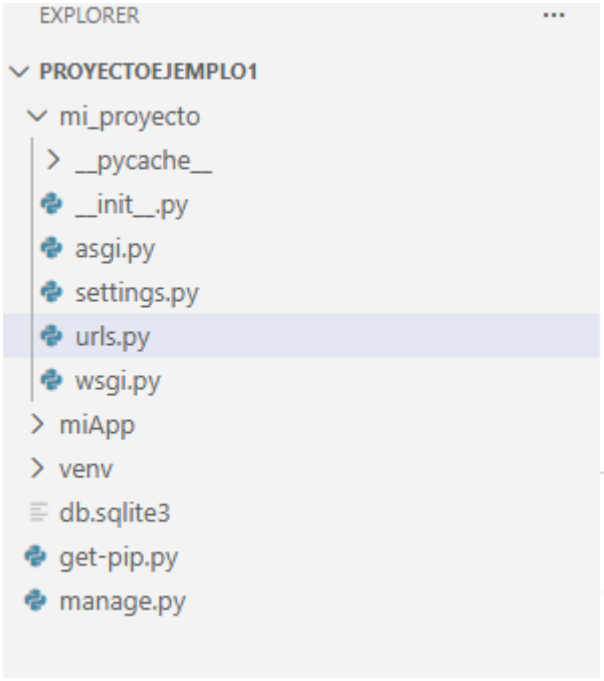
Django



VIEWS

1º Crear un archivo que contenga las vistas que vamos almacenando.

2º- Revisamos qué parámetros de entrada recibe esa vista y después qué es lo que nos va a devolver.

3º-Crear la URL para decirle al navegador que vista le corresponde

```
from django.contrib import admin
from django.urls import path
from mi_proyecto.views import encender
from mi_proyecto.views import apagar

urlpatterns = [
    path('admin/', admin.site.urls),
    path('encender/', encender),
    path('apagar/', apagar),

]
```

```
from django.http import HttpResponse

#Declaramos una función, recibe una petición y devuelve una respuesta
def saludar(Httprequest): #primera vista
    return HttpResponse("Buenas noches!")
```

Dentro de lo que es el proyecto pueden haber muchas aplicaciones, con el comando python manage.py startapp miApp logramos crear una estructura nueva de una aplicación.

```
myproject/
    manage.py
    myproject/
    myapp/
        admin.py
        apps.py
        models.py
        tests.py
        views.py
        __init__.py
        migrations/
```

Se puede observar que tiene los modelos, las vistas y demás configuración.

4º- En la configuración del proyecto también le podemos agregar la aplicación nueva que hemos creado:

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

Esta App será la nueva aplicación.

En esta App podemos meter también vistas. Aquí irá la lógica de la aplicación. Interactuar con la BD mediante el modelo para posteriormente interactuar con la plantilla y de esta manera poder generar algo bonito que visualizar.

Dentro de la aplicación se pueden crear las vistas, las cuales, se van a lincar a nuestro proyecto posteriormente.

miApp > views.py

```
1  from django.shortcuts import render
2
3  # Create your views here.
4  from django.http import HttpResponse
5
6  def index(request):
7      return HttpResponse("Listado de departamentos de clientes")
```

Una vez estamos aquí en nuestra app debemos decirle a Django que podamos asociar la vista que estamos construyendo a una URL. Por lo tanto hay que tener un archivo urls.py dentro de nuestra aplicación la cual vamos a lincar también. Se importa la función path que se usa para definir las rutas de Django. Importa el archivo views.py del mismo directorio. Se define una lista llamada urlpatterns donde django busca las rutas disponibles.
aquí:

path(‘’,views.index,name=’index’)
‘’ → Es la ruta vacía, la página principal (localhost:8000/)
views.index→ Es la función index dentro de views.py que se ejecutará cuando alguien entre a esa url.
name=’index’ → Le da un nombre a la ruta. Sirve para referenciarla en plantillas o con redirect().

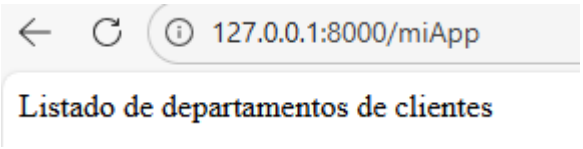
miApp > urls.py

```
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path ('', views.index, name = 'index')
6  ]
```

Ahora en este punto a nuestra app le falta enlazarla con el proyecto. Por lo que necesitamos enlazarla en nuestro archivo de URLS.py del proyecto principal. La función include lo que hace es incluir todas las urls que están en nuestra app.

```
17  from django.contrib import admin
18  from django.urls import path, include
19  from mi_proyecto.views import encender
20  from mi_proyecto.views import apagar
21
22  urlpatterns = [
23      path('admin/', admin.site.urls),
24      path('encender/', encender),
25      path('apagar/', apagar),
26      path('miApp', include('miApp.urls')),
27
28  ]
```

De esta manera le estamos diciendo a python que asocie la aplicación al proyecto. Y podemos ir a la url de localhost:800/miApp



MODELS

Las aplicaciones web desarrolladas con Django acceden y administran datos a través de objetos estos objetos se conocen como modelos y se encargan de definir la estructura de nuestros datos por ejemplo se definen normalmente en **models.py** **Se definen como subclases que heredan de la clase Model: django.db.models.Model y** representan tablas en la base de datos, cada atributo del modelo representa una columna.

```
from django.db import models

class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    creado = models.DateTimeField(auto_now_add=True)

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

Al heredar de **models.Model** django entiende que esta clase es un modelo y debe convertirse a una tabla de la base de datos y dentro de esta base de datos las columnas serán los atributos de la clase.

¿Qué significa Charfield?

Campo de caracteres, es decir, un texto corto con longitud limitada. Se utiliza cuando quieres guardar cadenas de texto como:

- nombres
- apellidos
- códigos
- estados
- ciudades
- títulos

Entonces… ¿solo existe CharField?

Nop! También existe muchos más tipos de campos:

Campos de texto como pueden ser:

```
models.CharField(max_length=100)      # Texto corto (VARCHAR)
models.TextField()                    # Texto largo (TEXT)
models.EmailField()                   # Email con validación
models.URLField()                     # URLs
models.SlugField()                    # Texto para URLs amigables
```

Campos numéricos:

```
models.IntegerField()           # INT
models.BigIntegerField()        # BIGINT
models.PositiveIntegerField()   # INT positivo
models.FloatField()             # FLOAT
models.DecimalField()           # DECIMAL (dinero)
```

Booleanos

```
models.BooleanField()           # TRUE / FALSE
```

Fechas

```
models.DateField()              # DATE
models.DateTimeField()          # DATETIME
models.TimeField()              # TIME
```

Relaciones

models.ForeignKey()
models.ManyToManyField()
models.OneToOneField()

Archivos

models.FileField()
models.ImageField()

¿Cuándo utilizar cada uno? Depende de la situación.

```python
class Ejemplo(models.Model):
    nombre = models.CharField(max_length=50)    # Texto corto
    descripcion = models.TextField()            # Texto largo
    edad = models.IntegerField()                # Número entero
    activo = models.BooleanField(default=True) # Verdadero/Falso
```

Django mapea la definición de nuestros modelos contra la base de datos.

Creamos el contenido de nuestro modelo.

```python
from django.db import models

# Create your models here.

class Categoria(models.Model):
    nombre = models.CharField(max_length=100)

    def __str__(self):
        return self.nombre


class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.IntegerField(default=0)
    categoria = models.ForeignKey( ##Creamos una relación de muchos a uno
        Categoria,
        on_delete=models.CASCADE, ##Si se elimina la categoría, se eliminan sus productos.
        related_name="productos"
    )
    creado = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.nombre
    #max length=numero es obligatorio por que las BD neceistan saber el tamaño del campo
```

Mediante los comandos makemigratios y migrate podemos plasmar todo lo que hemos puesto en nuestros modelos en la BD.

python manage.py makemigrations miApp

```
PS C:\Users\HEZITZAILE\Desktop\SGE\Django\ProyectoEjemplo1> python manage.py makemigrations miApp
Migrations for 'miApp':
  miApp\migrations\0001_initial.py
    + Create model Categoria
    + Create model Cliente
    + Create model Curso
    + Create model Estudiante
    + Create model Pedido
    + Create model Producto
    + Create model DetallePedido
```

python manage.py makemigrations miApp

```
PS C:\Users\HEZITZAILE\Desktop\SGE\Django\ProyectoEjemplo1> python manage.py makemigrations miApp
Migrations for 'miApp':
  miApp\migrations\0001_initial.py
    + Create model Categoria
    + Create model Cliente
  miApp\migrations\0001_initial.py
    + Create model Categoria
    + Create model Cliente
    + Create model Categoria
    + Create model Cliente
    + Create model Cliente
    + Create model Curso
    + Create model Estudiante
    + Create model Pedido
    + Create model Producto
    + Create model DetallePedido
PS C:\Users\HEZITZAILE\Desktop\SGE\Django\ProyectoEjemplo1> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, miApp, sessions
Running migrations:
  Applying miApp.0001_initial... OK
PS C:\Users\HEZITZAILE\Desktop\SGE\Django\ProyectoEjemplo1>
```

De esta manera se ejecuta la creación de tablas, alter tables, inserts… Con python manage.py shell podemos interactuar con nuestra base de datos.Tenemos una consola dentro de nuestra aplicación. Para poder utilizar los objetos del model hay que persistir los objetos en la BD.

```python
from myapp.models import Categoria, Producto, Cliente, Pedido, DetallePedido

# Crear categoría
c = Categoria.objects.create(nombre="Electrónica")

# Crear producto
p = Producto.objects.create(
    nombre="Laptop",
    precio=1200.50,
    stock=10,
    categoria=c
)
```

Dentro del shell, importa tus modelos:

from miApp.models import Modelo1, Modelo2, …

CREACIÓN
```
# Crear un objeto y guardarlo automáticamente
obj = Modelo.objects.create(campo1="valor", campo2=123)

# Crear sin guardar
obj = Modelo(campo1="valor", campo2=123)
obj.save()  # Guardar en la DB
```

CONSULTA:
```
# Todos los registros
Modelo.objects.all()

# Filtrar por campos
Modelo.objects.filter(campo="valor")

# Obtener un solo objeto (id único)
Modelo.objects.get(id=1)

# Excluir ciertos registros
Modelo.objects.exclude(campo="valor")

# Ordenar resultados
Modelo.objects.order_by("campo")       # ascendente
Modelo.objects.order_by("-campo")      # descendente
```

UPDATE
```
# Cambiar atributos y guardar
obj = Modelo.objects.get(id=1)
obj.campo = "nuevo valor"
obj.save()

# Actualizar directamente con queryset
Modelo.objects.filter(id=1).update(campo="nuevo valor")
```

Relaciones:
```
# ForeignKey
# producto.categoria → accede a la categoría de un producto
# categoria.productos.all() → accede a todos los productos de una categoría
(related_name="productos")

# ManyToManyField
# estudiante.cursos.all() → todos los cursos de un estudiante
# curso.estudiante_set.all() → todos los estudiantes de un curso (nombre automático si no
hay related_name)
```