



Basado en Curso Django de Píldoras Informáticas:

<https://www.youtube.com/watch?v=7XO1AzwkPPE&list=PLU8oAIHdN5BmfwxFO7HdPciOCmmYneAB&index=2>



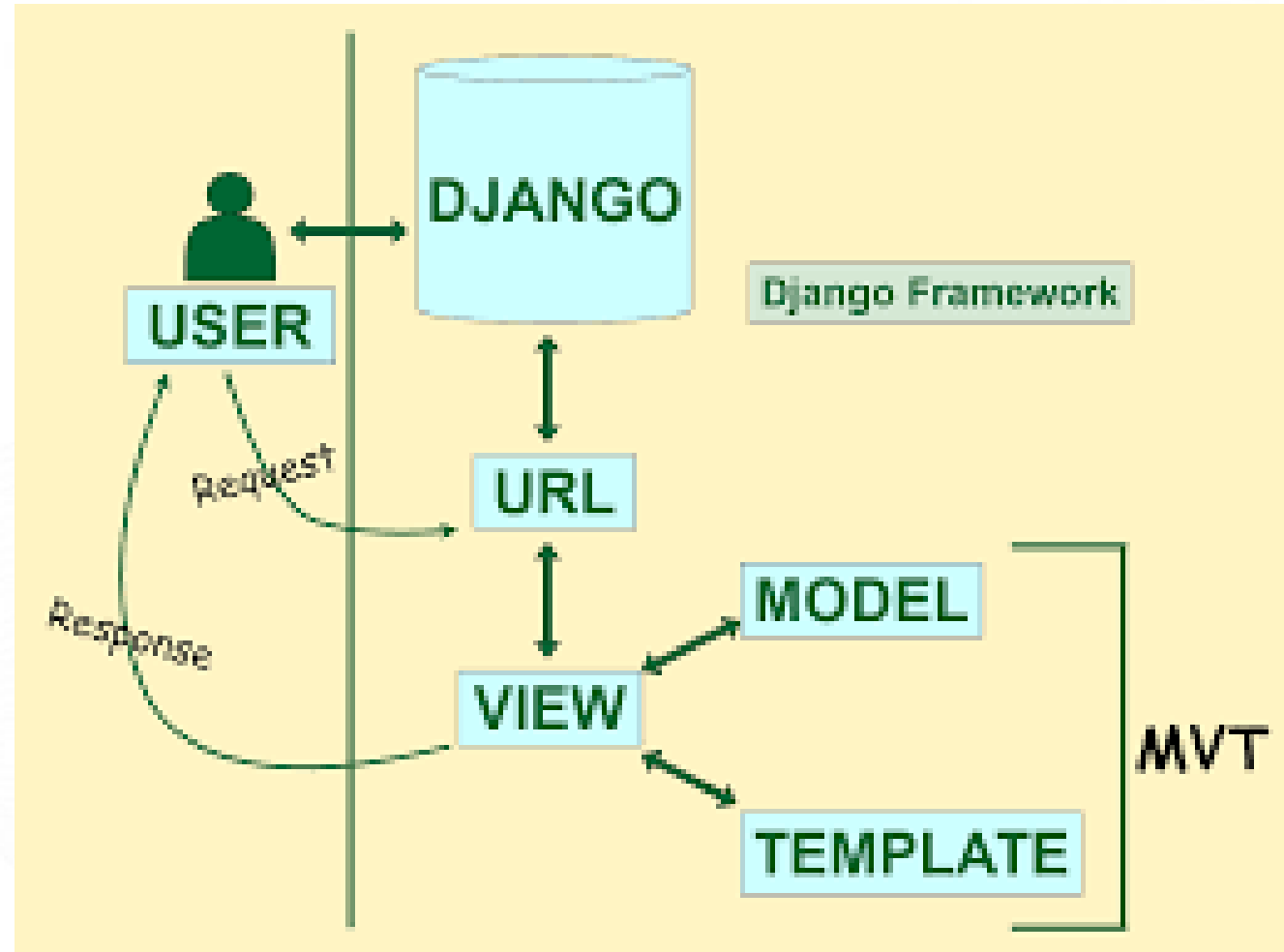
[www.sena.edu.co](http://www.sena.edu.co)



**Píldoras  
Informáticas**

Cursos de informática

# Django Base de Datos (Model)



# Base de Datos Soportadas por Django

Django incorpora compatibilidad directa con las siguientes bases de datos relacionales:

PostgreSQL,

MySQL,

Oracle,

SQLite

MariaDB.

En caso de querer usar otras bases de datos relacionales (como SQL Server) o no relacionales (como MongoDB) se debe hacer uso de librerías específicas para dicho propósito

# Proyectos vs Aplicación en Django

Proyecto Django: Es el conjunto global de configuración y aplicaciones que forman un sitio web. Incluye configuraciones generales, URLs principales, configuraciones de bases de datos, etc. Un proyecto puede contener múltiples aplicaciones.

Aplicación Django: Es un conjunto de funcionalidades relacionadas que se pueden reutilizar en diferentes proyectos. Una aplicación puede incluir modelos, vistas, URLconf, plantillas y archivos estáticos que trabajan juntos para realizar una función específica dentro del sitio web.

En resumen, el proyecto es el sitio web completo, mientras que una aplicación es una parte modular del mismo, diseñada para realizar una función específica.

# Proyectos vs Aplicación en Django

Proyecto Django: Es el conjunto global de configuración y aplicaciones que forman un sitio web. Incluye configuraciones generales, URLs principales, configuraciones de bases de datos, etc. Un proyecto puede contener múltiples aplicaciones.

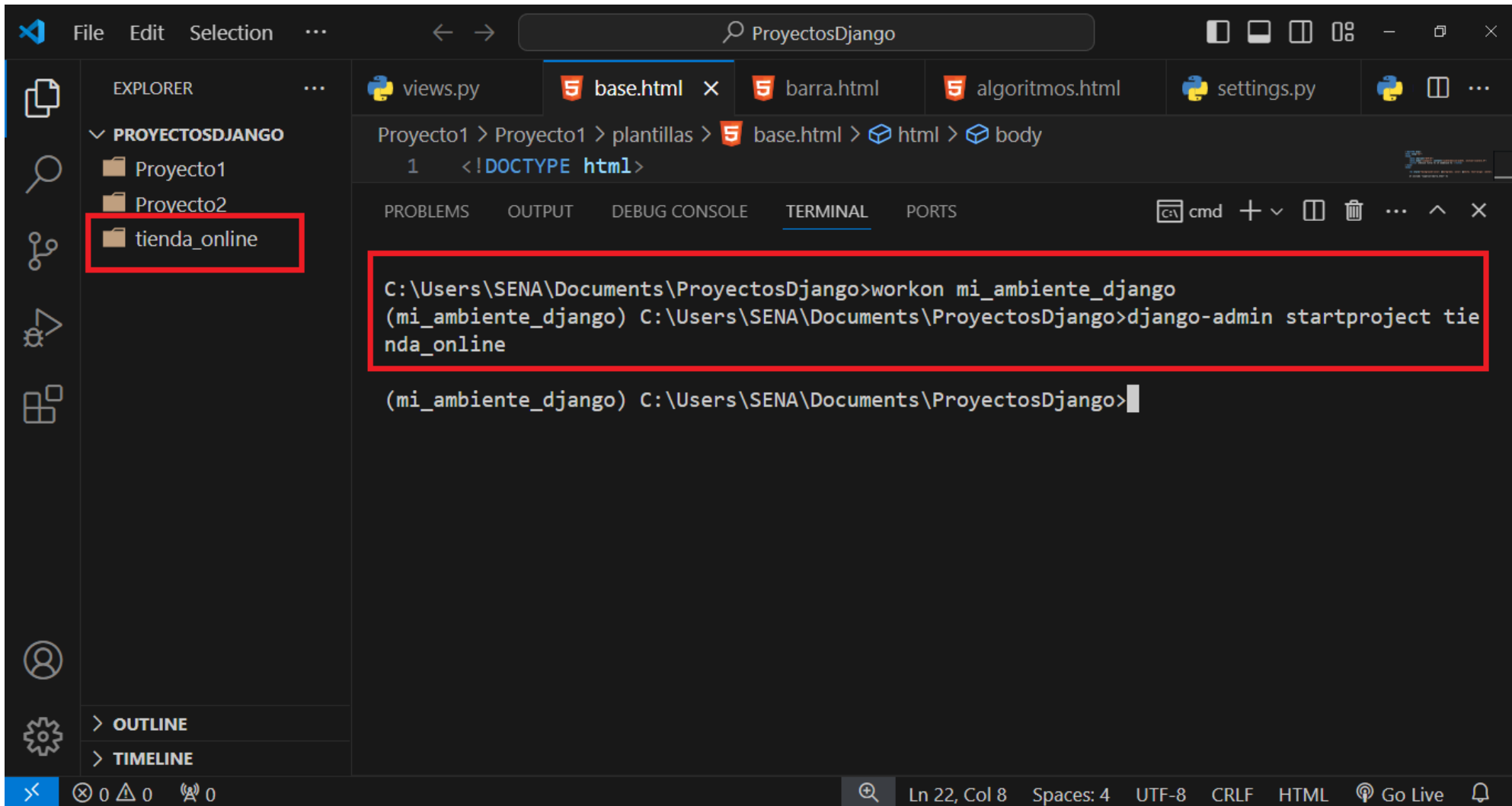
Aplicación Django: Es un conjunto de funcionalidades relacionadas que se pueden reutilizar en diferentes proyectos. Una aplicación puede incluir modelos, vistas, URLconf, plantillas y archivos estáticos que trabajan juntos para realizar una función específica dentro del sitio web.

En resumen, el proyecto es el sitio web completo, mientras que una aplicación es una parte modular del mismo, diseñada para realizar una función específica.

# Proyecto a Realizar: Base de Datos SQLite3



# Crear un nuevo proyecto



# Crear una aplicación

Para crear una aplicación seguiremos los siguientes pasos:

1. Utilizar el comando **django-admin startapp <nombre\_app>** en la terminal para crear una nueva aplicación Django.

Nota: Django generará una estructura de directorios y archivos básica para tu aplicación, incluyendo `models.py`, `views.py`, `urls.py`, y más.

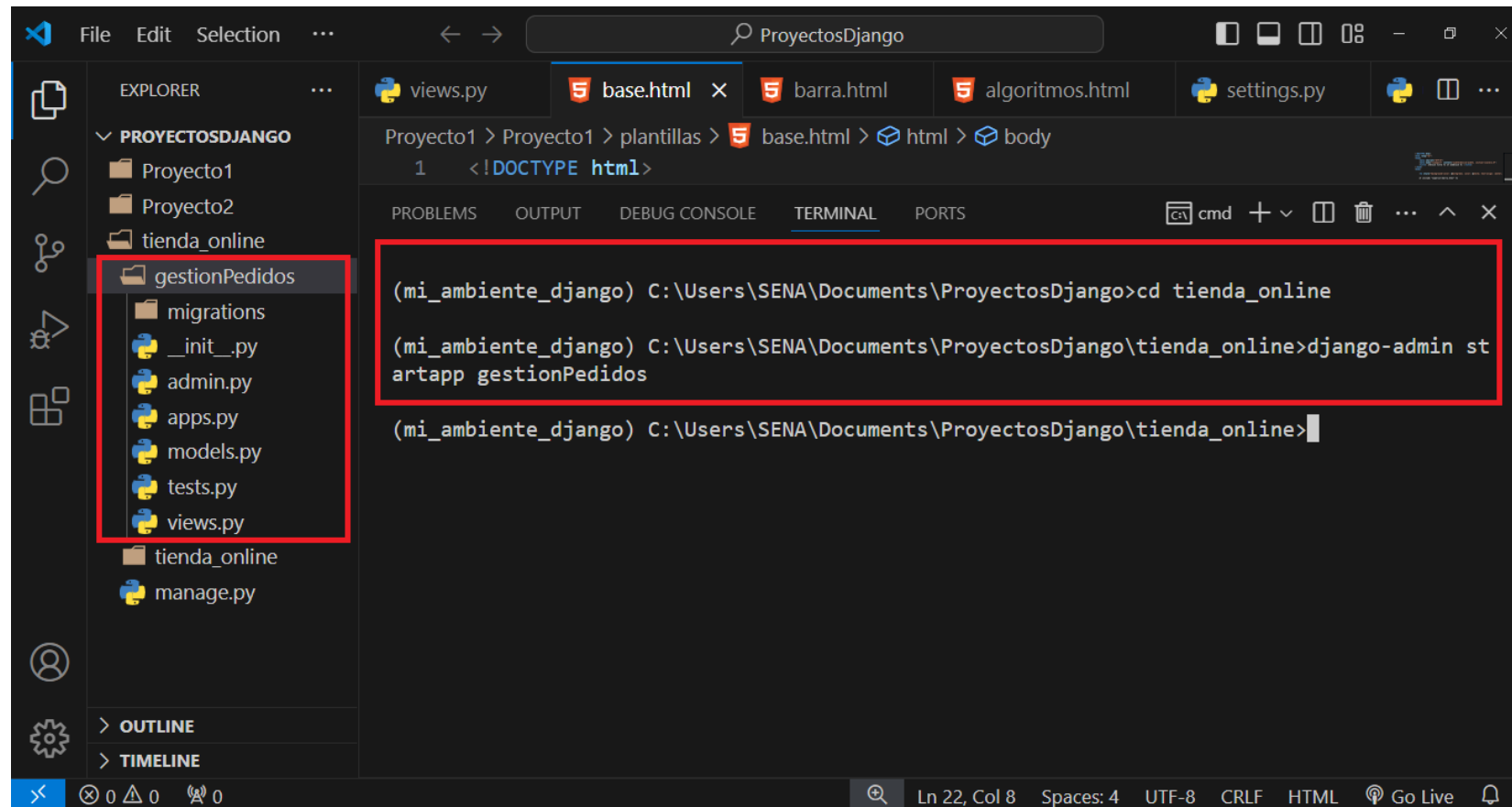
Define tus modelos en `models.py`, tus vistas en `views.py`, y las URL en `urls.py` para gestionar las solicitudes HTTP.

2. Agregar la aplicación al archivo `INSTALLED_APPS` en `settings.py` para que Django la reconozca.



# Crear una aplicación

Estando dentro del proyecto **tienda\_online** ejecutamos el comando:



The screenshot shows the Visual Studio Code interface with the Django project structure on the left and the terminal output on the right. The Explorer view shows the project structure with the `gestionPedidos` folder highlighted. The terminal shows the commands to create the application.

```

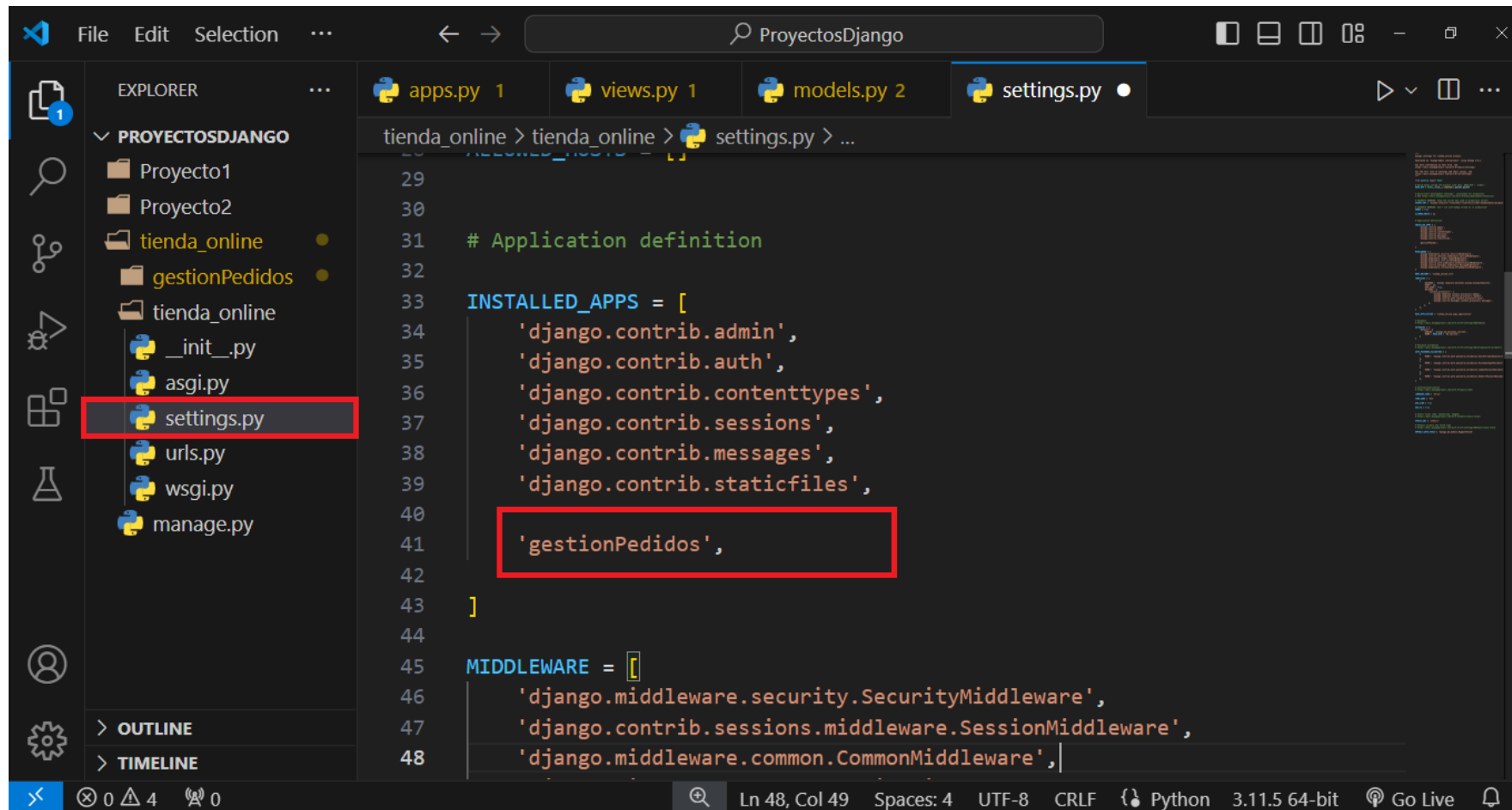
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango>cd tienda_online

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>django-admin st
artapp gestionPedidos

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>
  
```

# Registrar la aplicación

Vamos a **settings.py** en **INSTALLED\_APPS**



The screenshot shows the Visual Studio Code editor with a Django project named 'ProyectosDjango'. The Explorer sidebar on the left shows the project structure, with 'settings.py' highlighted under the 'tienda\_online' directory. The main editor window displays the 'settings.py' file, specifically the 'INSTALLED\_APPS' list. The list contains several Django default apps, and 'gestionPedidos' has been added at the end of the list, highlighted with a red box. The status bar at the bottom indicates the file is at line 48, column 49, using Python 3.11.5 64-bit.

```

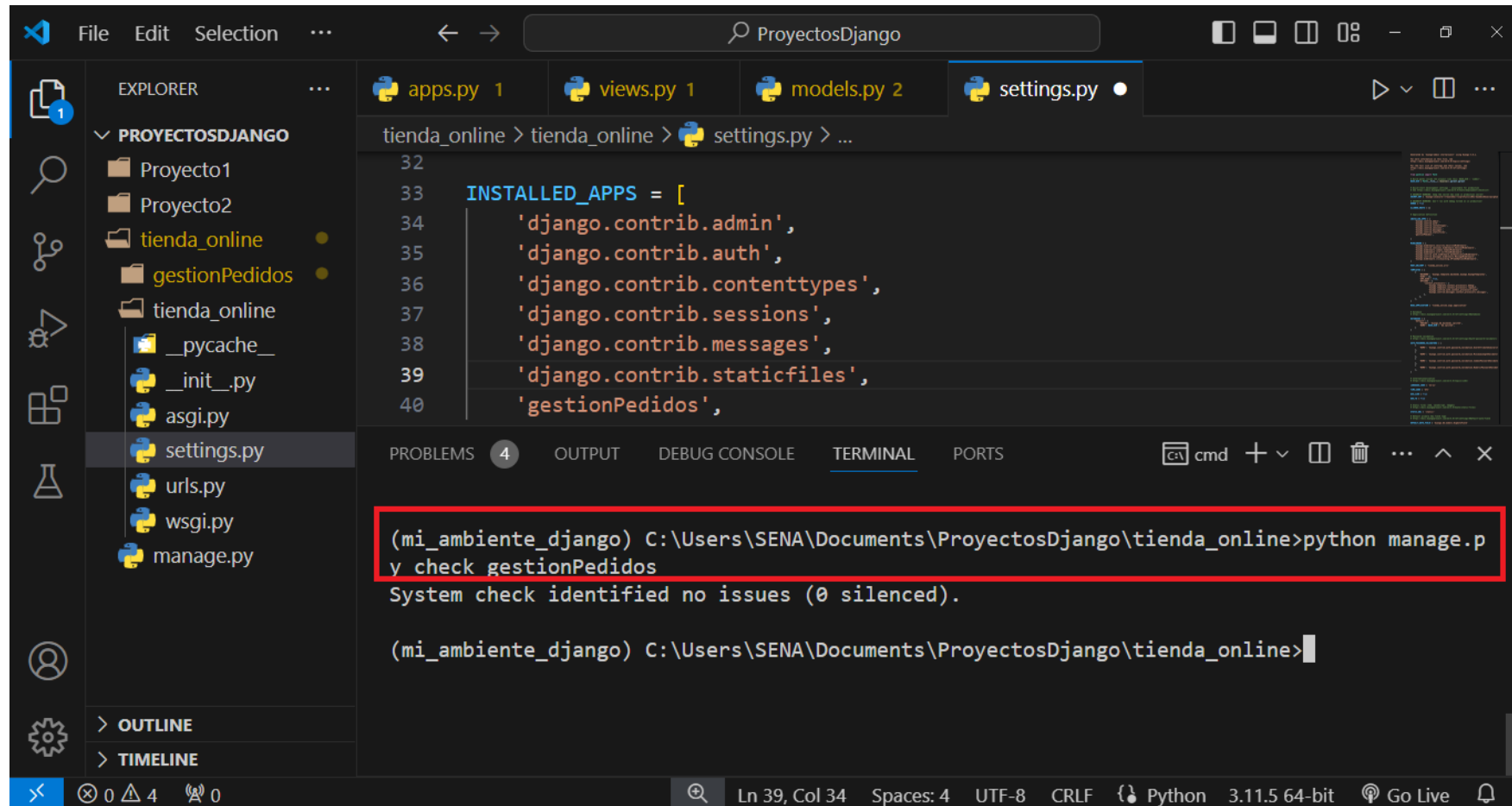
29
30
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40
41     'gestionPedidos',
42 ]
43
44
45 MIDDLEWARE = [
46     'django.middleware.security.SecurityMiddleware',
47     'django.contrib.sessions.middleware.SessionMiddleware',
48     'django.middleware.common.CommonMiddleware',

```

# Registrar la aplicación

Para verificar que todo marche correctamente utilizamos el comando:

**python manage.py check gestionPedidos**



The screenshot shows the Visual Studio Code interface with a Django project named 'ProyectosDjango'. The Explorer panel on the left shows the project structure, including a folder named 'tienda\_online' which contains a subfolder 'gestionPedidos'. The main editor window displays the 'settings.py' file, where the 'INSTALLED\_APPS' list is being edited. The list includes 'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles', and 'gestionPedidos'. The bottom panel shows the Terminal window with the command `(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.py check gestionPedidos` and the output `System check identified no issues (0 silenced).`

```
File Edit Selection ... < > ProyectosDjango
EXPLORER
  PROJECTOSDJANGO
    Proyecto1
    Proyecto2
    tienda_online
      gestionPedidos
      tienda_online
        __pycache__
        __init__.py
        asgi.py
        settings.py
        urls.py
        wsgi.py
        manage.py
  OUTLINE
  TIMELINE

apps.py 1 views.py 1 models.py 2 settings.py
tienda_online > tienda_online > settings.py > ...
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     'gestionPedidos',

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.p
y check gestionPedidos
System check identified no issues (0 silenced).

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>
```

# Creación de Modelos (tablas) en Django

Los modelos en Django son **clases** de Python que representan las tablas de la base de datos. Cada modelo define los campos de la tabla y sus tipos de datos correspondientes. Los modelos también pueden contener métodos que permiten realizar operaciones sobre los datos. En resumen, los modelos en Django proporcionan una forma de interactuar con la base de datos utilizando código Python en lugar de consultas SQL directas. Los pasos para trabajar con modelos en Django son:

1. Definir modelos: Se crean las clases de Python en el archivo **models.py** de la aplicación Django para representar las tablas de la base de datos.
2. Generar migraciones ó Crear Base de Datos: Se utiliza el comando **python manage.py makemigrations** para generar archivos de migración basados en los modelos definidos. Es decir, se crea la base de datos
3. Aplicar las migraciones: Se emplea el comando **python manage.py migrate** para aplicar las migraciones y crear las tablas en la base de datos SQLite3.

# Tipos de datos que se pueden definir en los modelos de Django

**CharField():** Un campo de texto corto, que generalmente se utiliza para cadenas de caracteres de longitud limitada.

**TextField():** Un campo de texto largo, que puede contener una cantidad considerable de texto.

**IntegerField():** Un campo para almacenar números enteros.

**FloatField():** Un campo para almacenar números de punto flotante.

**DecimalField():** Similar a FloatField, pero utilizado para precisión decimal exacta.

**BooleanField():** Un campo que almacena valores booleanos (True o False).

# Tipos de datos que se pueden definir en los modelos de Django

**DateField():** Un campo para almacenar fechas.

**DateTimeField():** Un campo para almacenar fechas y horas.

**TimeField():** Un campo para almacenar horas.

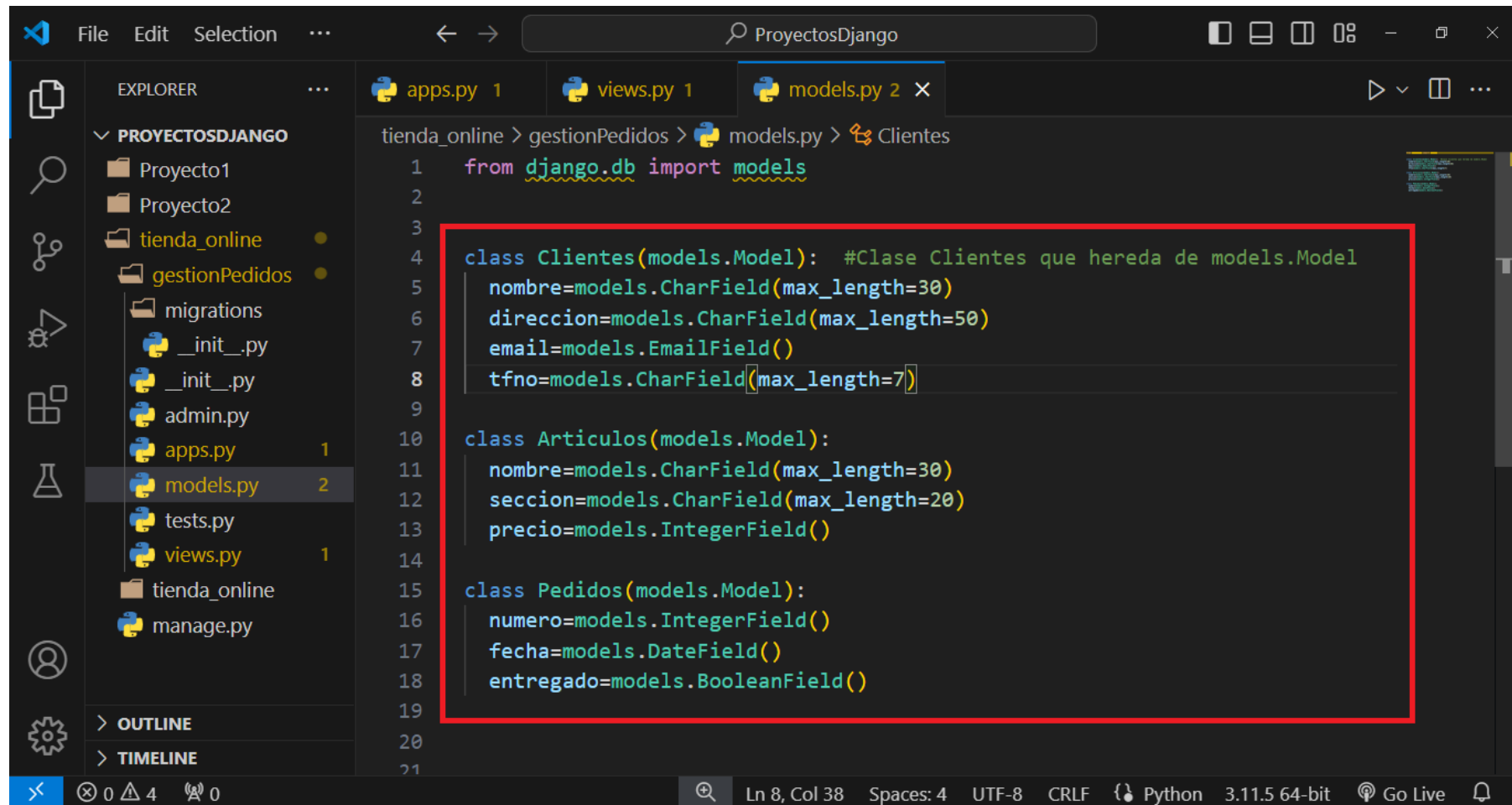
**EmailField():** Un campo para almacenar direcciones de correo electrónico.

**FileField():** Un campo para subir archivos.

**ImageField():** Similar a FileField, pero específicamente diseñado para manejar imágenes.

# 1. Creación de modelos

Estando en el archivo **models.py** procedemos a crear las clases que representan las tablas:



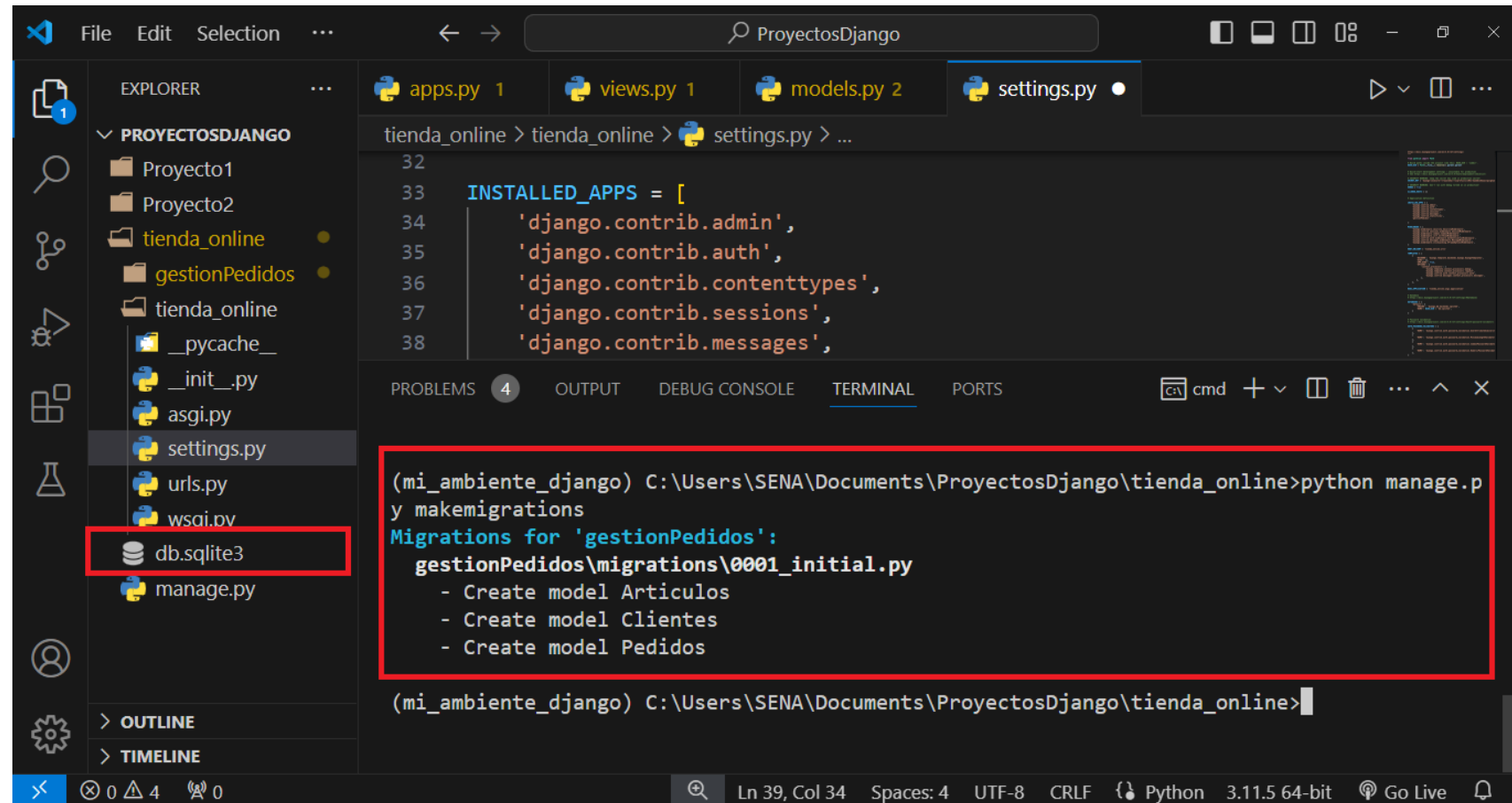
```

1  from django.db import models
2
3
4  class Clientes(models.Model): #Clase Clientes que hereda de models.Model
5      nombre=models.CharField(max_length=30)
6      direccion=models.CharField(max_length=50)
7      email=models.EmailField()
8      tfno=models.CharField(max_length=7)
9
10 class Articulos(models.Model):
11     nombre=models.CharField(max_length=30)
12     seccion=models.CharField(max_length=20)
13     precio=models.IntegerField()
14
15 class Pedidos(models.Model):
16     numero=models.IntegerField()
17     fecha=models.DateField()
18     entregado=models.BooleanField()
19
20
21

```

## 2. Generar migraciones:

Se utiliza el comando **python manage.py makemigrations** para generar archivos de migración basados en los modelos definidos. Es decir, se crea la base de datos.



The screenshot shows a Visual Studio Code editor with a Django project named 'ProyectosDjango'. The Explorer sidebar on the left shows the project structure, including a folder named 'tienda\_online' which contains a subfolder 'gestionPedidos'. The 'db.sqlite3' file is highlighted in the Explorer. The main editor window shows the 'settings.py' file with the 'INSTALLED\_APPS' list. The Terminal panel at the bottom shows the command `python manage.py makemigrations` being executed, resulting in the creation of a migration file 'gestionPedidos\migrations\0001\_initial.py'. The migration details are listed below the command output.

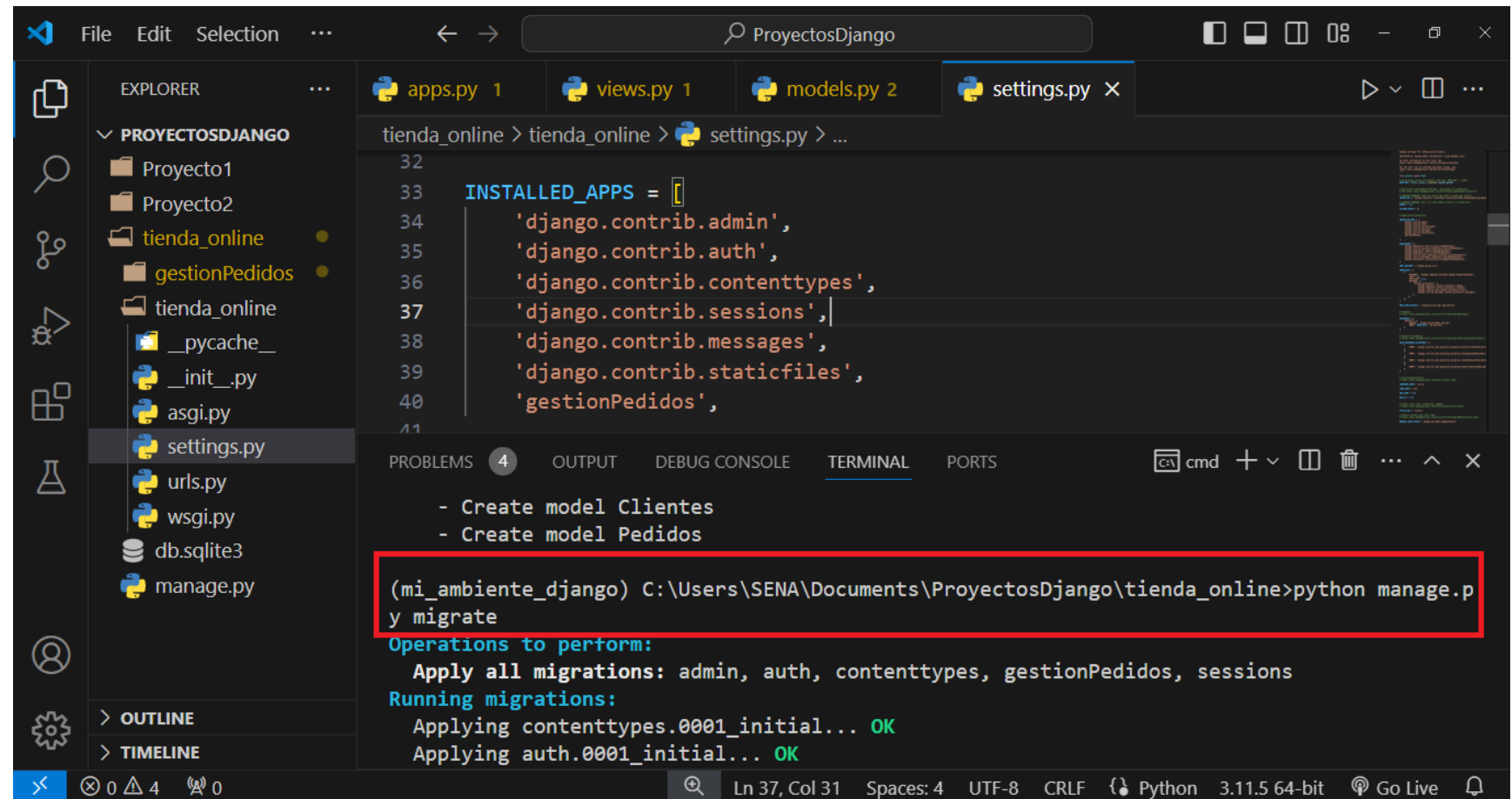
```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.py makemigrations
Migrations for 'gestionPedidos':
  gestionPedidos\migrations\0001_initial.py
    - Create model Articulos
    - Create model Clientes
    - Create model Pedidos

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>
```



### 3. Aplicar las migraciones:

Se emplea el comando **python manage.py migrate** para aplicar las migraciones y crear las tablas en la base de datos SQLite3.



The screenshot shows the Visual Studio Code interface with a Django project named 'ProyectosDjango'. The Explorer panel on the left shows the project structure, including a 'tienda\_online' app. The main editor displays the 'settings.py' file for the 'tienda\_online' app, where the 'INSTALLED\_APPS' list includes 'django.contrib.admin', 'django.contrib.auth', 'django.contrib.contenttypes', 'django.contrib.sessions', 'django.contrib.messages', 'django.contrib.staticfiles', and 'gestionPedidos'. The TERMINAL panel at the bottom shows the command `(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.py migrate` being executed. The output indicates that all migrations will be applied: 'admin', 'auth', 'contenttypes', 'gestionPedidos', and 'sessions'. The terminal also shows the progress of applying migrations, with 'contenttypes.0001\_initial...' and 'auth.0001\_initial...' both marked as 'OK'.

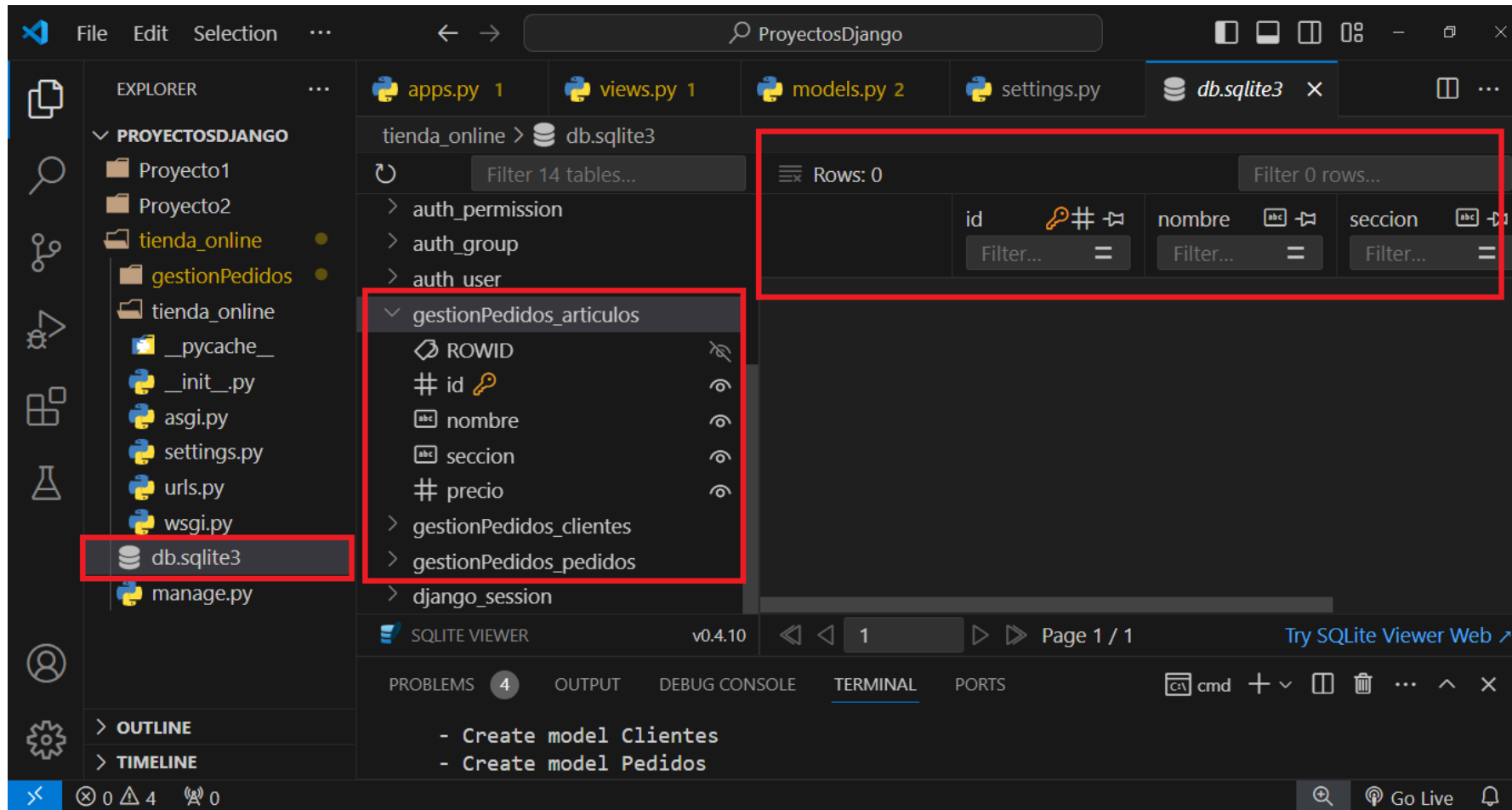
```
tienda_online > tienda_online > settings.py > ...
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'gestionPedidos',
41 ]

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS
- Create model Clientes
- Create model Pedidos

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, gestionPedidos, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
```

# 3. Aplicar las migraciones:

Se pueden revisar la base de datos y las tablas que hemos creado, se observa que Django agrega automáticamente un campo **id** para la llave primaria :



The screenshot displays the Visual Studio Code interface with the Django database interface open. The Explorer panel on the left shows the project structure, with the `db.sqlite3` database file highlighted. The central pane shows a list of tables, with the `gestionPedidos_articulos` table selected. The right pane shows the table schema for `gestionPedidos_articulos`, highlighting the `id` field as the primary key. The bottom terminal shows the command `python manage.py migrate` being executed.

Table	Field	Type	Primary Key
gestionPedidos_articulos	id	INTEGER	Yes
	nombre	TEXT	No
	seccion	TEXT	No
	precio	TEXT	No
gestionPedidos_clientes			
gestionPedidos_pedidos			
django_session			

# Shell de Django

Es una interfaz de línea de comandos que permite interactuar directamente con un proyecto Django utilizando código Python. Se puede acceder a este shell ejecutando el comando **python manage.py shell** desde la raíz del proyecto Django.

Una vez dentro del shell interactivo de Django, se tiene acceso a todo el entorno del proyecto, incluyendo modelos, vistas, funciones y cualquier otra configuración definida. Esto permite explorar y probar rápidamente el código, realizar consultas a la base de datos, crear o modificar objetos de la aplicación y realizar otras tareas de desarrollo de manera interactiva.

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.p
y shell
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> █
```

# CRUD de registros usando el Shell de Django: Create

- Importar el modelo correspondiente que se desea utilizar para insertar el registro. Esto se puede hacer utilizando una declaración **import**, por ejemplo:

```
from <mi_aplicacion.models> import <MiModelo>.
```

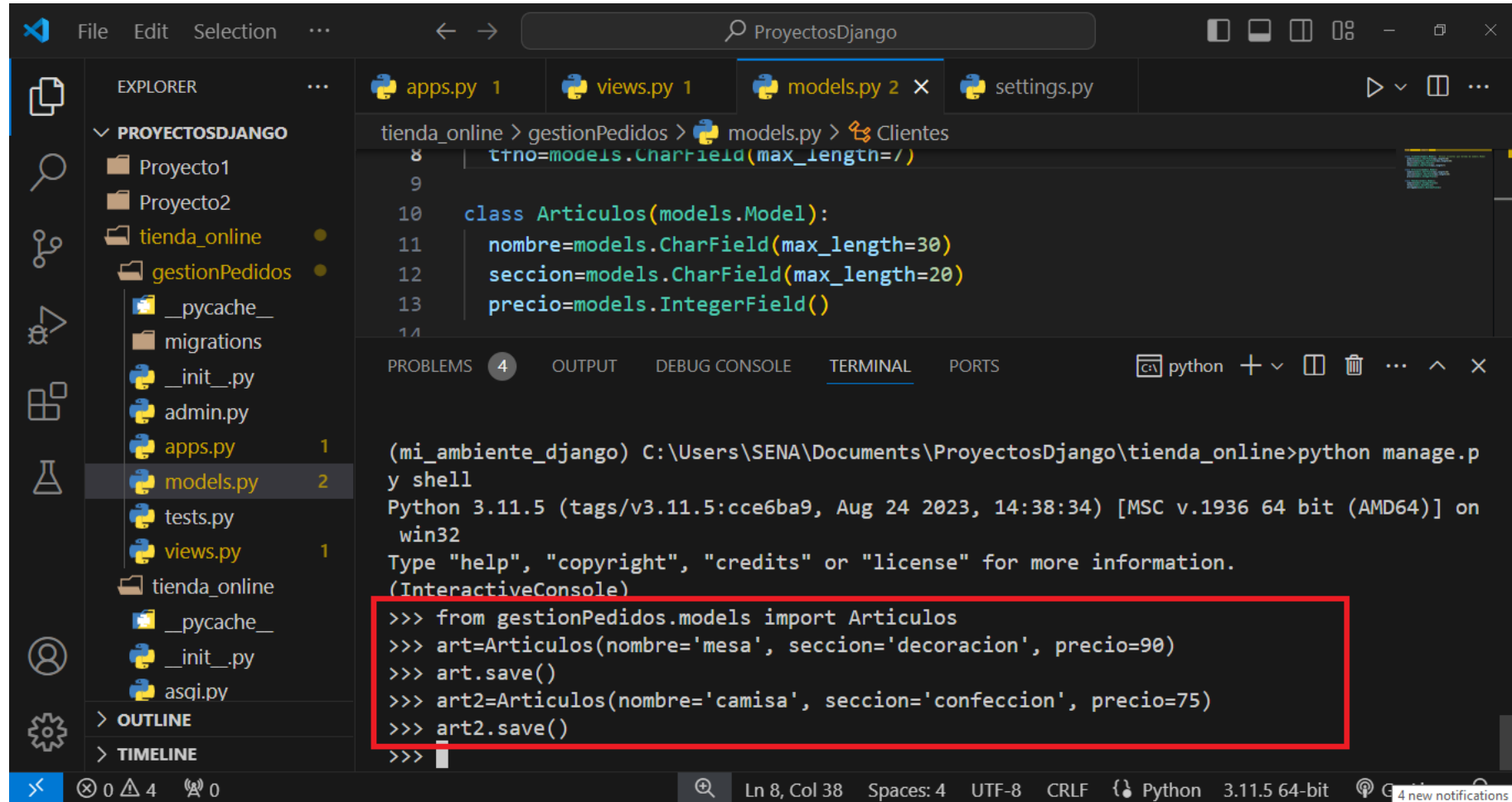
- Crear una instancia del modelo con los valores deseados para cada campo. Esto se hace instanciando la clase del modelo y asignando valores a los atributos correspondientes. Por ejemplo:

```
nuevo_registro = MiModelo(campo1='valor1', campo2='valor2').
```

- Guardar el registro en la base de datos utilizando el método `save()`. Por ejemplo:

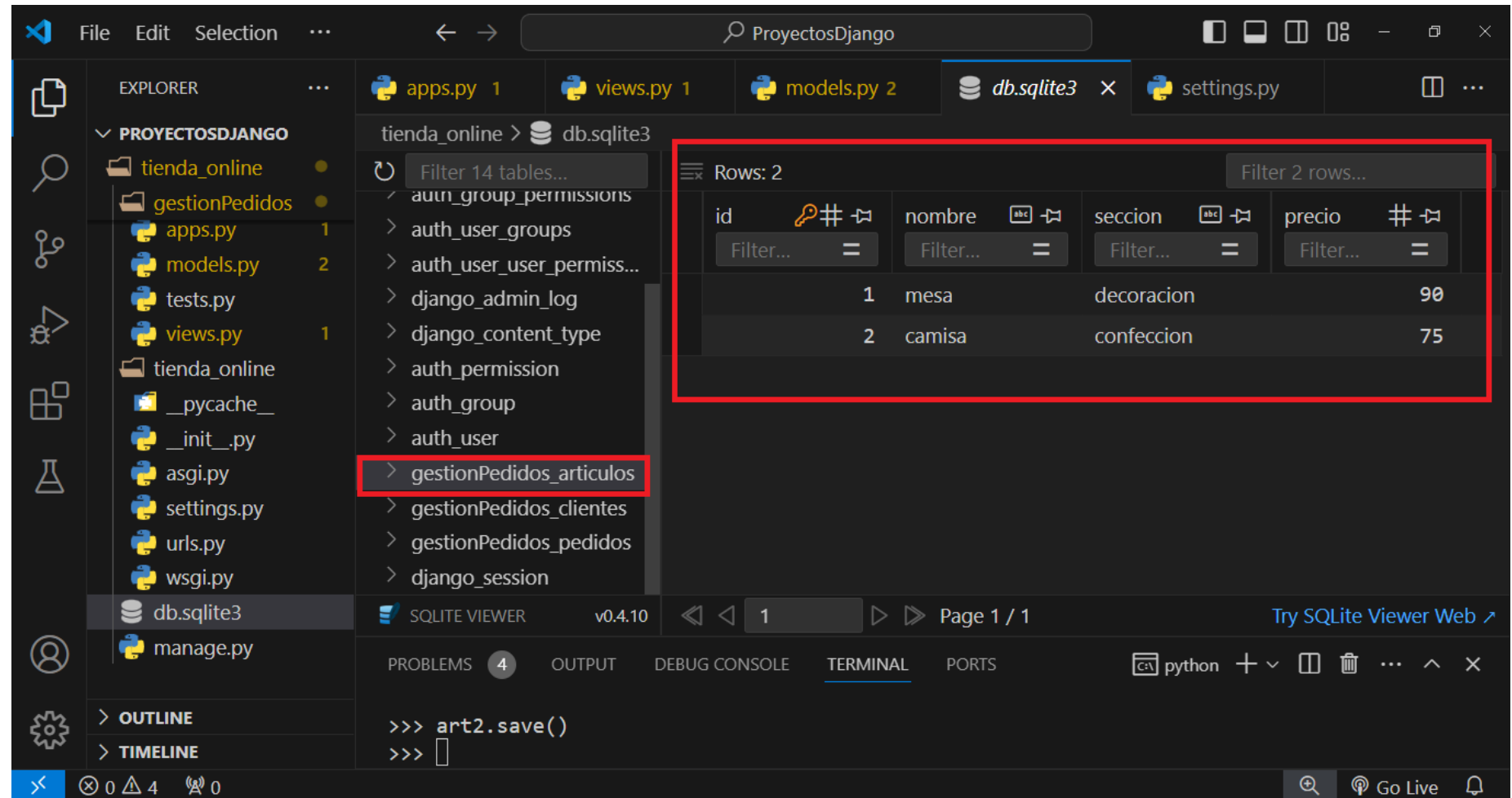
```
nuevo_registro.save()
```

## Insertamos dos registros en el modelo o la tabla Articulos



# CRUD de registros usando el Shell de Django

Verificamos la inserción



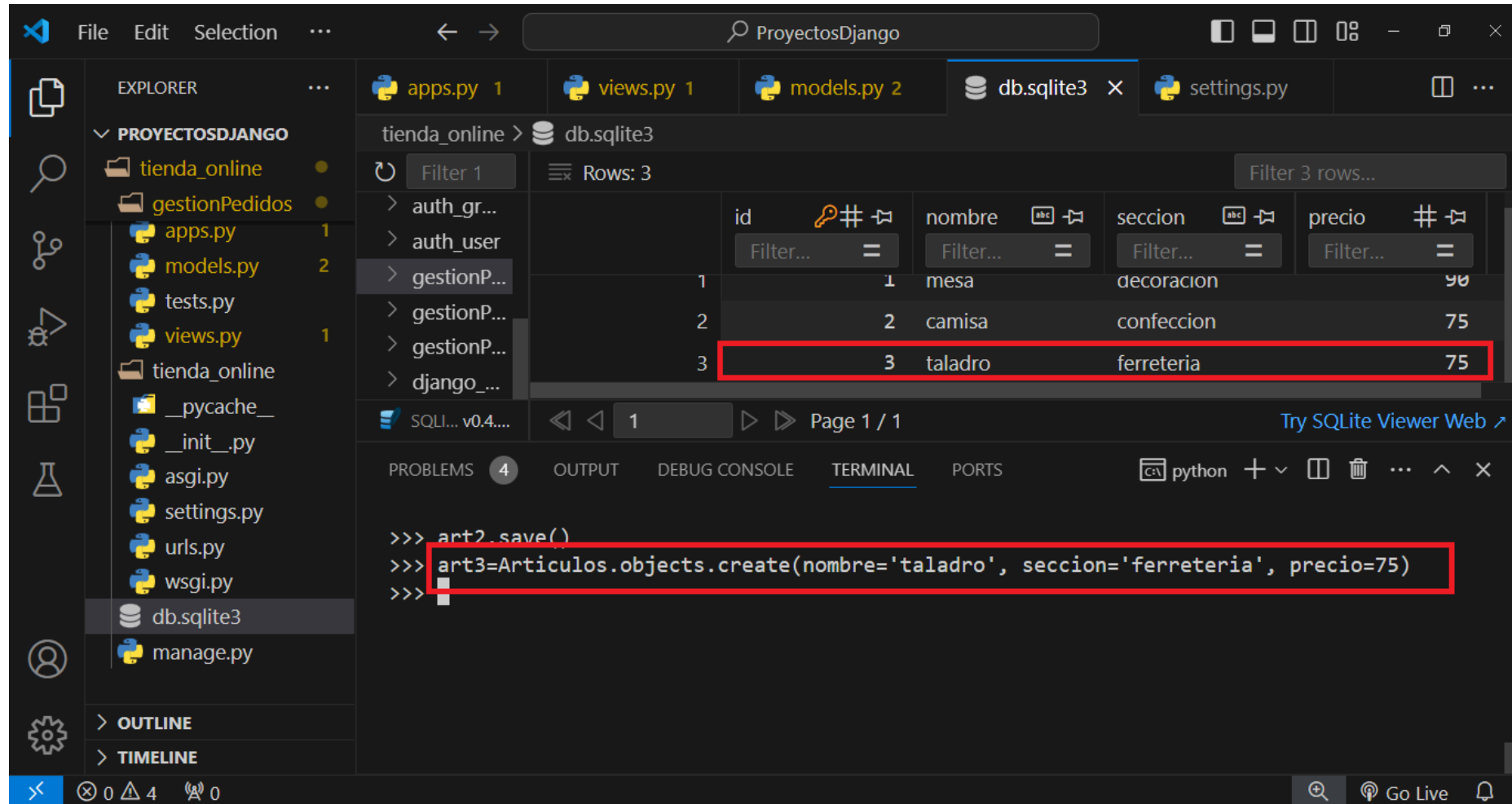
The screenshot shows a Visual Studio Code editor with a Django project named 'ProyectosDjango'. The Explorer panel on the left shows the project structure, including the 'tienda\_online' app. The 'db.sqlite3' file is selected, and the SQLite Viewer extension is open, displaying a table named 'gestionPedidos\_articulos'. The table has four columns: 'id', 'nombre', 'seccion', and 'precio'. Two rows are visible, representing inserted data.

id	nombre	seccion	precio
1	mesa	decoracion	90
2	camisa	confeccion	75

The terminal at the bottom shows the command `>>> art2.save()` being executed, which corresponds to the second row in the database table.

# CRUD de registros usando el Shell de Django

Insertamos otro artículo en la tabla Artículos usando un solo paso con el método `Artículos.objects.create()`



The screenshot shows the Django project structure in VS Code. The `db.sqlite3` database is open, displaying a table with 3 rows. The third row is highlighted, showing an article with `id=3`, `nombre='taladro'`, `seccion='ferreteria'`, and `precio=75`.

The terminal shows the following commands being executed in the Django shell:

```
>>> art2.save()
>>> art3=Articulos.objects.create(nombre='taladro', seccion='ferreteria', precio=75)
>>>
```

# CRUD de registros usando el Shell de Django: Update

- Importar el modelo correspondiente que se desea utilizar para actualizar el registro. Esto se hace mediante una declaración de importación, por ejemplo:

**`from mi_aplicacion.models import MiModelo.`**

- Obtener el registro que se desea actualizar utilizando consultas o métodos de filtrado proporcionados por Django. Por ejemplo:

**`registro = MiModelo.objects.get(id=1).`**

- Actualizar los campos del registro obtenido según sea necesario. Esto se hace directamente modificando los atributos del objeto obtenido. Por ejemplo:

**`registro.campo1 = 'nuevo_valor'.`**

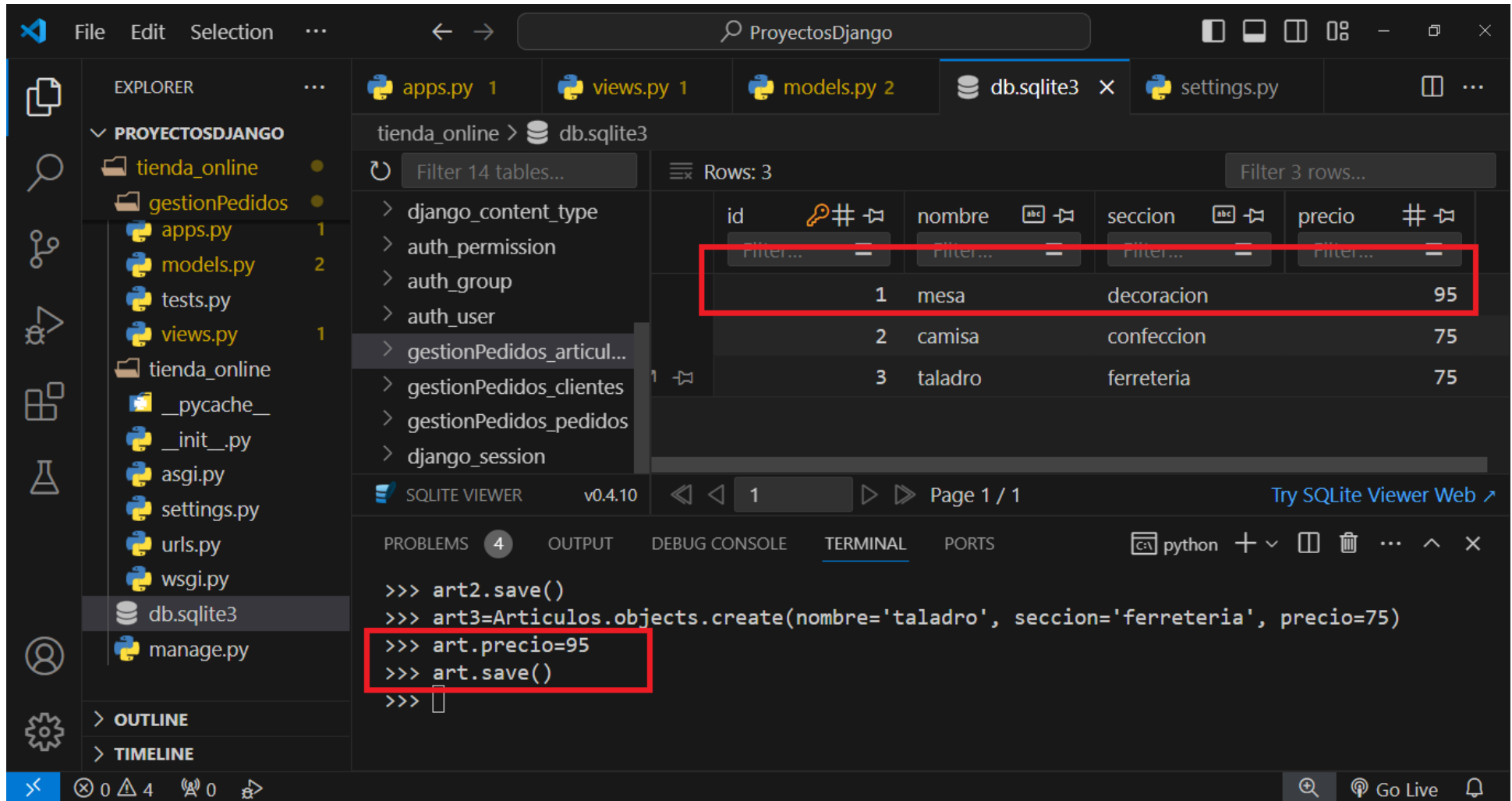
- Guardar los cambios en la base de datos utilizando el método `save()` en el objeto del registro. Por ejemplo:

**`registro.save().`**



# CRUD de registros usando el Shell de Django

Actualizar el primer artículo insertado en la tabla **Articulos**



The screenshot shows a Django project named 'ProyectosDjango' in VS Code. The Explorer panel on the left shows the project structure with folders like 'tienda\_online' and 'gestionPedidos', and files like 'apps.py', 'models.py', 'views.py', 'settings.py', 'urls.py', and 'wsgi.py'. The SQLite viewer panel in the center shows the 'tienda\_online' database with a table named 'Articulos'. The table has columns: id, nombre, seccion, and precio. The data is as follows:

id	nombre	seccion	precio
1	mesa	decoracion	95
2	camisa	confeccion	75
3	taladro	ferreteria	75

The terminal panel at the bottom shows the following commands being executed:

```
>>> art2.save()
>>> art3=Articulos.objects.create(nombre='taladro', seccion='ferreteria', precio=75)
>>> art.precio=95
>>> art.save()
>>>
```

# CRUD de registros usando el Shell de Django: Delete

- Importar el modelo correspondiente que se desea utilizar para eliminar el registro. Esto se realiza mediante una declaración de importación, por ejemplo:

```
from mi_aplicacion.models import MiModelo.
```

- Obtener el registro que se desea eliminar utilizando consultas o métodos de filtrado proporcionados por Django. Por ejemplo:

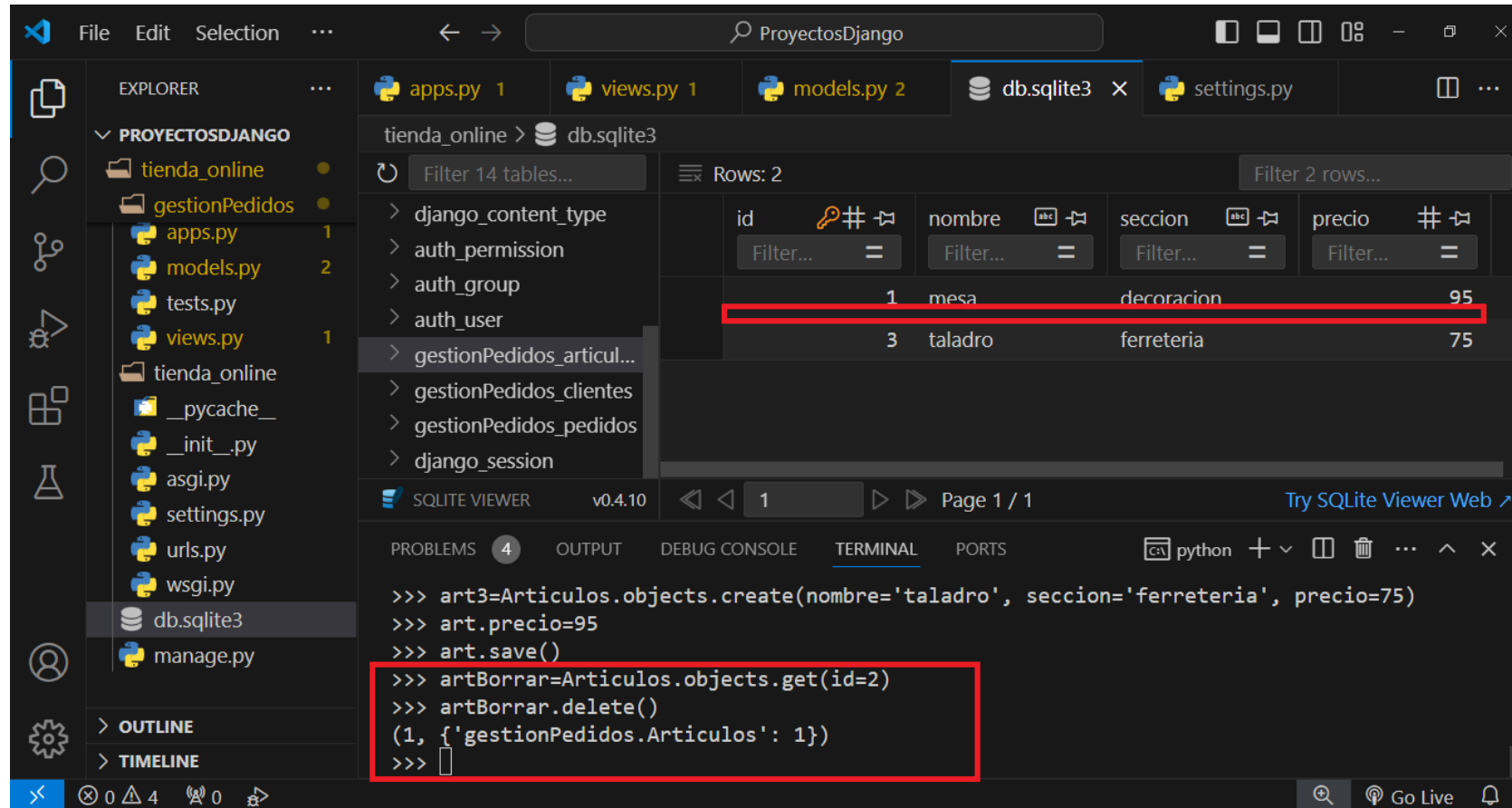
```
registro = MiModelo.objects.get(id=1).
```

- Eliminar el registro utilizando el método delete() en el objeto del registro obtenido. Por ejemplo:

```
registro.delete().
```

# CRUD de registros usando el Shell de Django

Borrar el segundo artículo insertado en la tabla **Articulos**



tienda\_online > db.sqlite3

Filter 14 tables... Rows: 2 Filter 2 rows...

id	nombre	seccion	precio
1	mesa	decoración	95
3	taladro	ferreteria	75

SQLITE VIEWER v0.4.10 Page 1 / 1 Try SQLite Viewer Web

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS python + -

```
>>> art3=Articulos.objects.create(nombre='taladro', seccion='ferreteria', precio=75)
>>> art.precio=95
>>> art.save()
>>> artBorrar=Articulos.objects.get(id=2)
>>> artBorrar.delete()
(1, {'gestionPedidos.Articulos': 1})
>>>
```

# CRUD de registros usando el Shell de Django: Read

- Para consultar registros utilizando el shell de Django, se pueden seguir estos pasos :
- Importar el modelo correspondiente que contiene los registros que se desean consultar. Esto se realiza mediante una declaración de importación, por ejemplo:

```
from mi_aplicacion.models import MiModelo.
```

- Realizar consultas utilizando métodos proporcionados por Django, como `all()`, `filter()`, `get()`, etc. Por ejemplo:

```
registros = MiModelo.objects.all()
```

para obtener todos los registros de la tabla.

- Iterar sobre los resultados para procesar o mostrar la información según sea necesario. Por ejemplo:

## CRUD de registros usando el Shell de Django: Read

- Iterar sobre los resultados para procesar o mostrar la información según sea necesario.  
Por ejemplo:

```
for registro in registros:  
    print(registro.campo1)
```

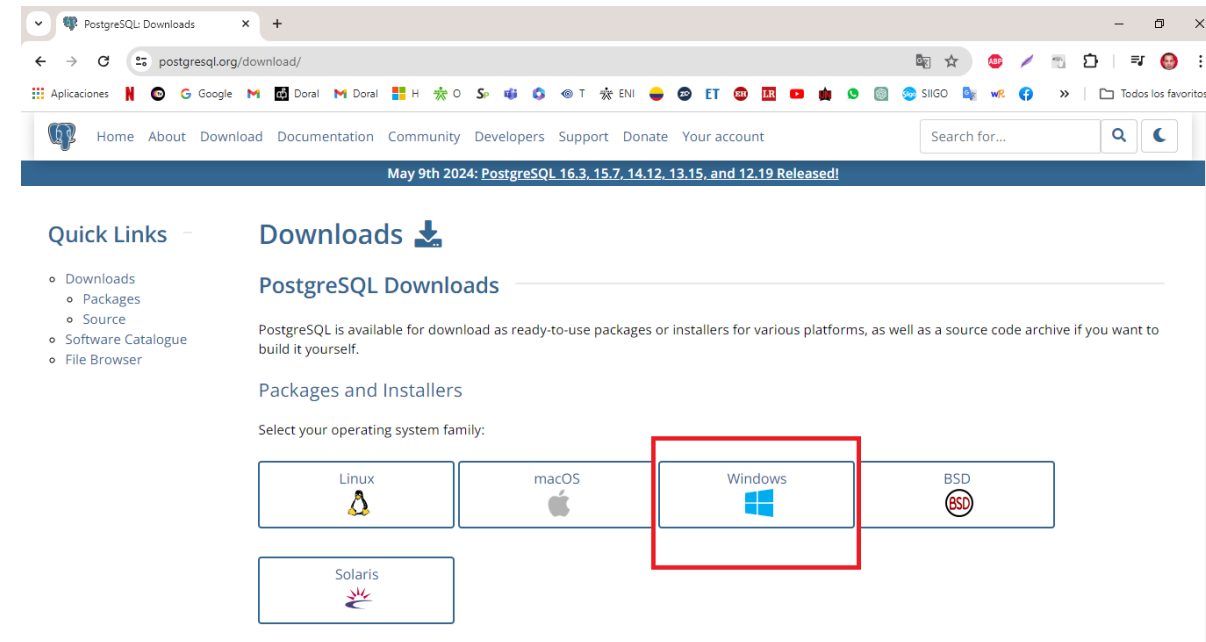
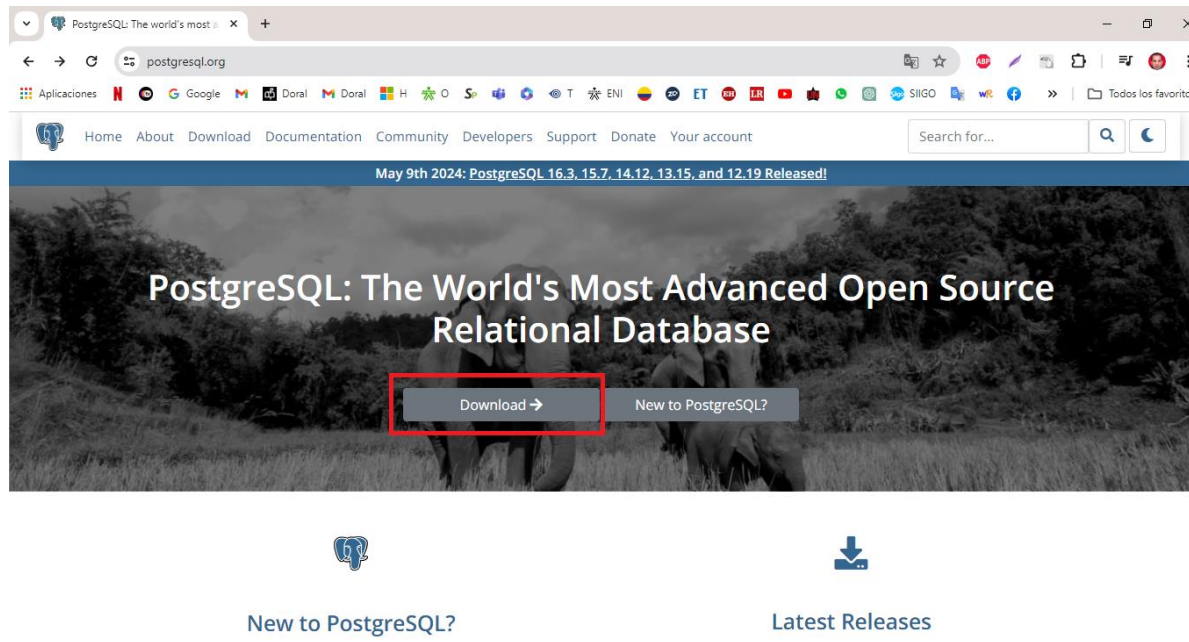
Nota: Para ejecutar el for exterior en el Shell recuerda dar doble enter al finalizar el bucle

```
>>> Lista=Articulos.objects.all()  
>>> for registro in Lista:  
...     print(registro.nombre)  
...  
mesa  
taladro
```

- Para salir del Shell de Django utilizar el comando **exit()**

# Utilizar PostgreSQL con Django: Instalando PostgreSQL

1. Descargar PostgreSQL desde: <https://www.postgresql.org/>



# Utilizar PostgreSQL con Django: Instalando PostgreSQL

PostgreSQL: Windows installers

postgresql.org/download/windows/

Home About Download Documentation Community Developers Support Donate Your account

May 9th 2024: PostgreSQL 16.3, 15.7, 14.12, 13.15, and 12.19 Released!

**Quick Links**

- Downloads
- Packages
- Source
- Software Catalogue
- File Browser

**Windows installers**

**Interactive installer by EDB**

**Download the installer** certified by EDB for all supported PostgreSQL versions.

**Note!** This installer is hosted by EDB and not on the PostgreSQL community servers. If you have issues with the website it's hosted on, please contact [webmaster@enterprisedb.com](mailto:webmaster@enterprisedb.com).

This installer includes the PostgreSQL server, pgAdmin; a graphical tool for managing and developing your databases, and StackBuilder; a package manager that can be used to download and install additional PostgreSQL tools and drivers. Stackbuilder includes management, integration, migration, replication, geospatial, connectors and other tools.

This installer can run in graphical or silent install modes.

The installer is designed to be a straightforward, fast way to get up and running with PostgreSQL on Windows.

Advanced users can also download a [zip archive](#) of the binaries, without the installer. This download is intended for users who wish to include PostgreSQL as part of another application installer.

**Platform support**

The installers are tested by EDB on the following platforms. They can generally be expected to run on other comparable versions, for example,

EDB: Open-Source, Enterprise PostgreSQL

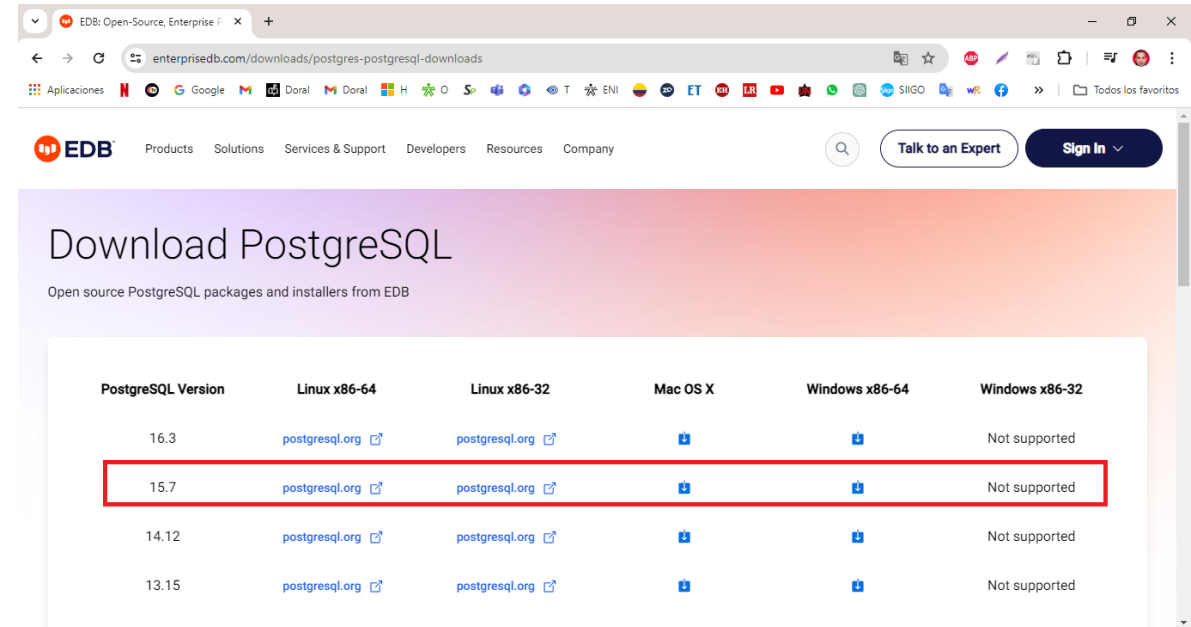







enterprisedb.com/downloads/postgres-postgresql-downloads

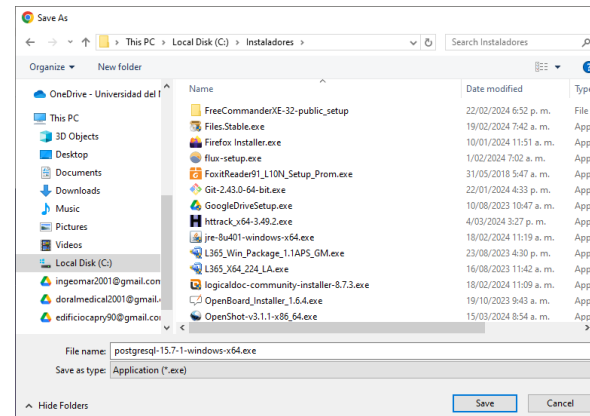
Products Solutions Services & Support Developers Resources Company

Talk to an Expert Sign In

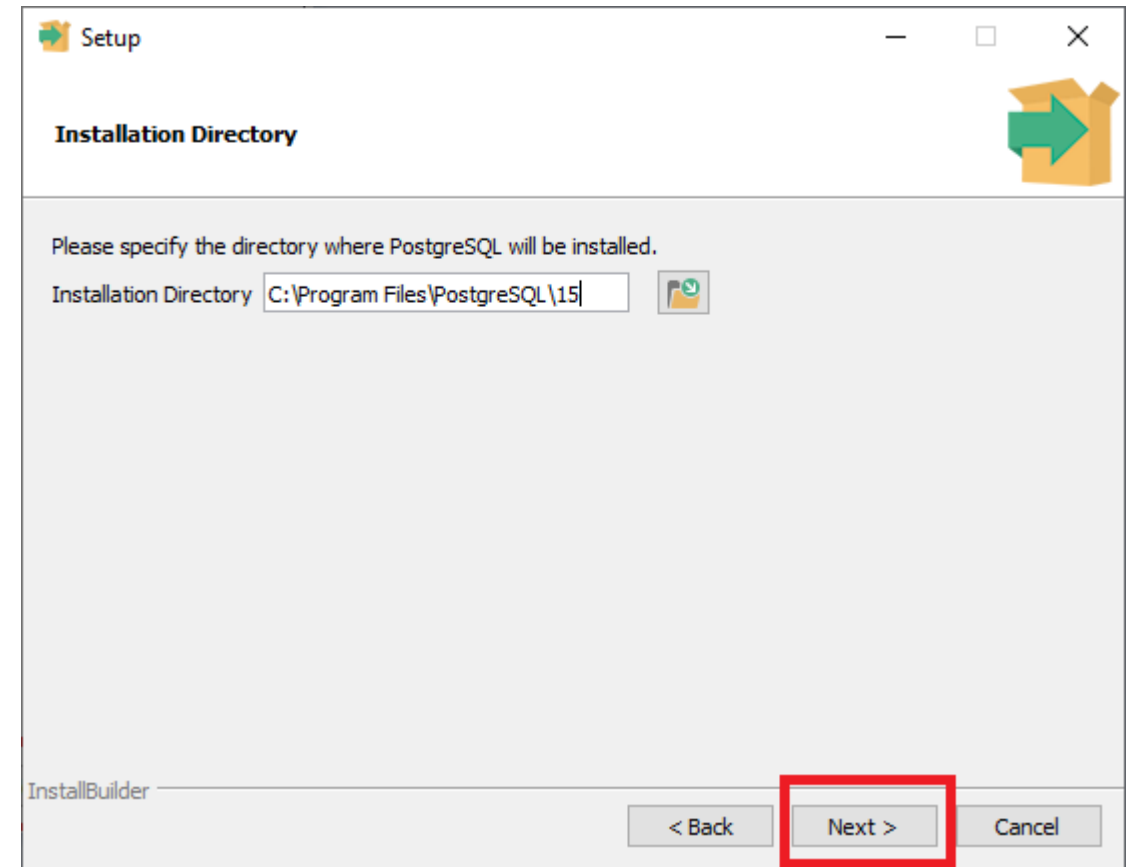
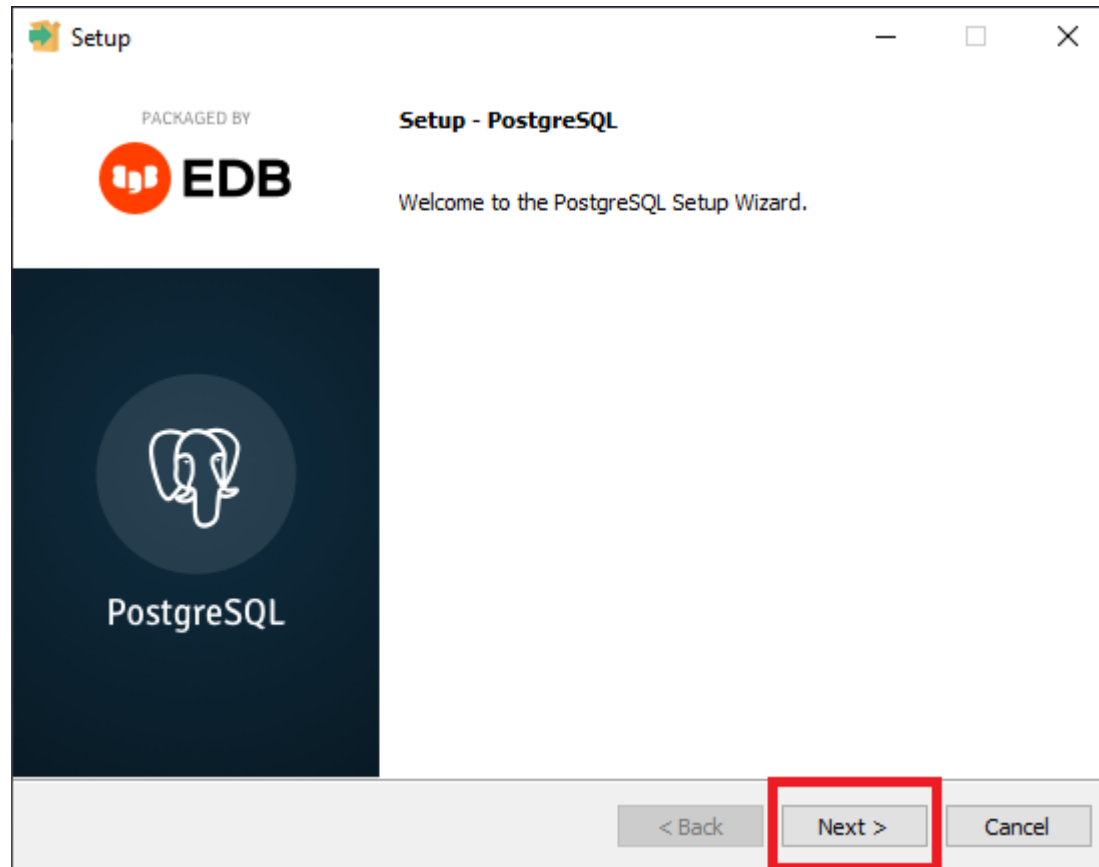
## Download PostgreSQL

Open source PostgreSQL packages and installers from EDB

PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
16.3	<a href="#">postgresql.org</a>	<a href="#">postgresql.org</a>			Not supported
15.7	<a href="#">postgresql.org</a>	<a href="#">postgresql.org</a>			Not supported
14.12	<a href="#">postgresql.org</a>	<a href="#">postgresql.org</a>			Not supported
13.15	<a href="#">postgresql.org</a>	<a href="#">postgresql.org</a>			Not supported

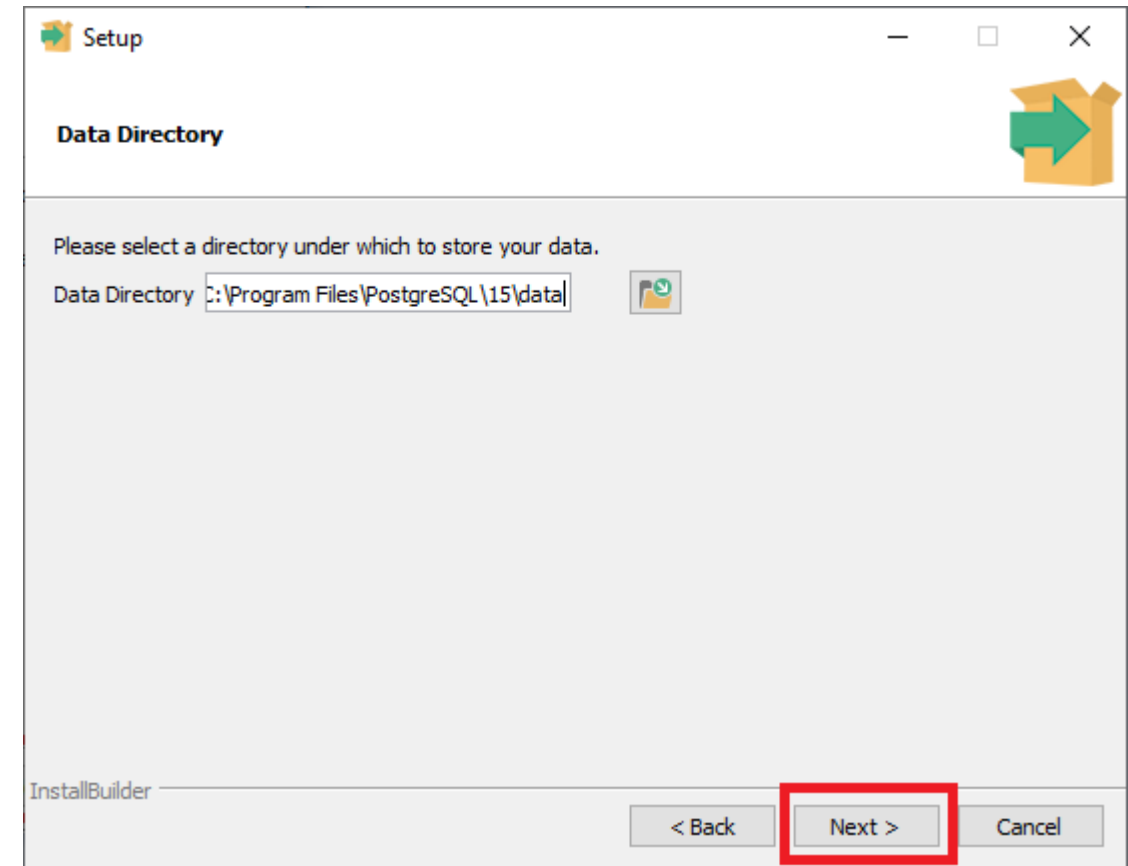
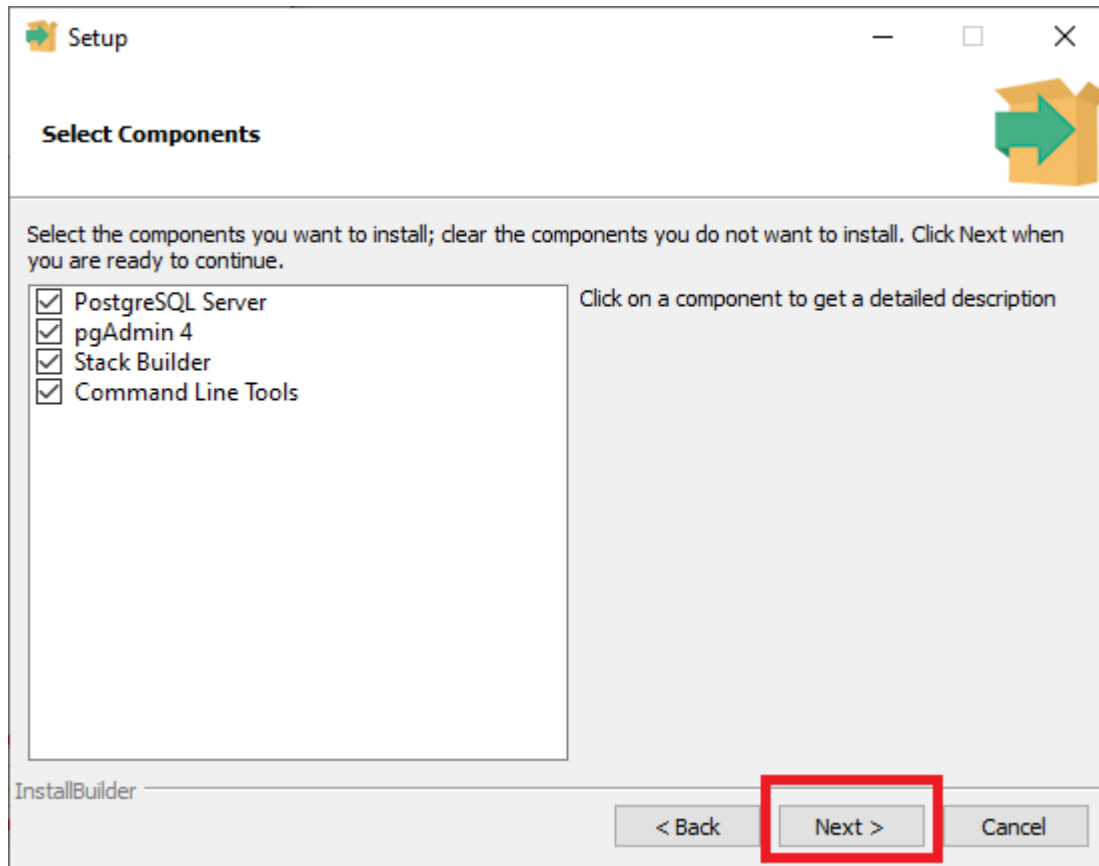


# Utilizar PostgreSQL con Django: Instalando PostgreSQL

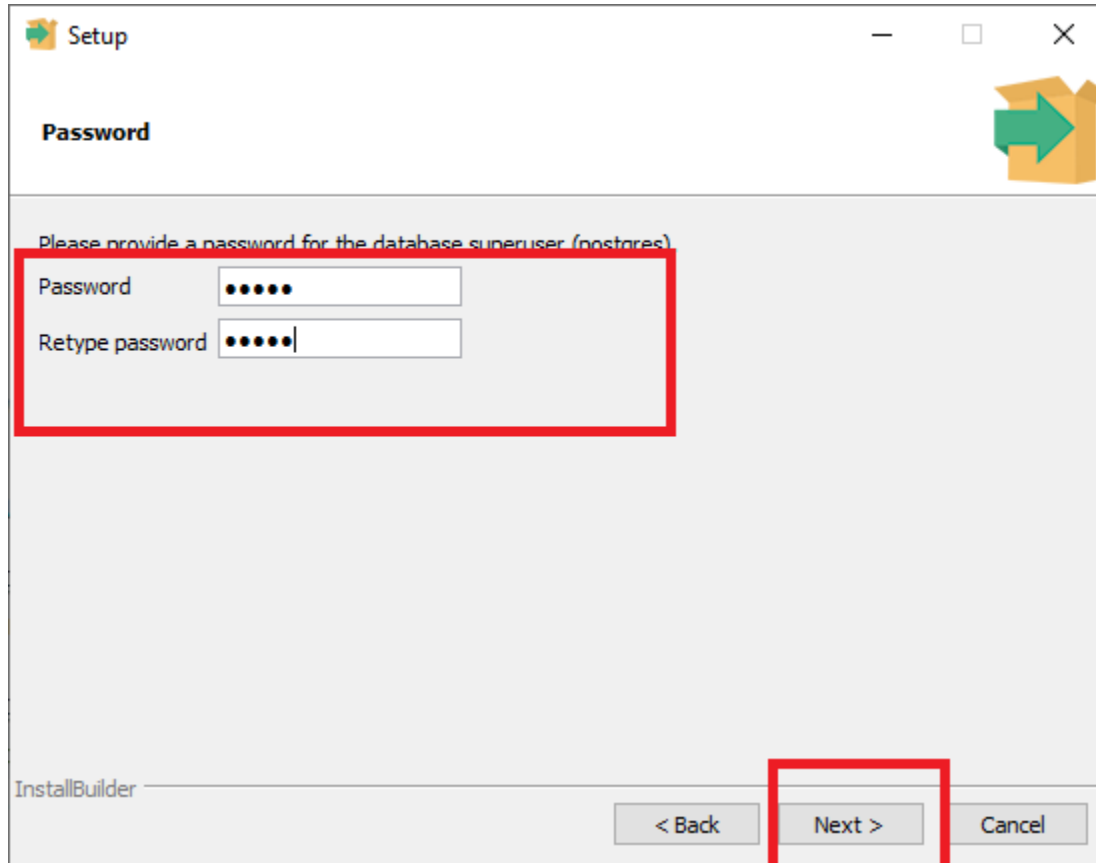




# Utilizar PostgreSQL con Django: Instalando PostgreSQL



# Utilizar PostgreSQL con Django: Instalando PostgreSQL



Setup

**Password**

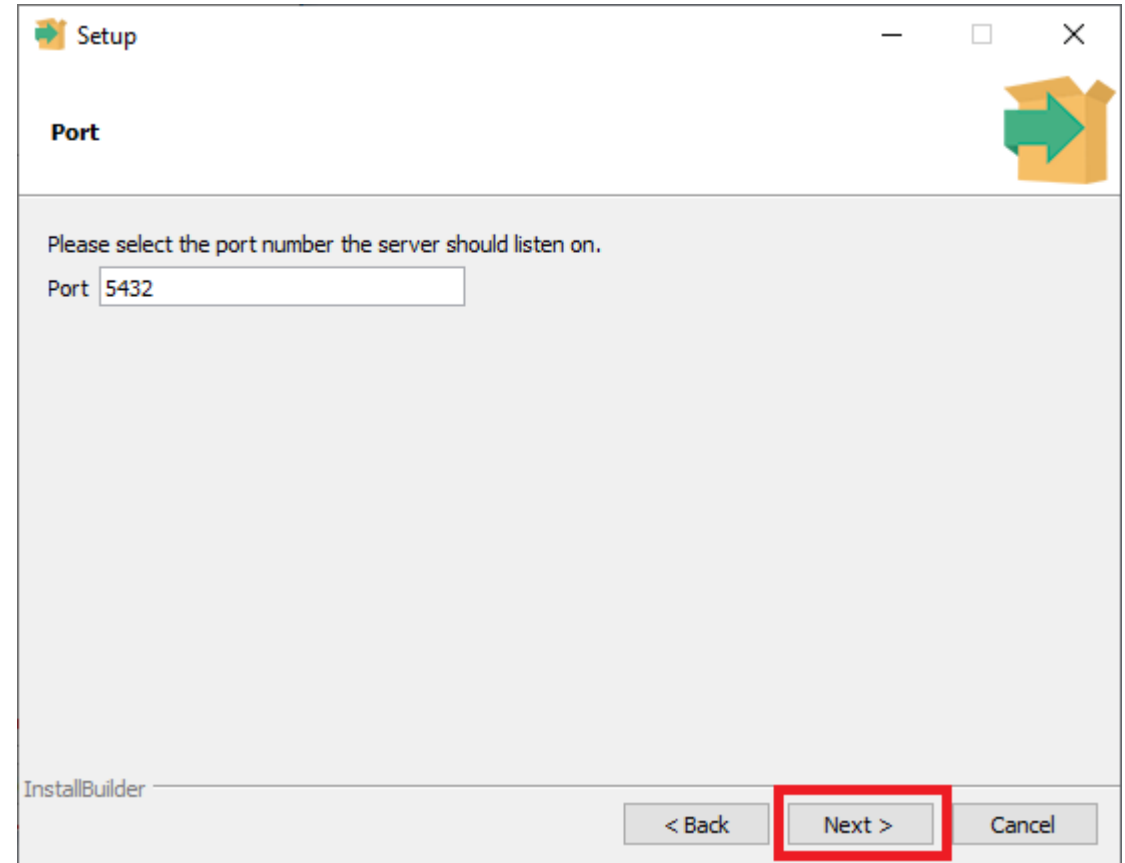
Please provide a password for the database superuser (postgres).

Password

Retype password

InstallBuilder

< Back Next > Cancel



Setup

**Port**

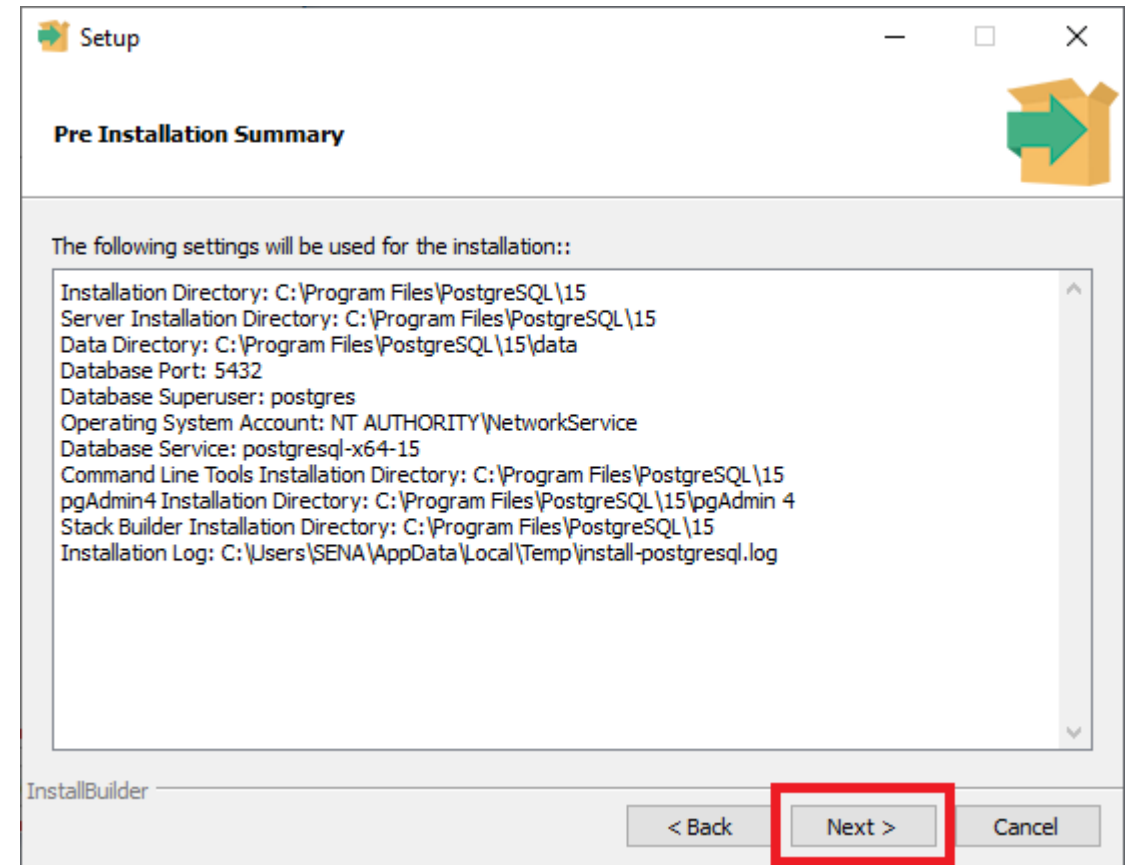
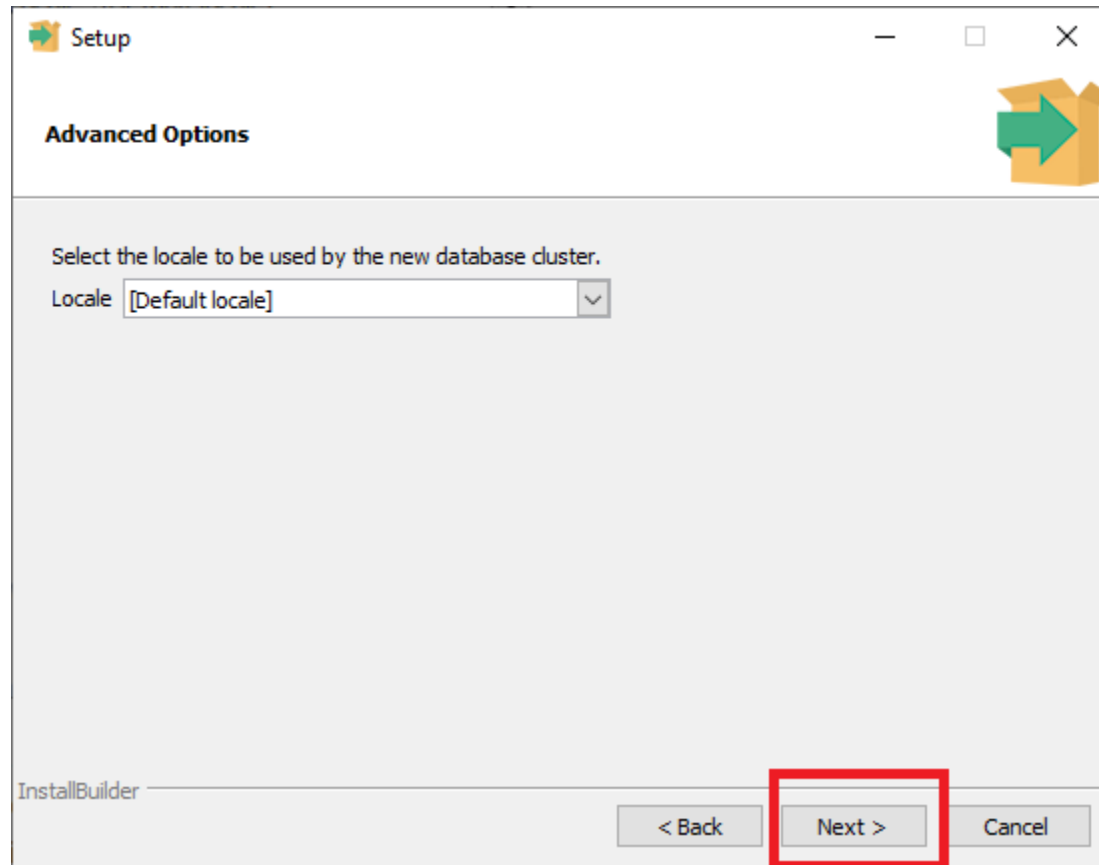
Please select the port number the server should listen on.

Port

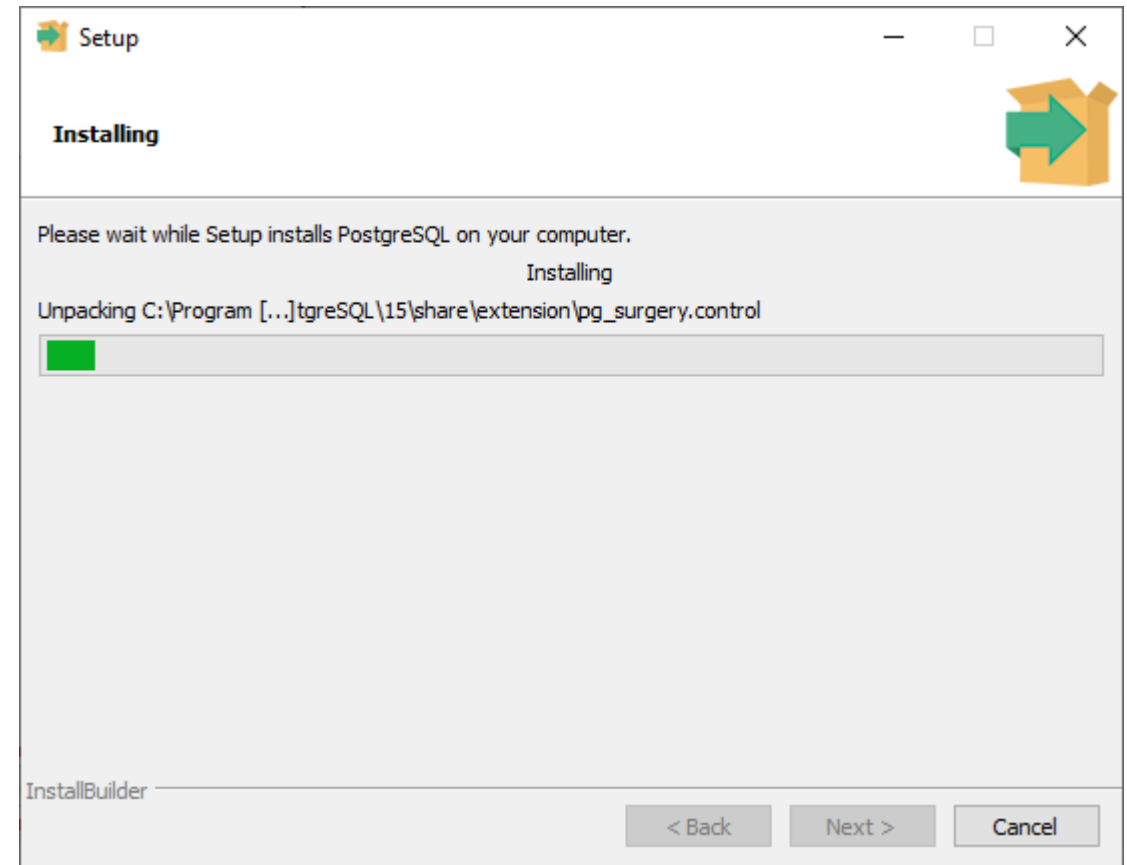
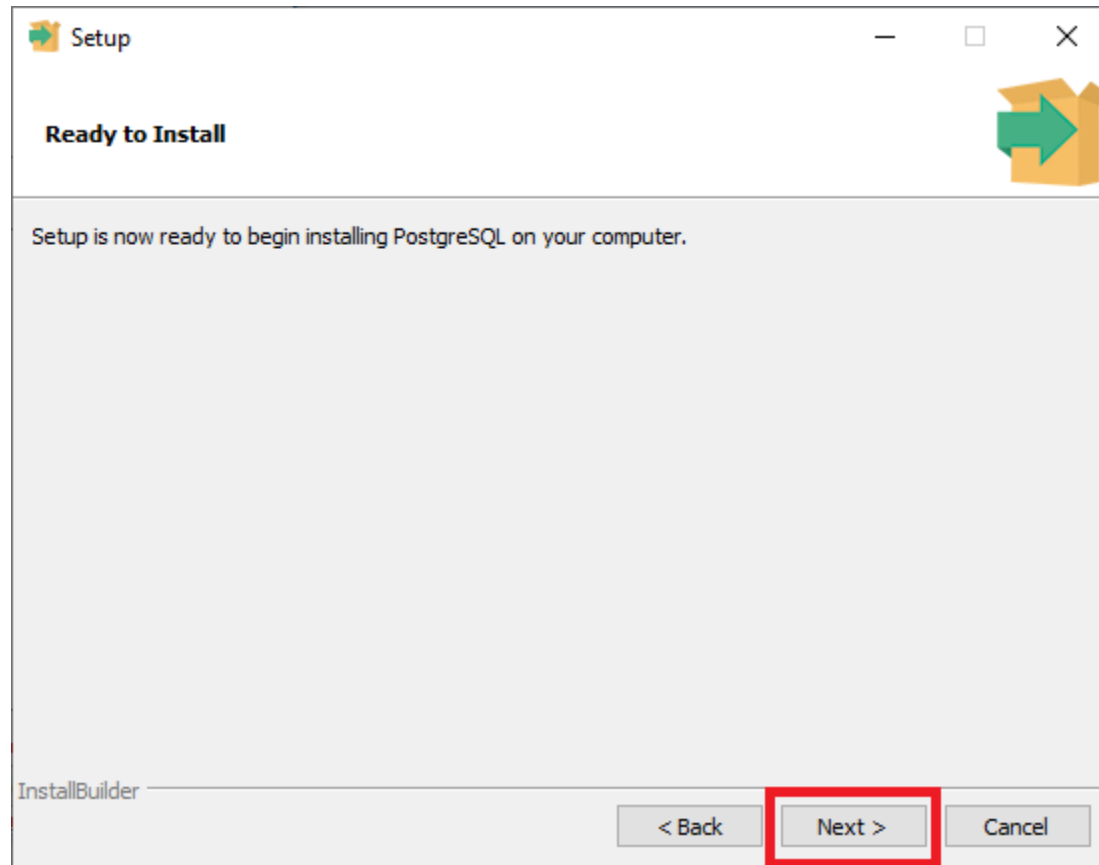
InstallBuilder

< Back Next > Cancel

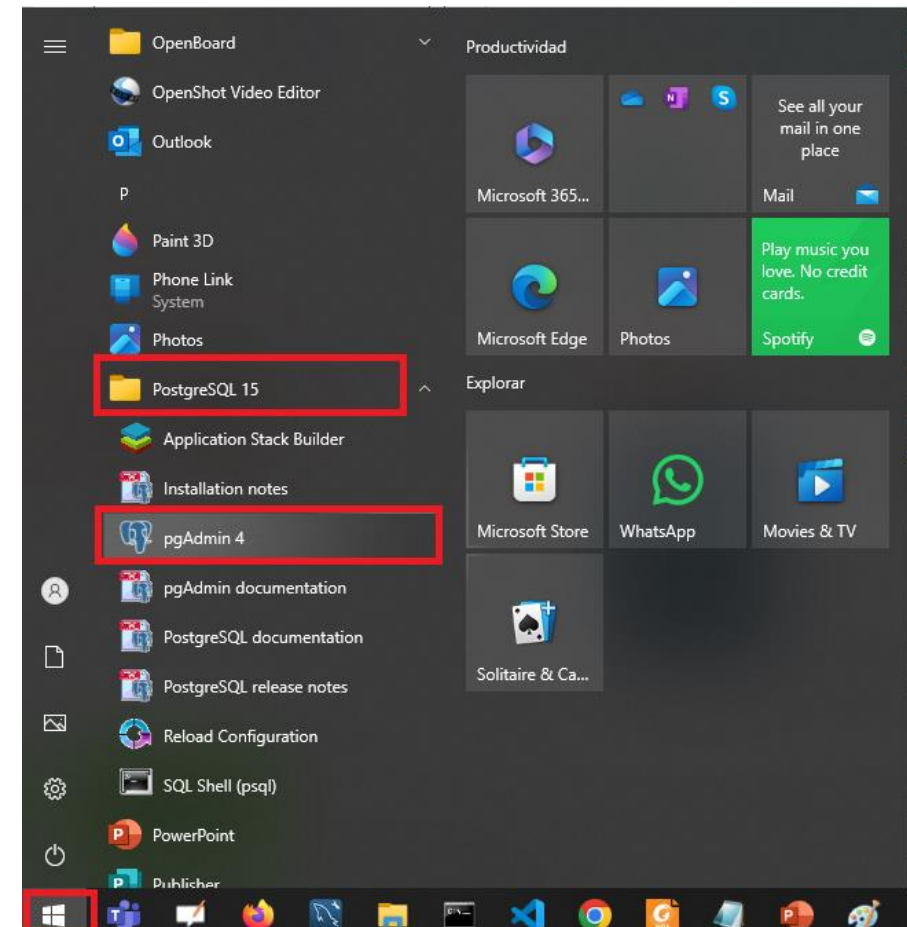
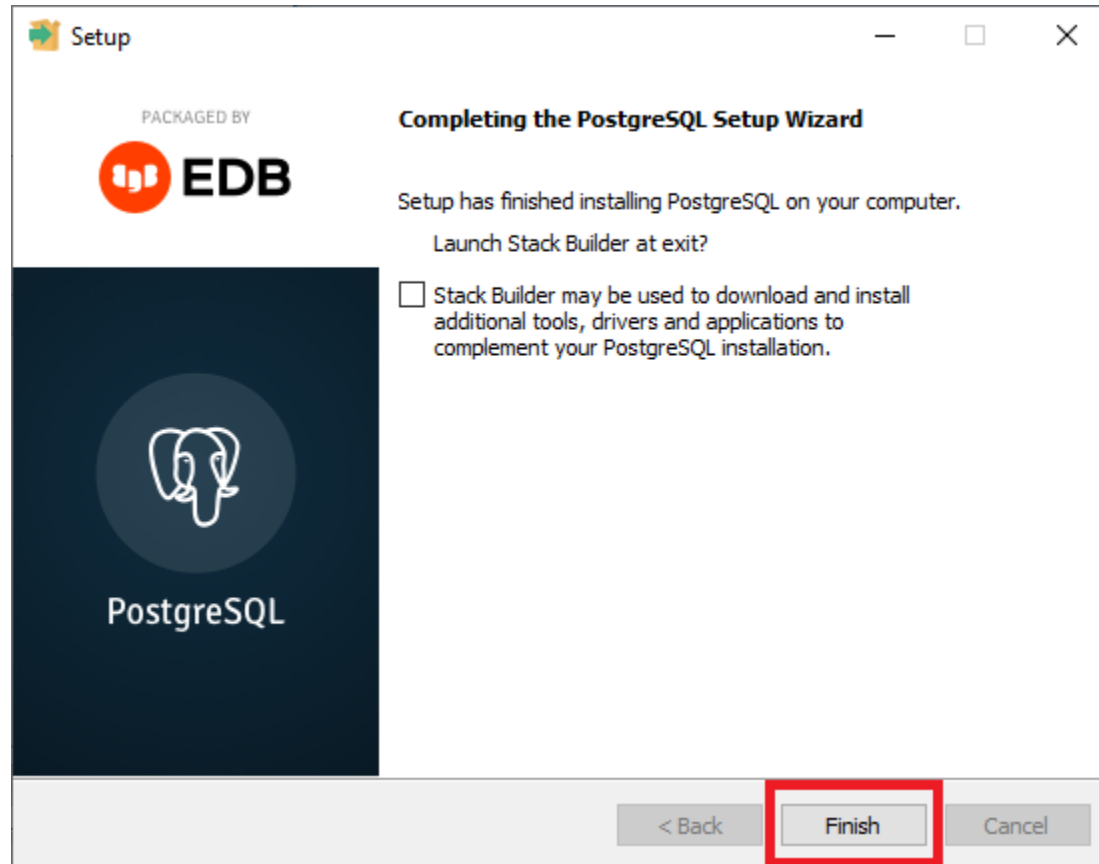
# Utilizar PostgreSQL con Django: Instalando PostgreSQL



# Utilizar PostgreSQL con Django: Instalando PostgreSQL

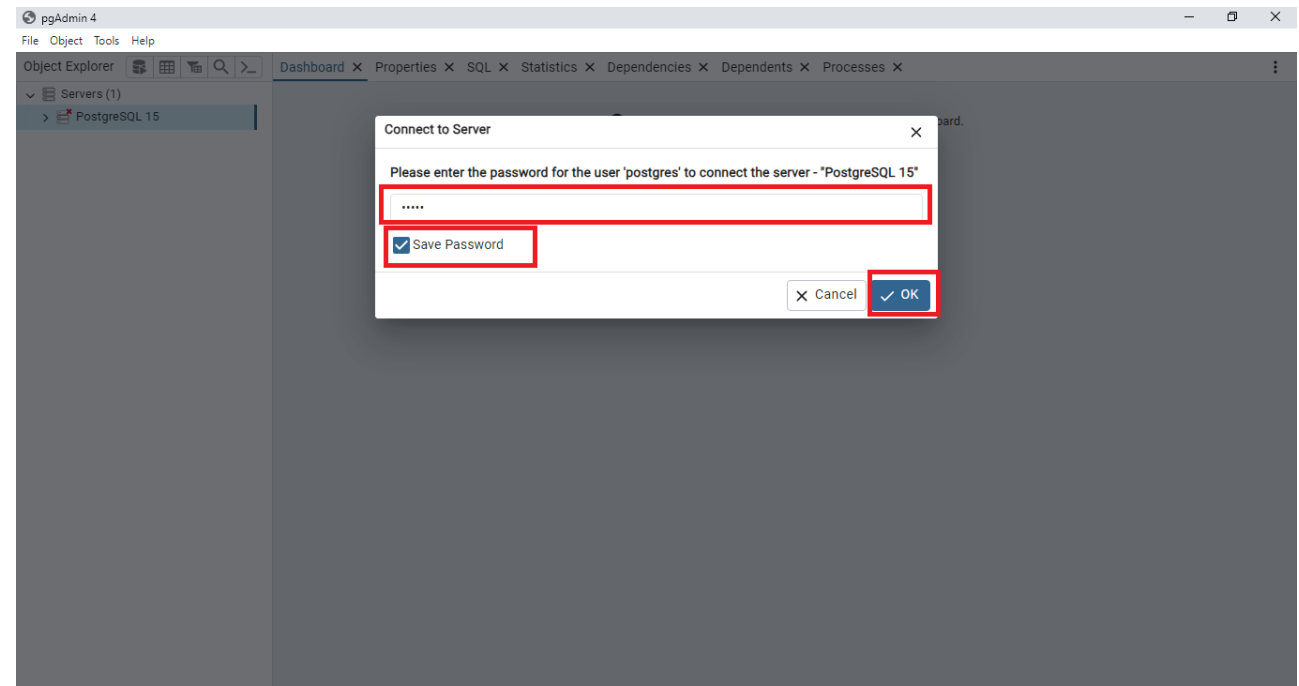
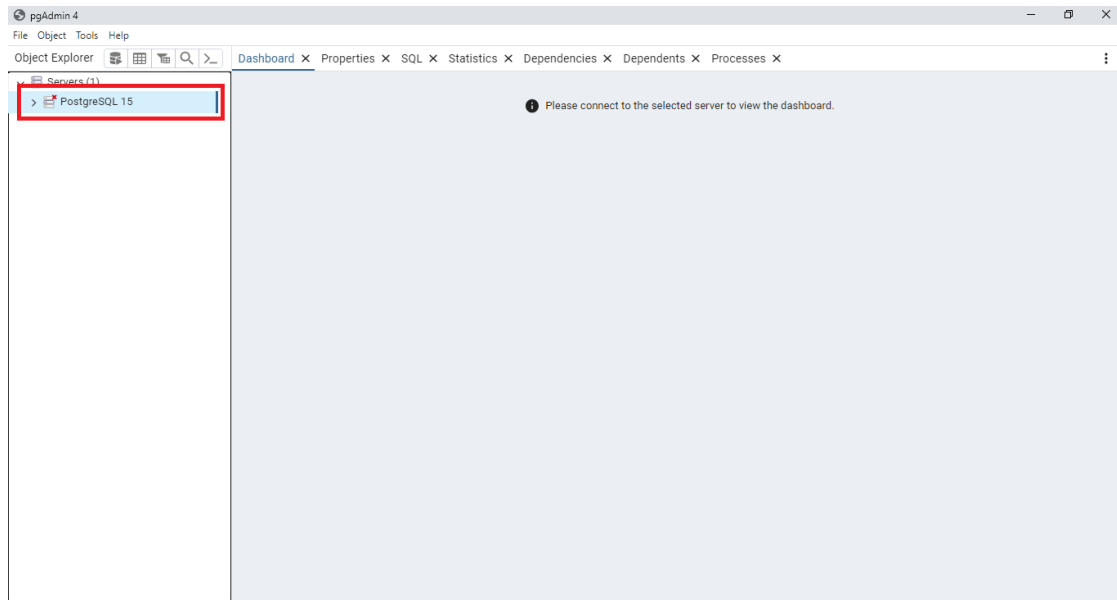


# Utilizar PostgreSQL con Django: Instalando PostgreSQL



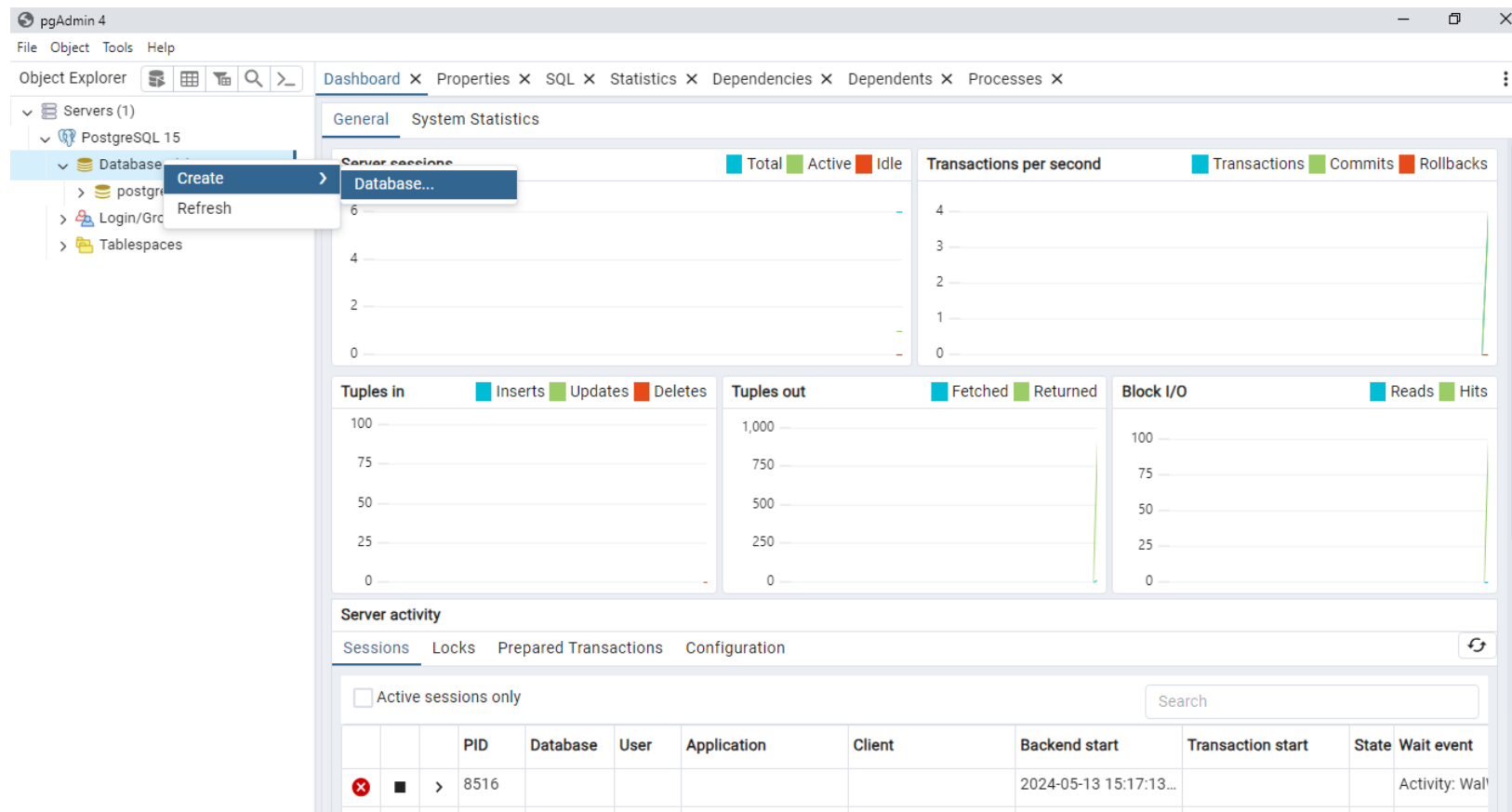
# Utilizar PostgreSQL con Django: Abriendo pgAdmin 4

Doble clic sobre PostgreSQL 15. A continuación damos nuestra contraseña ingresada durante la instalación y clic en Ok



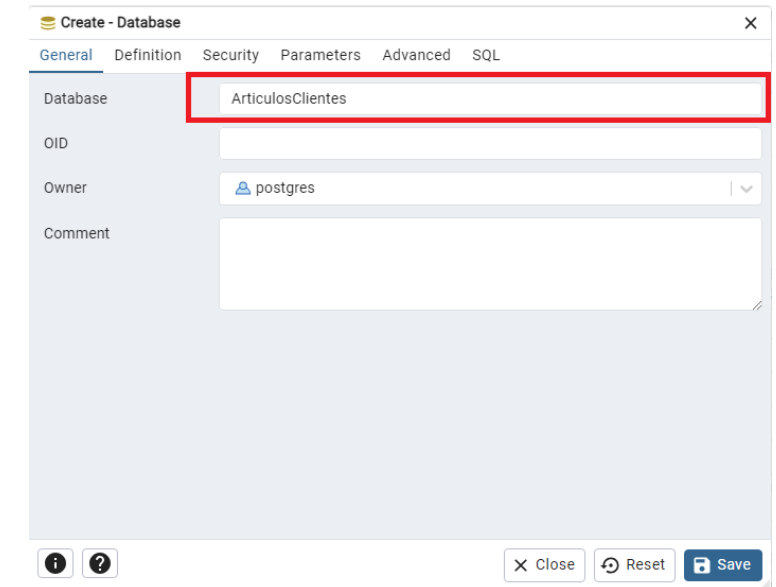
# Utilizar PostgreSQL con Django: Creando Database en pgAdmin 4

Clic contrario sobre Database -> Create -> Database. Le asignamos nombre a la base de datos y damos clic en Save.



The screenshot shows the pgAdmin 4 interface. In the Object Explorer on the left, the 'Database' folder under 'PostgreSQL 15' is selected. A right-click context menu is open, showing 'Create' and 'Refresh' options. The 'Create' option is highlighted, and a sub-menu is visible with 'Database...' selected. The main panel displays the 'General' tab of the 'Server statistics' window, showing various performance metrics like 'Server sessions', 'Transactions per second', 'Tuples in', 'Tuples out', and 'Block I/O'. At the bottom, the 'Server activity' window is visible, showing a table of active sessions.

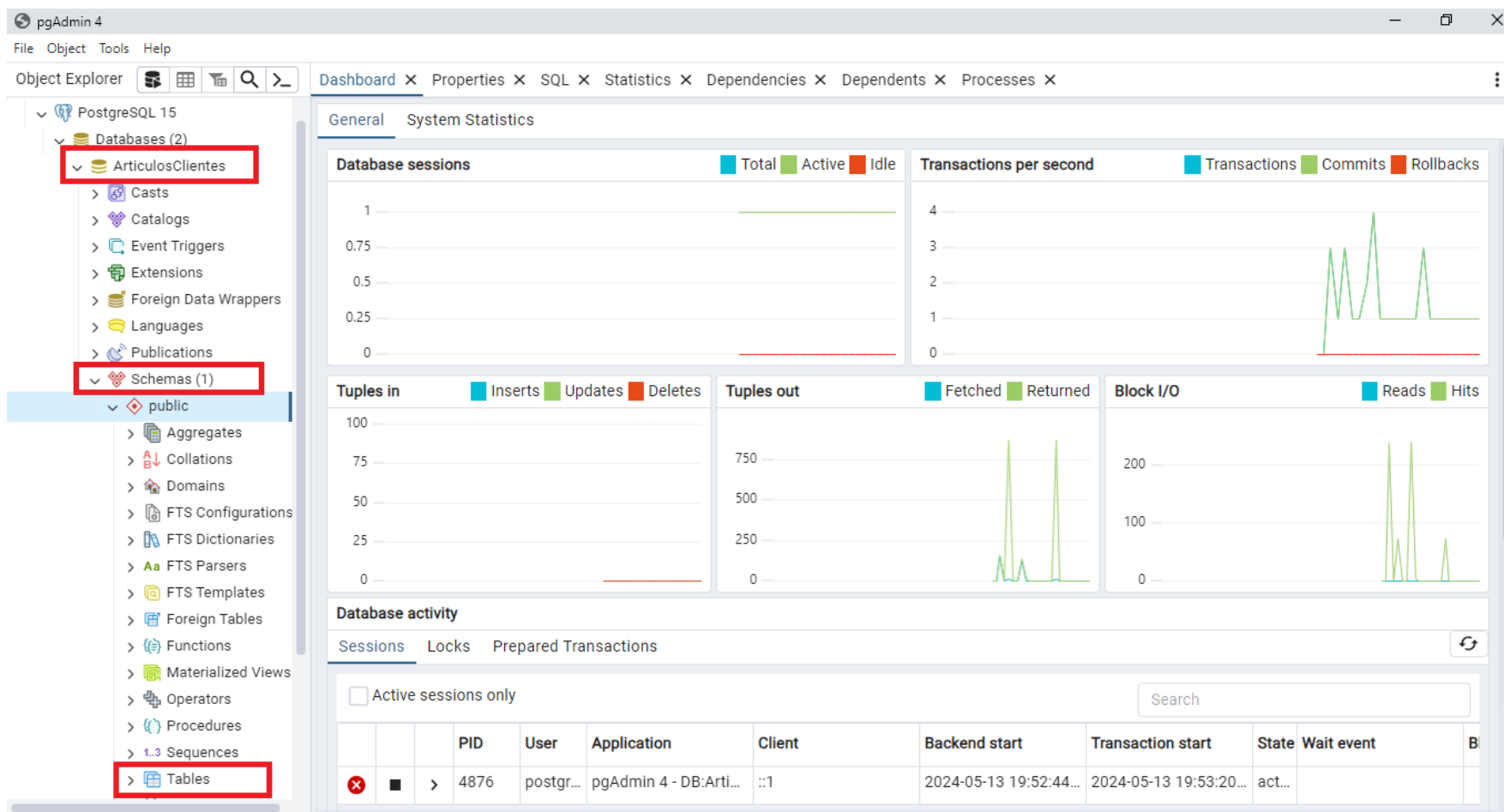
	PID	Database	User	Application	Client	Backend start	Transaction start	State	Wait event
✖	8516					2024-05-13 15:17:13...			Activity: Wal



The screenshot shows the 'Create - Database' dialog box in pgAdmin 4. The 'General' tab is selected. The 'Database' field is highlighted with a red rectangle and contains the text 'ArticulosClientes'. Other fields include 'OID', 'Owner' (set to 'postgres'), and 'Comment'. At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.

# Utilizar PostgreSQL con Django: Creando Database en pgAdmin 4

La base de datos creada aparece



The screenshot shows the pgAdmin 4 interface. On the left, the 'Object Explorer' pane displays the database structure. The 'Databases (2)' folder is expanded, showing 'ArticulosClientes' (highlighted with a red box). Under 'ArticulosClientes', the 'Schemas (1)' folder is expanded, showing the 'public' schema (highlighted with a red box). Below 'public', the 'Tables' folder is expanded (highlighted with a red box). The main pane shows the 'General' tab for the 'ArticulosClientes' database. It displays various system statistics: 'Database sessions' (Total: 1, Active: 0, Idle: 0), 'Transactions per second' (Transactions: 4, Commits: 0, Rollbacks: 0), 'Tuples in' (Inserts: 0, Updates: 0, Deletes: 0), 'Tuples out' (Fetched: 0, Returned: 0), and 'Block I/O' (Reads: 0, Hits: 0). At the bottom, the 'Database activity' section shows a table of active sessions.

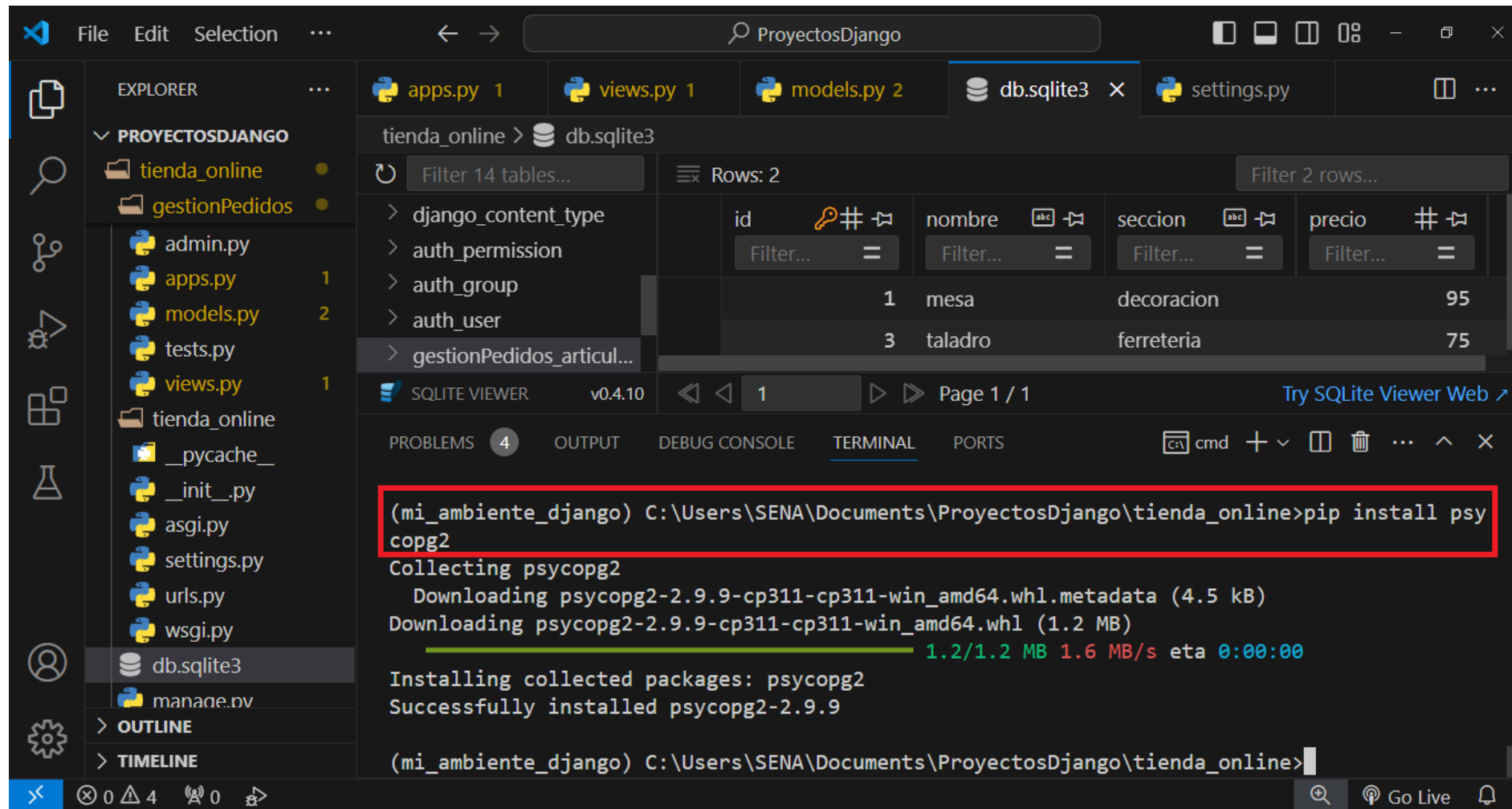
	PID	User	Application	Client	Backend start	Transaction start	State	Wait event	B
×	4876	postgr...	pgAdmin 4 - DB:Arti...	::1	2024-05-13 19:52:44...	2024-05-13 19:53:20...	act...		



# Utilizar PostgreSQL con Django: Pasos para conectar Django con PostgreSQL

1. Instalar el controlador de base de datos PostgreSQL para Python utilizando pip: **pip install psycopg2**.
2. Configurar la conexión a la base de datos en el archivo settings.py del proyecto Django. Esto implica agregar los detalles de la base de datos PostgreSQL, como el nombre de la base de datos, el usuario, la contraseña, el host y el puerto, dentro del diccionario DATABASES.
3. Ejecutar las migraciones del proyecto Django para crear las tablas en la base de datos PostgreSQL. Esto se hace ejecutando los comandos  
**python manage.py makemigrations**  
**python manage.py migrate.**

# 1, Instalar el controlador de base de datos PostgreSQL para Python utilizando pip y el comando `pip install psycopg2`.



The screenshot shows a Visual Studio Code editor with a Django project named 'ProyectosDjango'. The Explorer panel on the left shows the project structure, including a 'tienda\_online' app. The main editor area displays the 'db.sqlite3' database, showing a table named 'gestionPedidos\_articul...' with two rows of data:

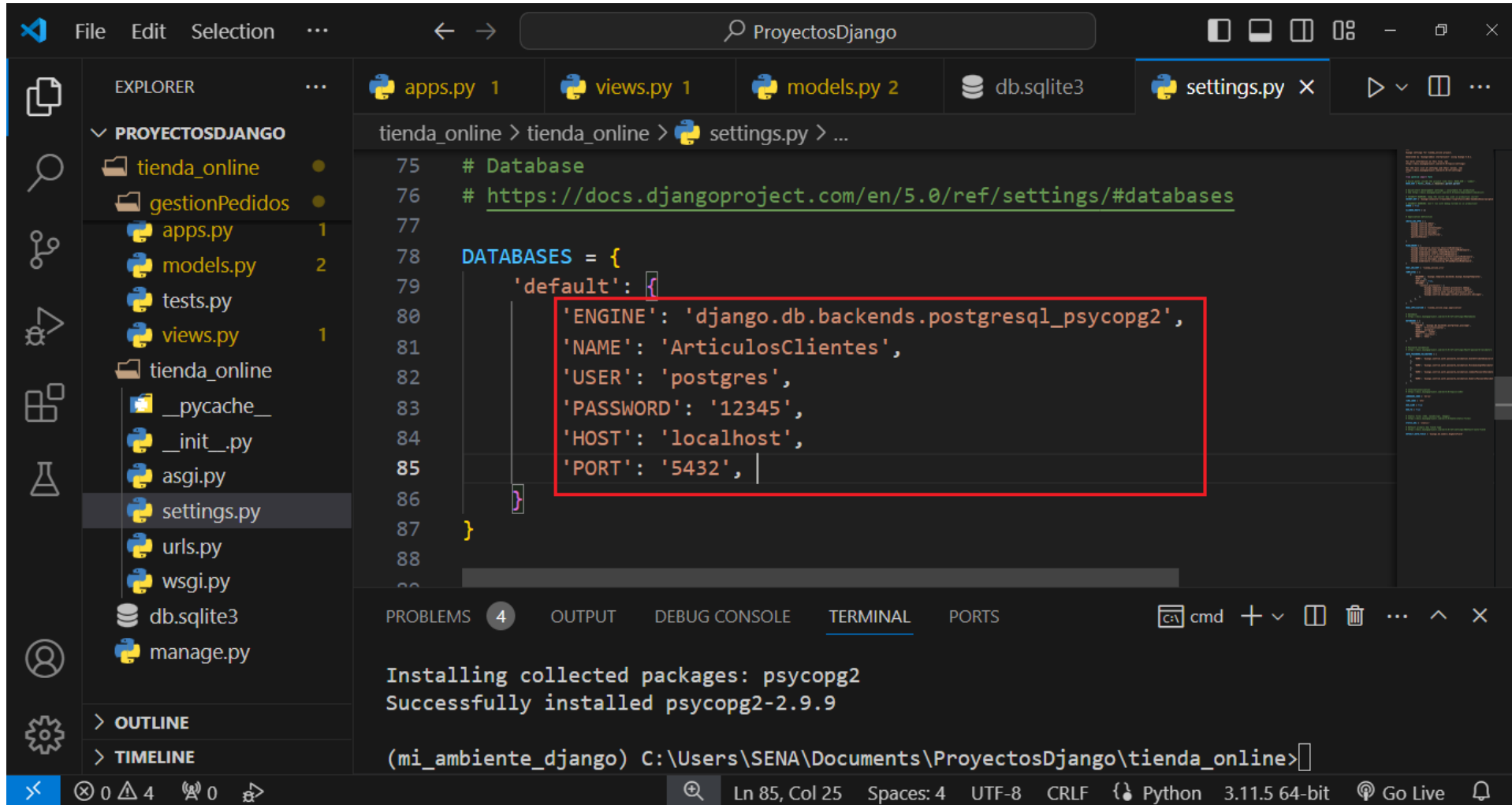
id	nombre	seccion	precio
1	mesa	decoracion	95
3	taladro	ferreteria	75

The bottom panel shows the 'TERMINAL' tab with the following output:

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>pip install psy
copg2
Collecting psycopg2
  Downloading psycopg2-2.9.9-cp311-cp311-win_amd64.whl.metadata (4.5 kB)
  Downloading psycopg2-2.9.9-cp311-cp311-win_amd64.whl (1.2 MB)
  1.2/1.2 MB 1.6 MB/s eta 0:00:00
Installing collected packages: psycopg2
Successfully installed psycopg2-2.9.9

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>
```

## 2. Configurar la conexión a la base de datos en el archivo settings.py del proyecto Django.



The screenshot shows the Visual Studio Code editor with the Django project 'ProyectosDjango' open. The file explorer on the left shows the project structure, including the 'tienda\_online' app. The main editor window displays the 'settings.py' file, where the database configuration is being set. A red box highlights the 'DATABASES' dictionary, specifically the 'default' entry, which is configured to use PostgreSQL. The terminal at the bottom shows the command prompt and the successful installation of the 'psycopg2' package.

```
75 # Database
76 # https://docs.djangoproject.com/en/5.0/ref/settings/#databases
77
78 DATABASES = {
79     'default': {
80         'ENGINE': 'django.db.backends.postgresql_psycopg2',
81         'NAME': 'ArticulosClientes',
82         'USER': 'postgres',
83         'PASSWORD': '12345',
84         'HOST': 'localhost',
85         'PORT': '5432',
86     }
87 }
```

Installing collected packages: psycopg2  
Successfully installed psycopg2-2.9.9

(mi\_ambiente\_django) C:\Users\SENA\Documents\ProyectosDjango\tienda\_online>

### 3. Ejecutar las migraciones del proyecto Django para crear las tablas en la base de datos PostgreSQL.

`python manage.py makemigrations`

```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS cmd + v [] [] ... ^ x

Installing collected packages: psycopg2
Successfully installed psycopg2-2.9.9

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.py makemigrations
No changes detected

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>
```

`python manage.py migrate.`

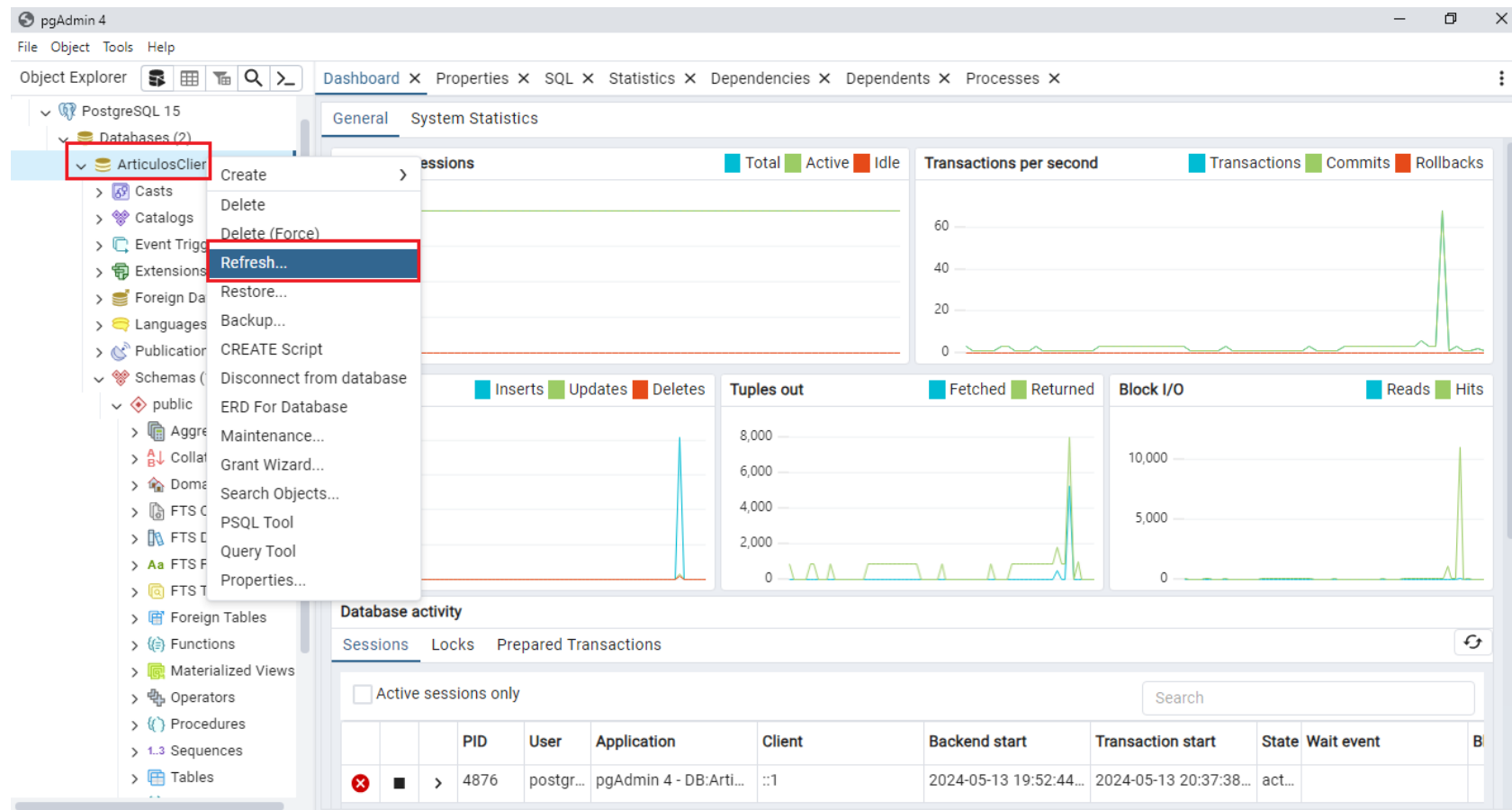
```
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS cmd + v [] [] ... ^ x

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.py migrate

Operations to perform:
  Apply all migrations: admin, auth, contenttypes, gestionPedidos, 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 contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
```

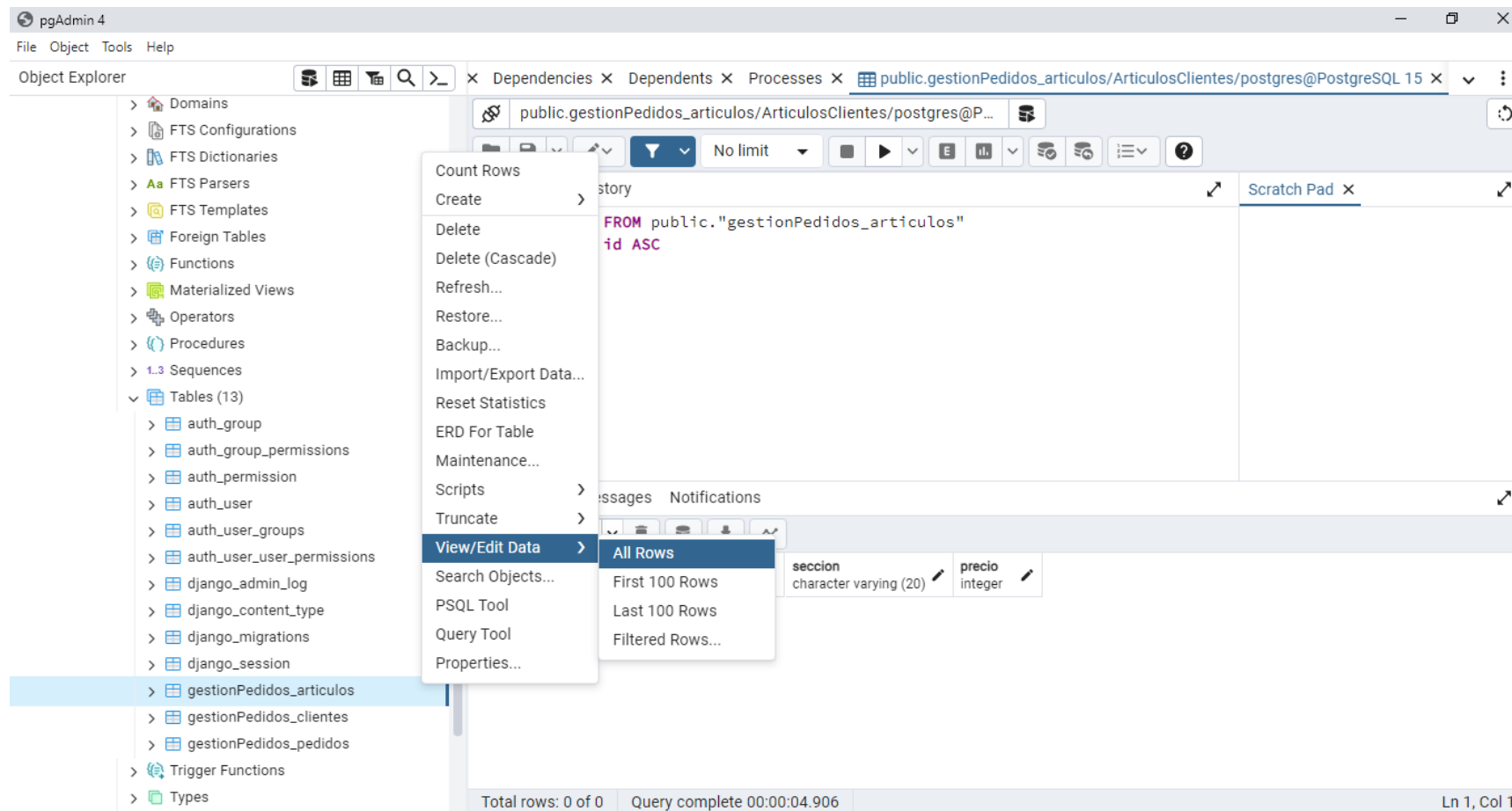
# Utilizar PostgreSQL con Django: Revisando Migraciones en pgAdmin 4

Vamos al gestor de base de datos y damos Clic contrario sobre la base de datos ArticulosClientes y damos Refresh...



# Utilizar PostgreSQL con Django: Revisando Migraciones en pgAdmin 4

Aparecen creadas las tablas de las migraciones



# Utilizar PostgreSQL con Django: Insertar registros

1. `python manage.py shell`
2. `from gestionPedidos.models import Articulos`
3. `cli=Clientes(nombre='Omar', direccion='calle 20', tfno='123456')`
4. `cli.save()`

# Utilizar PostgreSQL con Django: Insertar registros

pgAdmin 4

File Object Tools Help

Object Explorer

- Domains
- FTS Configurations
- FTS Dictionaries
- FTS Parsers
- FTS Templates
- Foreign Tables
- Functions
- Materialized Views
- Operators
- Procedures
- Sequences
- Tables (13)
  - auth\_group
  - auth\_group\_permissions
  - auth\_permission
  - auth\_user
  - auth\_user\_groups
  - auth\_user\_user\_permissions
  - django\_admin\_log
  - django\_content\_type
  - django\_migrations
  - django\_session
  - gestionPedidos\_articulos
  - gestionPedidos\_clientes**
  - gestionPedidos\_pedidos
- Trigger Functions
- Types

Dependencies Dependents Processes public.gestionPedidos\_clientes/ArticulosClientes/postgres@PostgreSQL 15

public.gestionPedidos\_clientes/ArticulosClientes/postgres@Po...

Query Query History Scratch Pad

```

1 SELECT * FROM public."gestionPedidos_clientes"
2 ORDER BY id ASC
  
```

Data Output Messages Notifications

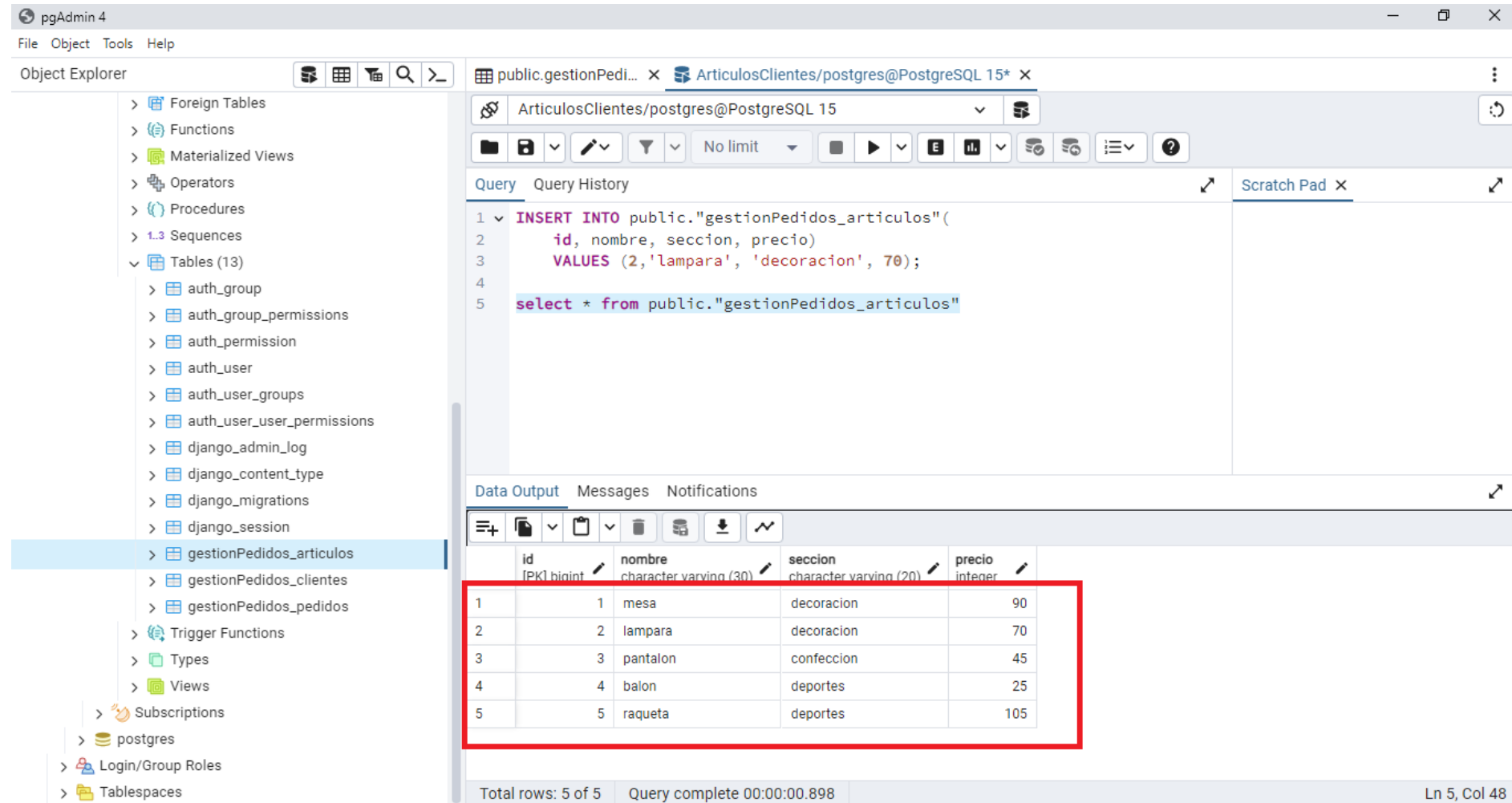
id	nombre	direccion	email	tfno
[PK] bigint	character varying (30)	character varying (50)	character varying (254)	character varying (7)
1	Omar	calle 20		123456

Total rows: 1 of 1 Query complete 00:00:00.257 Ln 2, Col 17



# Utilizar PostgreSQL con Django: Insertar registros

Insertamos algunos registros más desde el pgadmin 4:



The screenshot shows the pgAdmin 4 interface. On the left, the 'Object Explorer' sidebar displays the database structure, with the 'gestionPedidos\_articulos' table selected. The central pane shows a SQL query editor with the following query:

```
1 INSERT INTO public."gestionPedidos_articulos"(
2     id, nombre, seccion, precio)
3     VALUES (2,'lampara', 'decoracion', 70);
4
5 select * from public."gestionPedidos_articulos"
```

Below the query editor, the 'Data Output' tab displays the results of the query in a table format. The table has 5 rows and 4 columns: 'id', 'nombre', 'seccion', and 'precio'. The data is as follows:

	id	nombre	seccion	precio
1	1	mesa	decoracion	90
2	2	lampara	decoracion	70
3	3	pantalon	confeccion	45
4	4	balon	deportes	25
5	5	raqueta	deportes	105

The bottom status bar indicates 'Total rows: 5 of 5' and 'Query complete 00:00:00.898'.

# Utilizar PostgreSQL con Django: Consultar Registros

1. `python manage.py shell`
2. `from gestionPedidos.models import Articulos`
3. `Articulos.objects.filter(seccion='deportes')`

Respuesta: `<QuerySet [<Articulos: Articulos object (4)>, <Articulos: Articulos object (5)>]>`

Para mejorar esto incluimos en `class Articulos(models.Model)`: el siguiente método:

```
class Articulos(models.Model):
    nombre=models.CharField(max_length=30)
    seccion=models.CharField(max_length=20)
    precio=models.IntegerField()

    def __str__(self):
        return f"{self.nombre} - {self.seccion} - {self.precio}"
```

# Utilizar PostgreSQL con Django: Consultar Registros

Como hubo cambios en el modelo se deben generar las migraciones. Para ello salimos del Shell con `exit()` y ejecutamos:

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

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.p
y makemigrations
No changes detected

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.p
y migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, gestionPedidos, sessions
Running migrations:
  No migrations to apply.

(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>
```

# Utilizar PostgreSQL con Django: Consultar Registros

Volvemos a ingresar en el Shell y ejecutamos:

1. `python manage.py shell`
2. `from gestionPedidos.models import Articulos`
3. `Articulos.objects.filter(seccion='deportes')`

```
(mi_ambiente_django) C:\Users\SENA\Documents\ProyectosDjango\tienda_online>python manage.p
y shell
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from gestionPedidos.models import Articulos
>>> Articulos.objects.filter(seccion='deportes')
<QuerySet [<Articulos: balon - deportes - 25>, <Articulos: raqueta - deportes - 105>]>
>>> █
```

# Utilizar PostgreSQL con Django: Consultar Registros

- Consultar la mesa que se encuentra en decoración  
**`Articulos.objects.filter(nombre='mesa', seccion='decoracion')`**
- Consulta para mostrar los artículos cuyo precio sea mayor que 50  
**`Articulos.objects.filter(precio__gt=50)`**
- Consulta para mostrar los artículos cuyo precio sea mayor que 50 ordenados de menor a mayor  
**`Articulos.objects.filter(precio__gt=50).order_by('precio')`**
- Consulta para mostrar los artículos cuyo precio sea mayor que 50 ordenados de mayor a menor  
**`Articulos.objects.filter(precio__gt=50).order_by('-precio')`**
- Consulta para mostrar los artículos cuyo nombre comience con la letra "m"  
**`Articulos.objects.filter(nombre__startswith='m')`**
- Consulta para mostrar los artículos cuyo precio esté en el rango de 45 a 90  
**`Articulos.objects.filter(precio__gte=45, precio__lte=90)`**

# Utilizar PostgreSQL con Django: Consultar Registros

El Objeto Q: El objeto Q en Django es una clase que permite construir consultas complejas utilizando operadores lógicos como OR (|) y AND (&). Se utiliza principalmente para construir condiciones de filtrado más complejas en consultas de base de datos. Este objeto debe ser importado así desde el Shell:

```
from django.db.models import Q
```

- Consulta de artículos que pertenecen a la sección de deportes o confección

```
Articulos.objects.filter(Q(seccion='deportes') | Q(seccion='confeccion'))
```

- Consulta para obtener los artículos cuyo precio es mayor que 50 y cuya sección es "decoracion" o "confeccion"

```
Articulos.objects.filter(Q(precio__gt=50) & (Q(seccion='decoracion') | Q(seccion='confeccion')))
```

- Consulta para obtener los artículos cuyo precio es menor o igual a 70 y cuya sección es "confeccion" o "deportes"

```
Articulos.objects.filter(Q(precio__lte=70) & (Q(seccion='confeccion') | Q(seccion='deportes')))
```

# Taller

## Sin objeto Q:

1. Mostrar todos los artículos.
2. Mostrar los artículos cuyo precio sea mayor que 50.
3. Mostrar los artículos cuya sección sea "decoracion".
4. Mostrar los artículos cuyo nombre empiece con la letra "p".
5. Mostrar los artículos cuyo nombre contenga la palabra "mesa".
6. Mostrar los artículos cuyo precio esté entre 30 y 70.

## Con objeto Q:

1. Mostrar los artículos cuya sección sea "deportes" o "confeccion".
2. Mostrar los artículos cuyo precio sea mayor que 50 y cuya sección sea "decoracion" o "confeccion".
3. Mostrar los artículos cuyo precio sea menor o igual a 70 y cuya sección sea "confeccion" o "deportes".
4. Mostrar los artículos cuyo precio sea exactamente 45 o 105 y cuya sección sea "decoracion" o "deportes".



**G R A C I A S**

Línea de atención al ciudadano: 01 8000 910270  
Línea de atención al empresario: 01 8000 910682



[www.sena.edu.co](http://www.sena.edu.co)