

**Los comandos descritos son para Debian, para windows los comandos pueden cambiar “ligereamente”.**

**1. Instalar pip (si aún no está instalado):**

Primero, necesitarás asegurarte de tener `pip`, el gestor de paquetes de Python, instalado. Puedes instalarlo con el siguiente comando:

```
sudo apt-get install python3-pip
```

**2. Instalar Virtualenv:**

Es una buena práctica utilizar un entorno virtual para tus proyectos de Django. Esto asegura que las dependencias de tu proyecto estén aisladas. Puedes instalar Virtualenv usando `pip`:

```
pip3 install virtualenv
```

**3. Crear un Entorno Virtual:**

Una vez instalado Virtualenv, **navega a la carpeta donde deseas crear tu proyecto y ejecuta:**

```
python3 -m venv api_env
```

Esto creará un nuevo entorno virtual llamado `api_env`.

**4. Activar el Entorno Virtual:**

Para activar el entorno virtual, ejecuta:

```
source api_env/bin/activate
```

Verás que el nombre de tu entorno virtual aparece en el prompt, indicando que está activo.

**5. Instalar Django:**

Con tu entorno virtual activado, instala Django usando `pip`:

```
pip install django
```

**6. Crear un Nuevo Proyecto en Django:**

Ahora puedes crear un nuevo proyecto de Django con el comando:

```
django-admin startproject djangocrud .
```

**7. Iniciar un Servidor de Desarrollo:**

Puedes iniciar el servidor de desarrollo de Django con el siguiente comando:

```
python manage.py runserver
```

Esto iniciará un servidor en `http://127.0.0.1:8000/` por defecto. Puedes abrir un navegador y visitar esa dirección para ver tu nuevo proyecto de Django en acción.

**Procedimiento para crear la primera aplicación:**

## Manual Django

Ahora que ya tienes tu proyecto Django configurado, puedes proceder a crear tu primera aplicación dentro de este proyecto. Una aplicación en Django es un componente que puede manejar una funcionalidad específica de tu proyecto.

### 1. Navegar al Directorio del Proyecto:

Asegúrate de estar en el directorio raíz de tu proyecto Django (donde se encuentra el archivo `manage.py`). Si ya estás allí desde los pasos anteriores, perfecto.

### 2. Crear la Aplicación:

Ejecuta el siguiente comando para crear una nueva aplicación. Reemplaza `mi_app` con el nombre que desees para tu aplicación:

```
python manage.py startapp mi_app
```

### 3. Estructura de la Aplicación:

Después de ejecutar el comando, se creará un nuevo directorio con el nombre de tu aplicación (`mi_app` en este caso), que incluirá varios archivos. Estos archivos son para diferentes propósitos como modelos, vistas, pruebas, etc.

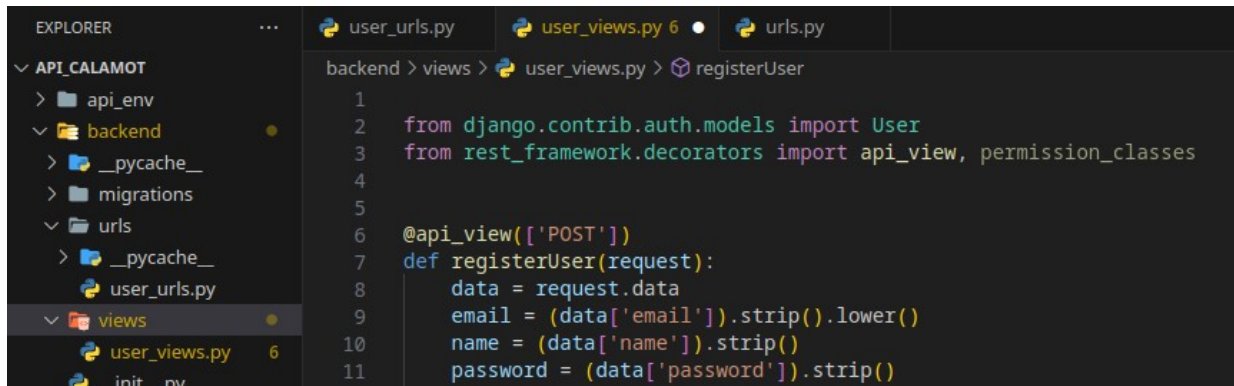
### 4. Registrar la Aplicación en tu Proyecto:

Para que tu proyecto Django reconozca la nueva aplicación, debes agregarla a la configuración. Abre el archivo `settings.py` en el directorio de tu proyecto Django. Busca la lista `INSTALLED_APPS` y agrega el nombre de tu aplicación. Debería verse algo así:

```
INSTALLED_APPS = [  
    # otras apps instaladas  
    'mi_app',  
]
```

### 5. Crear Vistas:

Las vistas se definen en el archivo `views.py` de tu aplicación. Pero vamos a darle un enfoque mas escalable así que elimina el archivo `views.py`. Crea una carpeta `views` y dentro de la carpeta crea un archivo llamado `user_views.py`.



The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORER' panel displays a file tree for a project named 'API\_CALAMOT'. The tree includes folders like 'api\_env', 'backend', 'migrations', and 'urls', along with files like 'user\_urls.py', 'user\_views.py', and 'init.py'. The 'views' folder is expanded, showing 'user\_views.py' with a line number of 6. The main editor area shows the code for 'registerUser' in 'user\_views.py'. The code imports 'User' from 'django.contrib.auth.models' and 'api\_view' and 'permission\_classes' from 'rest\_framework.decorators'. It uses the '@api\_view(['POST'])' decorator and defines a 'registerUser' function that takes a 'request' and extracts 'email', 'name', and 'password' from the request data, stripping whitespace and lowercasing the email.

```
1
2 from django.contrib.auth.models import User
3 from rest_framework.decorators import api_view, permission_classes
4
5
6 @api_view(['POST'])
7 def registerUser(request):
8     data = request.data
9     email = (data['email']).strip().lower()
10    name = (data['name']).strip()
11    password = (data['password']).strip()
```

### Instalar Django REST Framework y djangorestframework-simplejwt:

Asegúrate de que tu entorno virtual esté activado, y luego instala Django REST framework y djangorestframework-simplejwt usando pip:

```
pip install djangorestframework
pip install djangorestframework-simplejwt
```

### Agregar Django REST Framework a tu Proyecto:

Al igual que hiciste con tu aplicación, necesitas registrar Django REST framework en tu proyecto Django. Abre el archivo `settings.py` y agrega 'rest\_framework' a la lista **INSTALLED\_APPS**:

```
INSTALLED_APPS = [  
    # otras apps instaladas  
    'rest_framework',  
    'mi_app', # Tu aplicación personal  
]  
  
REST_FRAMEWORK = {  
    'DEFAULT_AUTHENTICATION_CLASSES': (  
        'rest_framework_simplejwt.authentication.JWTAuthentication',  
    )  
}
```

#Config JWT

```
SIMPLE_JWT = {  
    'ACCESS_TOKEN_LIFETIME': timedelta(days=30),  
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),  
    'ROTATE_REFRESH_TOKENS': False,  
    'BLACKLIST_AFTER_ROTATION': False,  
    'UPDATE_LAST_LOGIN': False,  
  
    'ALGORITHM': 'HS256',  
    'SIGNING_KEY': SECRET_KEY,  
    'VERIFYING_KEY': None,  
    'AUDIENCE': None,  
    'ISSUER': None,  
    'JWK_URL': None,  
    'LEEWAY': 0,  
  
    'AUTH_HEADER_TYPES': ('Bearer',),  
    'AUTH_HEADER_NAME': 'HTTP_AUTHORIZATION',  
    'USER_ID_FIELD': 'id',  
    'USER_ID_CLAIM': 'user_id',  
    'USER_AUTHENTICATION_RULE':  
        'rest_framework_simplejwt.authentication.default_user_authentication_rule',  
  
    'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken',),  
    'TOKEN_TYPE_CLAIM': 'token_type',  
    'TOKEN_USER_CLASS': 'rest_framework_simplejwt.models.TokenUser',  
  
    'JTI_CLAIM': 'jti',  
  
    'SLIDING_TOKEN_REFRESH_EXP_CLAIM': 'refresh_exp',  
    'SLIDING_TOKEN_LIFETIME': timedelta(minutes=5),  
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),  
}
```

## Manual Django

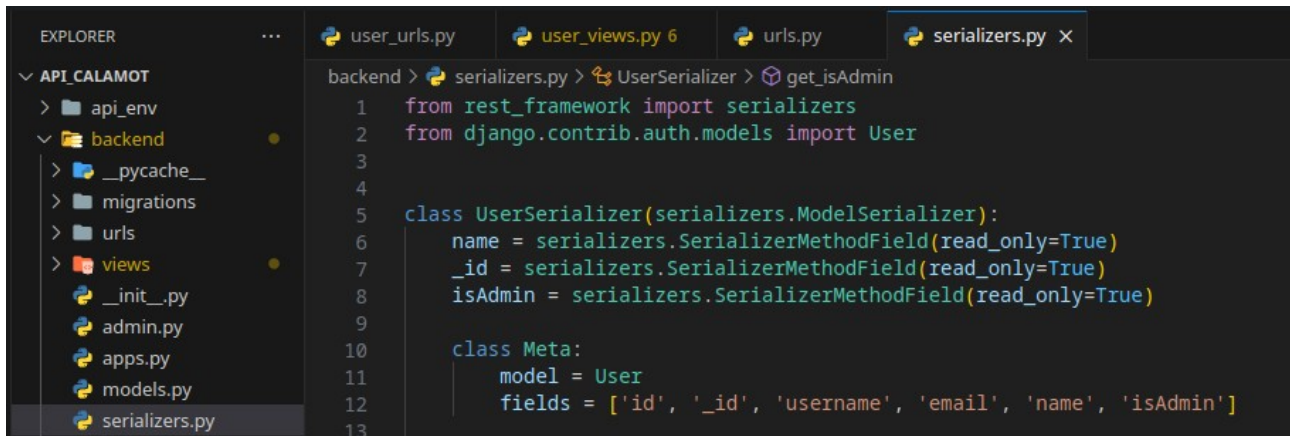
```
#code for user_views.py
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from rest_framework import status
from django.contrib.auth.models import User
from django.contrib.auth.hashers import make_password
from backend.serializers import UserSerializer, UserSerializerWithToken

@api_view(['POST'])
def registerUser(request):
    data = request.data
    email = (data['email']).strip().lower()
    name = (data['name']).strip()
    password = (data['password']).strip()

    try:
        user = User.objects.create(
            first_name=name,
            username=email,
            email=email,
            password=make_password(password)
        )
        serializer = UserSerializerWithToken(user, many=False)
        print(f'Usuario registrado con éxito: {email}.')
        return Response(serializer.data)
    except Exception as e:
        print(f'Error al registrar usuario: {str(e)}.')
        message = {'detail': 'La información proporcionada no es válida, revisa el formato de tu correo'}
        return Response(message, status=status.HTTP_400_BAD_REQUEST)
```

## Crear Serializadores:

Los serializadores en Django REST framework te permiten convertir complejas consultas de datos y tipos de datos en Python a JSON y viceversa. Crea un archivo **serializers.py** en tu aplicación y define un serializador para tus modelos. Por ejemplo:



```
# Code for serializers.py
from rest_framework import serializers
from django.contrib.auth.models import User
from rest_framework_simplejwt.tokens import RefreshToken
```

```
class UserSerializer(serializers.ModelSerializer):
    name = serializers.SerializerMethodField(read_only=True)
    _id = serializers.SerializerMethodField(read_only=True)
    isAdmin = serializers.SerializerMethodField(read_only=True)
```

```
class Meta:
    model = User
    fields = ['id', '_id', 'username', 'email', 'name', 'isAdmin']
```

```
def get__id(self, obj):
    return obj.id
```

```
def get_isAdmin(self, obj):
    return obj.is_staff
```

```
def get_name(self, obj):
    name = obj.first_name
    if name == "":
        name = obj.email
    return name
```

```
class UserSerializerWithToken(UserSerializer):
    token = serializers.SerializerMethodField(read_only=True)
```

```
class Meta:
    model = User
    fields = ['id', '_id', 'username', 'email', 'name', 'isAdmin', 'token']
```

```
def get_token(self, obj):
    token = RefreshToken.for_user(obj)
    return str(token.access_token)
```

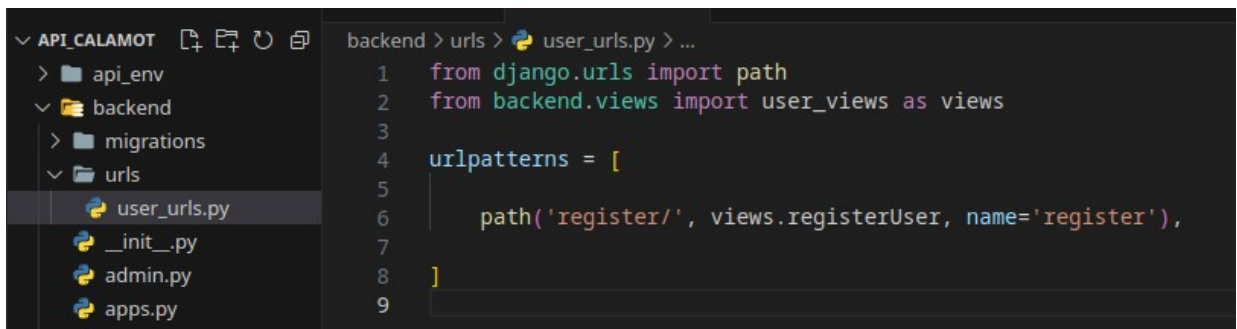
## 6. Configurar URLs:

Necesitarás configurar una URL para tu vista. Primero, crea una carpeta `urls` en el directorio de tu aplicación.

Luego, crea un fichero `user_urls.py`:

```
# Code for user_urls.py
from django.urls import path
from backend.views import user_views as views

urlpatterns = [
    path('register/', views.registerUser, name='register'),
]
```

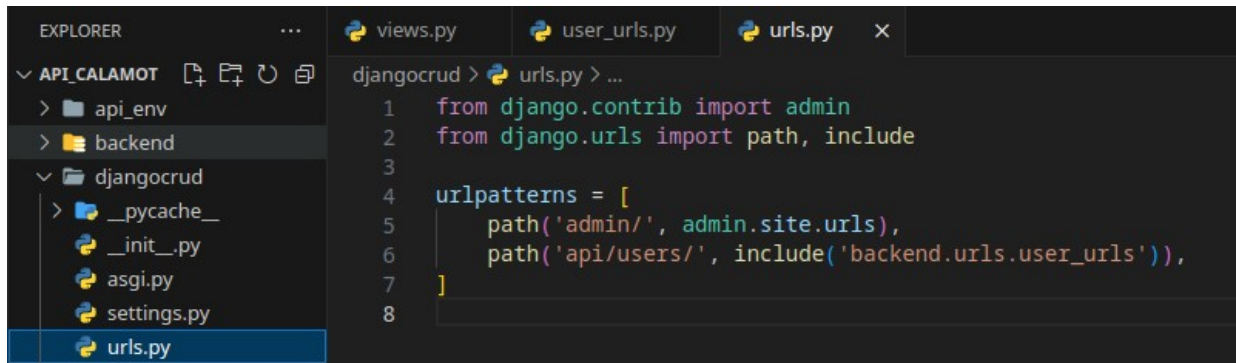


Luego, incluye esta URL en las URL globales de tu proyecto (`urls.py` en el directorio del proyecto).

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/users/', include('backend.urls.user_urls')),
]
```

## Manual Django



### Creación del super usuario

Después de configurar django, debemos crear el super usuario para acceder a la consola de administrador que nos proporciona Django. Antes de lanzar el comando para crear el super usuario debemos realizar las migraciones necesarias para que el ORM de Django haga su magia y configure y cree la base de datos.

7. **Aplicar Migraciones:** Ejecuta el siguiente comando para aplicar todas las migraciones pendientes. Esto creará las tablas necesarias en tu base de datos, incluyendo la tabla `auth_user` que es requerida para el modelo `User`.

```
python manage.py migrat
```

Este comando ejecutará todas las migraciones pendientes para las apps `admin`, `auth`, `contenttypes`, `sessions` y cualquier otra app que tenga migraciones pendientes en tu proyecto.

8. **Verificar Migraciones:** Después de aplicar las migraciones, puedes verificar que todas estén aplicadas correctamente con el comando:

```
python manage.py showmigrations
```



## Manual Django

Esto mostrará una lista de todas las migraciones junto con marcas que indican si han sido aplicadas o no.

```
admin
[X] 0001_initial
[X] 0002_logentry_remove_auto_add
[X] 0003_logentry_add_action_flag_choices
auth
[X] 0001_initial
[X] 0002_alter_permission_name_max_length
[X] 0003_alter_user_email_max_length
[X] 0004_alter_user_username_opts
[X] 0005_alter_user_last_login_null
[X] 0006_require_contenttypes_0002
[X] 0007_alter_validators_add_error_messages
[X] 0008_alter_user_username_max_length
[X] 0009_alter_user_last_name_max_length
[X] 0010_alter_group_name_max_length
[X] 0011_update_proxy_permissions
[X] 0012_alter_user_first_name_max_length
backend
(no migrations)
contenttypes
[X] 0001_initial
[X] 0002_remove_content_type_name
sessions
[X] 0001_initial
```

### Crear el Superusuario:

Una vez que todas las migraciones estén aplicadas, intenta crear el superusuario con el comando:

```
python manage.py createsuperuser
```

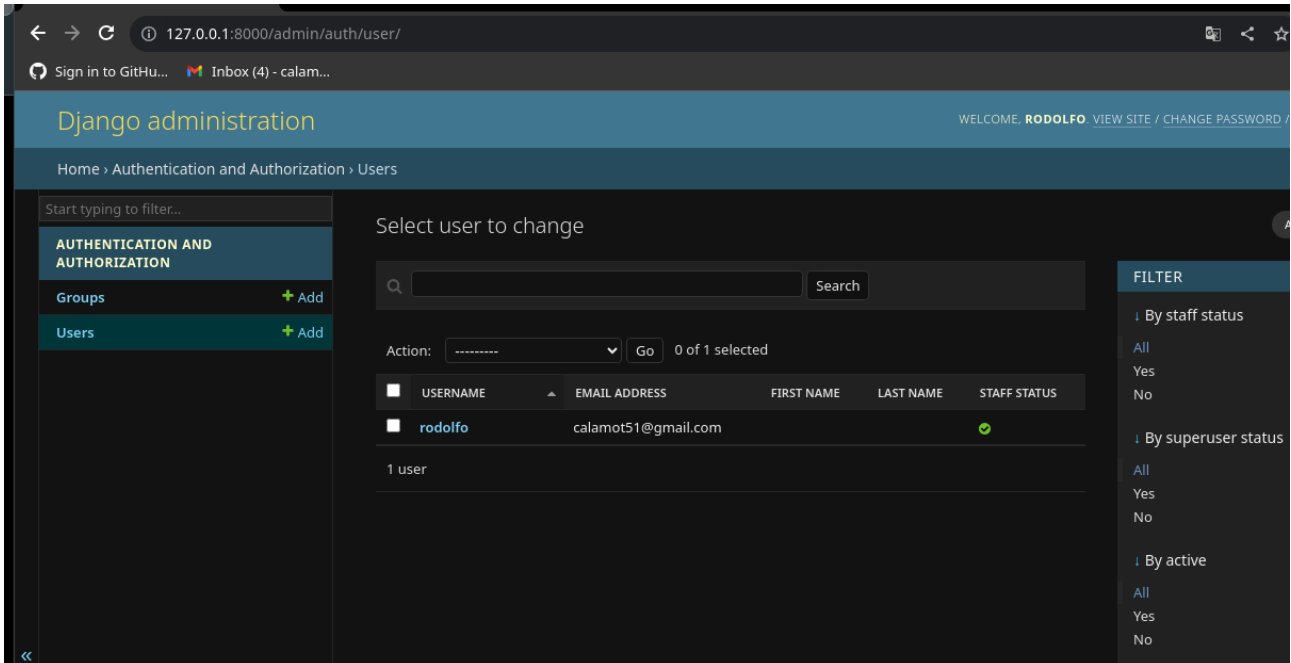
Ahora deberías poder seguir los pasos sin encontrar errores, ingresando un nombre de usuario, email (si tu configuración lo requiere) y una contraseña.

User: rodolfo

Pass: 1234

## Manual Django

¡Y listo! Ahora tienes tu primera aplicación funcionando en tu proyecto Django. Puedes continuar desarrollando más modelos, vistas, y templates según las necesidades de tu proyecto.



## Testeo del endpoint para el registro de usuarios:

Ya tenemos el registro de usuarios configurado y funcionando. Podéis probar a crear un usuario con postman, es un cliente para probar apis.

**Para iniciar un servidor de desarrollo de Django usando tu dirección IP local, debes especificar la IP y el puerto al ejecutar el comando `runserver`. Aquí tienes los pasos:**

### 1. Encontrar tu dirección IP local:

Primero, necesitas saber cuál es tu dirección IP local. Esta dirección varía según tu red y sistema operativo.

### 2. Ejecutar el Servidor de Django:

Una vez que tengas tu dirección IP local, puedes utilizarla con el comando `runserver`. Por ejemplo, si tu dirección IP local es `192.168.1.5`, el comando sería:

```
python manage.py runserver 192.168.1.5:8000
```

Aquí, `8000` es el puerto que se utiliza por defecto. Puedes cambiarlo a otro número de puerto si lo deseas.



## Manual Django

### Testear la url de inicio de sesión:

#### Probar la url de inicio de sesión:

Para probar la url del inicio de sesión solo debemos modificar el archivo `user_urls.py` que cuelga de nuestra aplicación:

```
from django.urls import path
```

```
from backend.views import user_views as views
```

```
urlpatterns = [  
    path('login/', views.MyTokenObtainPairView.as_view(),  
         name='token_obtain_pair'),  
    path('register/', views.registerUser, name='register'),  
]
```

Probando con postman

The screenshot shows the Postman interface with a POST request to `http://192.168.1.158:8000/api/users/login/`. The request body is in JSON format with the following data:

Key	Value	Description
username	test@hotmail.com	
password	123Abcd!	

The response status is 200 OK, with a time of 223 ms and a size of 1.11 KB. The response body is in JSON format, showing a successful login with a refresh token, access token, and user details.

```
{  
  "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
    eyJ0b2t1b190eXB1IjoicmVmcVzaCisImV4cCI6MTcwMDkyNTYyOSwiaWF0IjoxNzAwODM5MjI0OTI0ZTFmYmY1N2E0OWE3MDkyZWU3M1IsInV  
    zZXJfaWQ1OjJ9.c88I82oGQvnzcbK_Xh58BTE5HaCAfy4Ey3J6hwve7Vc",  
  "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
    eyJ0b2t1b190eXB1IjoicmVmcVzaCisImV4cCI6MTcwMDkyNTYyOSwiaWF0IjoxNzAwODM5MjI0OTI0ZTFmYmY1N2E0OWE3MDkyZWU3M1IsInV  
    lc19pZCI6Mn0.sNnh8SBaoShfFg5-0MADpg1XuqhrdkYm-wEbb6y9-iE",  
  "id": 2,  
  "username": "test@hotmail.com",  
  "email": "test@hotmail.com",  
  "name": "test",  
  "isAdmin": false,  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
    eyJ0b2t1b190eXB1IjoicmVmcVzaCisImV4cCI6MTcwMDkyNTYyOSwiaWF0IjoxNzAwODM5MjI0OTI0ZTFmYmY1N2E0OWE3MDkyZWU3M1IsInV  
    lc19pZCI6Mn0.Qn812yUBFpLYHmfS_kENKhAqG89DVtdOnCZSVdoukdk"
```

**Cuando termineis el CRUD, procederemos con los modelos.**

**Crear Modelos *(Para el CRUD de usuarios no hace falta)*:**

Utilizar modelos (representaciones de tablas de base de datos), **debes** definirlos en el archivo `models.py` dentro de tu aplicación. Por ejemplo:

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

```
class MiModelo(models.Model):  
    mi_campo = models.CharField(max_length=100)
```

No olvides realizar las migraciones después de definir tus modelos.