

# Apuntes de Django

Ezequiel Remus: < *ezequielremus@gmail.com* >

# Resumen

Este documento esta basado en la documentación de [Django](#) . En particular corresponde a la version Django 3.0, que admite Python 3.6 y versiones posteriores.

## Índice

<b>1. Introducción</b>	<b>3</b>
1.1. ¿Que es Django?	3
1.2. ¿Porqué usarlo?	3
<b>2. Iniciando un proyecto</b>	<b>3</b>
2.1. Entornos Virtuales	3
<b>3. Comenzando con Django</b>	<b>4</b>
3.1. Escribiendo tu primera aplicación Django , parte 1	4
3.1.1. Creando un Proyecto	4
3.1.2. El servidor de desarrollo	5
3.1.3. Creando la aplicación de encuestas (polls)	6
3.1.4. Escribiendo la primer vista (view)	7
3.2. Escribiendo tu primera aplicación Django , parte 2	9
3.2.1. Configuración de la Base de Datos	9
3.2.2. Creando un Modelo	10
3.2.3. Modelos de Activación	11
3.2.4. Jugando con la API	14
3.2.5. Introducción al administrador de Django	17
<b>4. Referencias</b>	<b>21</b>
4.1. Refes de las partes de los tutoriales	21
4.2. Refes W3School	21
4.3. Atajos	21

# 1. Introducción

## 1.1. ¿Que es Django?

**Django** es un framework web diseñado para realizar aplicaciones de cualquier complejidad en unos tiempos muy razonables.

Está escrito en **Python** y tiene una comunidad muy amplia, que está en continuo crecimiento

## 1.2. ¿Porqué usarlo?

Los motivos principales para usar **Django** son:

- Es muy rápido : Si tenés una startup, estas apurado por terminar un proyecto proyecto o, simplemente, querés reducir costes, con **Django** *podéis construir una aplicación muy buena en poco tiempo.*
- Viene bien cargado : Cualquier cosa que necesitéis realizar, ya estará implementada, sólo hay que adaptarla a vuestras necesidades. Ya sea porque hay módulos de la comunidad, por cualquier paquete **Python** que encontréis o las propias aplicaciones que **Django** trae, que son muy útiles.
- Es bastante seguro : Podemos estar tranquilos con **Django** , ya que implementa por defecto algunas medidas de seguridad, las más clásicas, para que no haya **SQL Injection**, no haya *Cross site request forgery (CSRF)* o no haya **Clickjacking** por *JavaScript*. **Django** se encarga de manejar todo esto de una manera realmente sencilla.
- Es muy escalable : Podemos pasar desde muy poco a una aplicación enorme perfectamente, una aplicación que sea modular, que funcione rápido y sea estable.
- Es increíblemente versátil : Es cierto que en un principio **Django** comienza siendo un Framework para almacenar noticias por sitios de prensa, blogs y este estilo de webs, pero con el tiempo ha ganado tanta popularidad que se puede usar para el propósito que queráis.

*Otras bondades de **Django** que no se destacan en la web son:*

Su **ORM**, su interfaz para acceso a la base de datos , ya que hacer consultas con ella es una maravilla, es una herramienta muy buena.

Trae de serie un panel de administración, con el cual podemos dejar a personas sin ningún tipo de conocimiento técnico manejando datos importantes de una forma muy cómoda

# 2. Iniciando un proyecto

## 2.1. Entornos Virtuales

Lo primero y más importante es asegurarnos de crear un entorno para trabajar en nuestro proyecto.

Un entorno virtual es básicamente una abstracción la cual crea un conjunto vacío en **Python** , donde solo esta instalada la versión de **Python** que se utiliza junto con **pip** y las librerías básicas. En este conjunto podremos instalar todas las librerías que utilizaremos en el proyecto. Esto nos permitirá crear luego un archivo de referencia para conocer las librerías y versiones de estas utilizadas por el proyecto.

Existen varias formas de crear entornos virtuales:

### ■ Anaconda:

Es una distribución libre y abierta de los lenguajes **Python** y **R**, utilizada en ciencia de datos y aprendizaje automático (*Machine Learning*). Esto incluye procesamiento de grandes volúmenes de información, análisis predictivo y cómputos científicos. Esta orientado a simplificar el despliegue y administración de los paquetes de software.

Las diferentes versiones de los paquetes se administran mediante el sistema de gestión de paquetes de **conda**, el cual lo hace bastante sencillo de instalar, correr y actualizar software de ciencia de datos y machine learning como ser *Scikit-team*, *Tensorflow* y *Scipy*.

La distribución Anaconda incluye más de 250 paquetes de ciencia de datos validos para **Windows**, **Linux** y **MacOs**.

**Referencias:** <https://docs.anaconda.com>

- **Virtualenv**: Es una herramienta para crear entornos de **Python** aislados, es decir entornos donde las librerías o las versiones de **Python** no interfieren con las carpetas que **Python** tiene por defecto en la máquina. Haciendo una analogía con un edificio, un entorno vendría siendo como una planta, usa ciertos recursos como el agua o la energía eléctrica (para el caso de **Python** usa la misma máquina) y a su vez cada planta tiene sus propios recursos, tales como los muebles, las habitaciones y demás (para el caso de **python** hablamos de librerías.)
- **pyenv**(Linux):

## 3. Comenzando con Django

### 3.1. Escribiendo tu primera aplicación Django , parte 1

A lo largo de este tutorial, nos guiaremos a través de la creación de una aplicación de encuesta básica. Consiste de dos partes:

- Un sitio público que permite a las personas ver encuestas y votar en ellas.
- Un sitio de administración que le permite agregar, cambiar y eliminar encuestas (*polls*).

Asumiremos que ya tiene **Django** instalado. Puede decir que **Django** está instalado y qué versión ejecuta el siguiente comando en un indicador de **shell** (indicado por el prefijo \$):

```
$ python -m django --version
```

Si **Django** está instalado, debería ver la versión de su instalación. Si no es así, recibirá un error que dice *"No module named django"* (Ningún módulo llamado django).

#### 3.1.1. Creando un Proyecto

Si es la primera vez que usa **Django** , tendrá que encargarse de la configuración inicial. Es decir, deberá generar automáticamente un código que establezca un proyecto de **Django** : una colección de configuraciones para una instancia de **Django** , incluida la configuración de la base de datos, las opciones específicas de **Django** y las configuraciones específicas de la aplicación.

Desde la línea de comando, Entre en un directorio donde le gustaría almacenar su código, luego ejecute el siguiente comando

```
$ django-admin startproject mysite
```

Esto creará un directorio **mysite** en su directorio actual. Si no funcionó, vea Problemas al ejecutar *django-admin*: <https://docs.djangoproject.com/en/3.0/faq/troubleshooting/#troubleshooting-django-admin>.

#### ¿Donde deberia estar este codigo?

Si su fondo está en **PHP** antiguo (sin el uso de marcos modernos), probablemente esté acostumbrado a poner código debajo de la raíz de documentos del servidor web (en un lugar como **/var/www**).

Con **Django** , no es necesario hacer eso. No es una buena idea colocar ninguno de estos códigos de **Python** en la raíz de documentos de su servidor web, ya que se corre el riesgo de que las personas puedan ver su código en la Web, lo cual no es bueno para la seguridad. Coloque su código en algún directorio fuera de la raíz del documento, como **/home/mycode**.

Al entrar a la carpeta **mysite** creada por el comando *startproject*, se crea el árbol de archivos de la figura de acá al costado

#### Nota:

Deberá evitar nombrar proyectos después de los componentes integrados de **Python** o Django. En particular, esto significa que debe evitar el uso de nombres como **django** (que entrará en conflicto con Django) o **test** (que entra en conflicto con un paquete Python incorporado).

```
mysite/
  manage.py
  mysite/
    __init__.py
    settings.py
    urls.py
    asgi.py
    wsgi.py
```

Figura 1: Árbol de la carpeta **mysite**

Enumeremos estos archivos:

1. El directorio externo **mysite/root** es un contenedor para su proyecto. Su nombre no le importa a Django ; puedes cambiarle el nombre a lo que quieras
2. **manage.py**: es una utilidad de línea de comandos que te permite interactuar con este proyecto de Django de varias maneras. Puede leer todos los detalles sobre *manage.py* en [django-admin y manage.py](#).
3. El directorio **mysite/** interno es el paquete real de Python para su proyecto. Su nombre es el nombre del paquete Python que necesitará usar para importar cualquier cosa dentro de él (*por ejemplo, mysite.urls*).
4. **mysite/\_\_init\_\_.py**: un archivo vacío que le dice a Python que este directorio debe considerarse un paquete de Python. Si eres un principiante de Python , [lee mas sobre estos paquetes en la documentacion oficial Python](#)
5. **mysite/settings.py**: configuración para este proyecto de Django . La configuración de Django le dirá todo sobre cómo funciona la configuración ([django settings](#)).
6. **mysite/urls.py**: las declaraciones de URL para este proyecto de Django ; una "tabla de contenido" de su sitio impulsado por Django. Puede leer más sobre las URL en el [despachador de URL](#).
7. **mysite/asgi.py**: Un punto de entrada para servidores web compatibles con ASGI para servir su proyecto. Consulte [Cómo implementar con ASGI](#) para obtener más detalles.
8. **mysite/wsgi.py**: Un punto de entrada para servidores web compatibles con WSGI para servir su proyecto. Consulte [Cómo implementar con WSGI](#) para obtener más detalles.

### 3.1.2. El servidor de desarrollo

Verifiquemos que su proyecto Django funciona. Cambie al directorio externo de **mysite**, si aún no lo ha hecho, y ejecute el siguiente comando:

```
$ python manage.py runserver
```

Luego, deberías ver sobre la línea de comandos, algo similar a lo que aparece en la siguiente imagen:

```
Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are
applied.
Run 'python manage.py migrate' to apply them.

February 27, 2020 - 15:50:53
Django version 3.0, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Figura 2: Inicio del servidor de django sobre la línea de comandos

Has iniciado el servidor de desarrollo Django , un servidor web ligero escrito exclusivamente en Python . Hemos incluido esto con Django para que pueda desarrollar las cosas rápidamente, sin tener que lidiar con la configuración de un servidor de producción, como Apache, hasta que esté listo para la producción.

Ahora es un buen momento para tener en cuenta que no debe usar este servidor en nada parecido a un entorno de producción. Está destinado solo para su uso durante el desarrollo.

Ahora que el servidor se está ejecutando, visite `http://127.0.0.1:8000/` con su navegador web. Verás un **¡Felicitaciones!** sobre la página y con un cohete despegando. Si es así entonces ¡Funcionó!.

## *Cambiando el puerto*

De manera predeterminada, el comando `runserver` inicia el servidor de desarrollo en la IP interna en el puerto 8000.

Si desea cambiar el puerto del servidor, páselo como un argumento de línea de comandos.

Por ejemplo, este comando inicia el servidor en el puerto 8080:

```
$ python manage.py runserver 8080
```

Si desea cambiar la IP del servidor, páselo junto con el puerto.

Por ejemplo, para escuchar todas las IP públicas disponibles

(lo cual es útil si está ejecutando Vagrant o desea mostrar su trabajo en otras computadoras en la red), use:

```
$ python manage.py runserver 0:8000
```

0 es un atajo para 0.0.0.0. Los documentos completos para el servidor de desarrollo se pueden encontrar en la referencia del servidor de ejecución.

## *Reinicio automático del servidor*

El servidor de desarrollo vuelve a cargar automáticamente el código [Python](#) para cada solicitud según sea necesario.

No necesita reiniciar el servidor para que los cambios de código surtan efecto.

Sin embargo, algunas acciones como agregar archivos no activan un reinicio, por lo que deberá reiniciar el servidor en estos casos.

### 3.1.3. Creando la aplicación de encuestas (polls)

Una vez que su entorno de proyecto está listo y configurado, estamos listos para empezar a trabajar.

Cada aplicación que escribe en [Django](#) consiste en un paquete de [Python](#) que sigue una determinada convención. [Django](#) viene con una utilidad que genera automáticamente la estructura básica de directorios de una aplicación, por lo que puede centrarse en escribir código en lugar de crear directorios.

## *Proyectos Vs Aplicaciones*

¿Cuál es la diferencia entre un proyecto y una aplicación?

Una aplicación es una aplicación web que hace algo, por ejemplo, un sistema de registro web, una base de datos de registros públicos o una pequeña aplicación de encuestas.

Un proyecto es una colección de configuraciones y aplicaciones para un sitio web en particular.

Un proyecto puede contener múltiples aplicaciones.

Una aplicación puede estar en múltiples proyectos.

Sus aplicaciones pueden vivir en cualquier lugar de su [ruta de Python](#). En este tutorial, crearemos nuestra aplicación de encuestas justo al lado de su archivo `manage.py` para que pueda importarse como su propio módulo de nivel superior, en lugar de un submódulo de `mysite`.

Para crear su aplicación, asegúrese de estar en el mismo directorio que `manage.py` y escriba este comando:

```
$ python manage.py startapp polls
```

Eso creará un directorio de encuestas, que se presenta así:

```
polls/
  __init__.py
  admin.py
  apps.py
  migrations/
    __init__.py
  models.py
  tests.py
  views.py
```

Figura 3: directorio de encuestas

Esta estructura de directorios albergará la aplicación de la encuesta.

#### 3.1.4. Escribiendo la primer vista (view)

Escribamos la primera vista. Abra el archivo **polls/views.py** y coloque el siguiente código de [Python](#) :

```
polls/views.py

from django.http import HttpResponseRedirect

def index(request):
    return HttpResponseRedirect("Hello, world. You're at the polls index.")
```

Figura 4: Código [Python](#) de la primer vista

Esta es la vista más simple posible en [Django](#) . Para llamar a la vista, necesitamos asignarla a una **URL**, y para esto necesitamos una **URLconf**.

Para crear una **URLconf** en el directorio de encuestas, cree un archivo llamado *urls.py*. Su directorio de aplicaciones ahora debería verse como en la *figura 5* Donde en el archivo *urls.py* ubicado en *polls* se incluirá el código quedando este como el código de la *figura 5*.

<pre>polls/   __init__.py   admin.py   apps.py   migrations/     __init__.py   models.py   tests.py   urls.py   views.py</pre>	<pre>polls/urls.py  from django.urls import path  from . import views  urlpatterns = [     path('', views.index, name='index'), ]</pre>
--	---

Figura 5: Modificación del directorio polls y agregado del código sobre el *urls.py*

El siguiente paso es apuntar la **URLconf** raíz al módulo *polls.urls*. En **mysite/urls.py**, agregue una importación para [django.urls.include](#) e inserte un [include\(\)](#) en la lista *urlpatterns*, Debe quedar como en la *figura 6*.

```

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

urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
]

```

Figura 6: Modificación del código sobre el `urls.py` del proyecto

La función `include()` permite hacer referencia a otros **URLconfs**. Cada vez que Django se encuentra con `include()`, corta cualquier parte de la **URL** que coincida con ese punto y envía la cadena restante a la **URLconf** incluida para su posterior procesamiento.

La idea detrás de `include()` es facilitar la conexión y reproducción de **URL**. Como las encuestas están en su propia **URLconf** (`polls/urls.py`), se pueden colocar debajo de `/polls/`, o debajo de `/fun_polls/`, o debajo de `/content/polls/`, o cualquier otra ruta raíz, y la aplicación seguirá funcionando.

### *Cuando usamos include()*

Siempre debe usar `include()` cuando incluya otros patrones de **URL**. `admin.site.urls` es la única excepción a esto.

Ahora ha conectado una vista de índice en la **URLconf**. Verifique que esté funcionando con el siguiente comando:

```
$ python manage.py runserver
```

Vaya a `http://localhost:8000/polls/` en su navegador, y debería ver el texto *"Hello, world. You're at the polls index"*, que definiste en la vista de índice.

### *Te aparece Page Not Found?*

Si recibe una página de error aquí, verifique que vaya a `http://localhost:8000/polls/` y no `http://localhost:8000/`.

La función `path()` recibe cuatro argumentos, dos obligatorios: ruta y vista, y dos opcionales: `kwargs` y nombre. En este punto, vale la pena revisar para qué sirven estos argumentos.

#### ■ *Argumento route del path():*

`route` es una cadena que contiene un patrón de **URL**. Al procesar una solicitud, Django comienza en el primer patrón en `urlpatterns` y avanza por la lista, comparando la **URL** solicitada con cada patrón hasta que encuentre uno que coincida.

Los patrones no buscan parámetros **GET** y **POST**, o el nombre de dominio. Por ejemplo, en una solicitud a `https://www.example.com/myapp/`, la **URLconf** buscará `myapp/`. En una solicitud a `https://www.example.com/myapp/?page=3`, la **URLconf** también buscará `myapp/`.

#### ■ *Argumento view del path():*

Cuando Django encuentra un patrón coincidente, llama a la función de vista especificada con un objeto `HttpRequest` como primer argumento y cualquier valor "capturado" de la ruta como argumentos de palabras clave. Vamos a dar un ejemplo de esto en un momento.



- **Argumento *kwargs* del *path()*:**

Los argumentos de palabras clave arbitrarias se pueden pasar en un diccionario a la vista de destino. No vamos a utilizar esta función de Django en el tutorial.

- **Argumento *name* del *path()*:**

Nombrar su **URL** le permite referirse a ella sin ambigüedades desde otros lugares de Django , especialmente desde las plantillas. Esta poderosa característica le permite realizar cambios globales en los patrones de **URL** de su proyecto mientras solo toca un solo archivo.

## 3.2. Escribiendo tu primera aplicación Django , parte 2

### 3.2.1. Configuración de la Base de Datos

Ahora, abra **mysite/settings.py**. Es un módulo Python normal con variables de nivel de módulo que representan la configuración de Django .

Por defecto, la configuración usa **SQLite**. Si eres nuevo en las bases de datos, o simplemente estás interesado en probar Django , esta es la opción más fácil. **SQLite** está incluido en Python , por lo que no necesitará instalar nada más para admitir su base de datos. Sin embargo, al comenzar su primer proyecto real, es posible que desee utilizar una base de datos más escalable como **PostgreSQL**, para evitar dolores de cabeza de cambio de base de datos en el futuro.

Si desea utilizar otra base de datos, instale los enlaces de base de datos apropiados ([database bindings](#)) y cambie las siguientes claves en el elemento **DATABASES** 'default' para que coincida con la configuración de conexión de su base de datos:

- **ENGINE** - Puede ser: `'django.db.backends.sqlite3'`, `'django.db.backends.postgresql'`, `'django.db.backends.mysql'`, or `'django.db.backends.oracle'`. Otros backends también están disponibles.
- **NAME** - El nombre de su base de datos. Si está utilizando **SQLite**, la base de datos será un archivo en su computadora; en ese caso, **NAME** debe ser la ruta absoluta completa, incluido el nombre de archivo, de ese archivo. El valor predeterminado, `os.path.join(BASE_DIR, 'db.sqlite3')`, almacenará el archivo en el directorio de su proyecto.

Si no está utilizando **SQLite** como su base de datos, se deben agregar configuraciones adicionales como **USUARIO** , **CONTRASEÑA** y **HOST** . Para obtener más detalles, consulte la documentación de referencia para **BASES DE DATOS**.

### *Para bases de datos que no sean SQLite*

Si está utilizando una base de datos además de **SQLite**, asegúrese de haber creado una base de datos en este momento. Haga eso con `"CREATE DATABASE database_name;"` dentro de la solicitud interactiva de su base de datos. También asegúrese de que el usuario de la base de datos proporcionado en **mysite/settings.py** tenga privilegios de crear base de datos". Esto permite la creación automática de una base de datos de prueba que será necesaria en un tutorial posterior.

Si está utilizando **SQLite**, no necesita crear nada de antemano; el archivo de la base de datos se creará automáticamente cuando sea necesario.

Mientras editas **mysite/settings.py**, configura **TIME\_ZONE** en su zona horaria.

Además, tenga en cuenta la configuración **INSTALLED\_APPS** en la parte superior del archivo. Esta contiene los nombres de todas las aplicaciones de Django que se activan en esta instancia de Django . Las aplicaciones se pueden usar en múltiples proyectos, y puede empaquetarlas y distribuirlas para que otras personas las usen en sus proyectos.

Por defecto, **INSTALLED\_APPS** contiene las siguientes aplicaciones, las cuales se instalan al instalar Django

- `django.contrib.admin` – Sitio de administración.
- `django.contrib.auth` – Sistema de autenticación.
- `django.contrib.contenttypes` – Un marco para los tipos de contenido.

- [django.contrib.sessions](#) – framework (marco) de sesiones.
- [django.contrib.messages](#) – framework (marco) de mensajes.
- [django.contrib.staticfiles](#) – Un marco para administrar archivos estáticos.

Estas aplicaciones se incluyen por defecto como una conveniencia para casos específicos.

Sin embargo, algunas de estas aplicaciones utilizan al menos una tabla de base de datos, por lo que debemos crear las tablas en la base de datos antes de poder usarlas. Para hacer eso, ejecuta el siguiente comando:

```
$ python manage.py migrate
```

El comando migrate analiza la configuración [INSTALLED\\_APPS](#) y crea las tablas de base de datos necesarias de acuerdo con la configuración de la base de datos en su archivo **mysite/settings.py** y las migraciones de la base de datos que se envían con la aplicación (las cubriremos más adelante). Verá un mensaje para cada migración que aplique. Si está interesado, ejecute el cliente de línea de comandos para su base de datos y escriba \dt (PostgreSQL), MOSTRAR TABLAS; (MariaDB, MySQL), .schema (SQLite) o SELECT TABLE\_NAME FROM USER\_TABLES; (Oracle) para mostrar las tablas que Django creó.

### *Para los minimalistas*

Como dijimos anteriormente, las aplicaciones predeterminadas se incluyen para el caso común, pero no todos las necesitan. Si no necesita ninguno o todos ellos, no dude en comentar o eliminar las líneas apropiadas de [INSTALLED\\_APPS](#) antes de ejecutar la [migración](#).

El comando [migrate](#) solo ejecutará migraciones para aplicaciones en [INSTALLED\\_APPS](#).

#### 3.2.2. Creando un Modelo

### *Filosofía*

Un modelo es la fuente única y definitiva de verdad sobre sus datos. Contiene los campos y comportamientos esenciales de los datos que está almacenando.

[Django](#) sigue el principio DRY ([DRY principle](#)).

El objetivo es definir su modelo de datos en un lugar y derivar automáticamente cosas de él.

Esto incluye las migraciones, a diferencia de **Ruby On Rails**, por ejemplo, las migraciones se derivan completamente de su archivo de modelos, y son esencialmente un historial que [Django](#) puede recorrer para actualizar su esquema de base de datos para que coincida con sus modelos actuales.

En nuestra aplicación de encuestas, crearemos dos modelos: Pregunta (Question) y Elección (Choice).

```
polls/models.py

from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
```

Figura 7: Código de los modelos **Question** y **Choice**

Una `Question` tiene una pregunta y una fecha de publicación. Una opción tiene dos campos: el texto de la opción y un conteo de votos. Cada opción está asociada con una pregunta.

Estos conceptos están representados por las clases de `Python`. Edite el archivo `polls/models.py` para que se vea como en la *figura 7*.

Aquí, cada modelo está representado por una clase que subclasifica `django.db.models.Model`. Cada modelo tiene una serie de variables de clase, cada una de las cuales representa un campo de base de datos en el modelo.

Cada campo está representado por una instancia de clase `Campo`, por ejemplo, `CharField` para campos de caracteres y `DateTimeField` para fechas y horas. Esto le dice a `Django` qué tipo de datos contiene cada campo.

El nombre de cada instancia de campo (por ejemplo, `question_text` o `pub_date`) es el nombre del campo, en formato amigable para la máquina. Utilizará este valor en su código de `Python`, y su base de datos lo usará como el nombre de la columna.

Puede usar un primer argumento posicional opcional del campo para designar un nombre que sea legible por humanos. Eso se usa en un par de partes introspectivas de `Django`, y también sirve como documentación. Si no se proporciona este campo, `Django` usará el nombre legible por la máquina. En este ejemplo, solo hemos definido un nombre legible para humanos para `Question.pub_date`. Para todos los demás campos en este modelo, el nombre del campo legible por la máquina será suficiente como su nombre legible por humanos.

Algunas clases de campo tienen argumentos requeridos. `CharField`, por ejemplo, requiere que le des una **longitud máxima** (`max_length`). Eso se usa no solo en el esquema de la base de datos, sino también en la validación.

Un campo también puede tener varios argumentos opcionales; en este caso, hemos establecido el valor predeterminado (`default`) de votos en 0.

Finalmente, tenga en cuenta que se define una relación, usando `ForeignKey`. Eso le dice a `Django` que cada Elección está relacionada con una sola Pregunta. `Django` admite todas las relaciones de base de datos comunes: muchas a una, muchas a muchas y una a una.

### 3.2.3. Modelos de Activación

Ese pequeño fragmento de código del modelo le da a `Django` bastante información. Con él, `Django` es capaz de:

- Crear un esquema de base de datos (sentencias `CREATE TABLE`) para esta aplicación.
- Crear una **API** de acceso a la base de datos de `Python` para acceder a los objetos `Question` y `Choice`.

Pero primero debemos decirle a nuestro proyecto *que la aplicación de encuestas está instalada*.

## Filosofía

Las aplicaciones de `Django` son "conectables": puede usar una aplicación en varios proyectos y puede distribuir aplicaciones, ya que no tienen que estar vinculadas a una determinada instalación de `Django`.

Para incluir la aplicación en nuestro proyecto, necesitamos agregar una referencia a su clase de configuración en la configuración `INSTALLED_APPS`. La clase `PollsConfig` está en el archivo `polls/apps.py`, por lo que su ruta punteada es `'polls.apps.PollsConfig'`. Hay que editar el archivo `mysite/settings.py` y agregar esa ruta punteada a la configuración `INSTALLED_APPS`. Se verá así:

```
mysite/settings.py

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

Figura 8: Instalación de la aplicación `polls` en el `settings.py` del proyecto

Listo, **Django** con esto ya sabe incluir la aplicación de encuestas. Ahora, debemos aplicar migraciones, lo cual se hace con el comando:

```
$ python manage.py makemigrations polls
```

Luego, en la terminal vas a ver algo similar a:

```
Migrations for 'polls':
polls/migrations/0001_initial.py:
- Create model Choice
- Create model Question
- Add field question to choice
```

Figura 9: Terminal con un makemigrations satisfactorio

Al correr *makemigrations*, le estás diciendo a Django que has realizado algunos cambios en tus modelos (En este caso, agregaste modelos al **models.py**) y que queremos que esos cambios sean almacenados como migraciones.

Las migraciones es la forma en la cual **Django** almacena los cambios en sus modelos (y, por lo tanto, en el esquema de su base de datos): estos son archivos en el disco. Podemos leer las migraciones si queremos; en este caso son las encuestas de **polls/migrations/0001\_initial.py**. No se preocupe, no se espera que los lea cada vez que Django hace uno, pero están diseñados para ser editados por humanos en caso de que desee modificar manualmente cómo **Django** cambia las cosas.

Hay un comando que ejecutará las migraciones por usted y administrará el esquema de su base de datos automáticamente, este es el comando *migrate*, y lo veremos en un momento, pero primero, veamos qué **SQL** ejecutará esa migración. El comando *sqlmigrate* toma nombres de migración y devuelve su SQL:

```
$ python manage.py sqlmigrate polls 0001
```

Vas a ver algo similar a esto, quizás en otro formato pero con el mismo contenido:

```
BEGIN;
--
-- Create model Choice
--
CREATE TABLE "polls_choice" (
    "id" serial NOT NULL PRIMARY KEY,
    "choice_text" varchar(200) NOT NULL,
    "votes" integer NOT NULL
);
--
-- Create model Question
--
CREATE TABLE "polls_question" (
    "id" serial NOT NULL PRIMARY KEY,
    "question_text" varchar(200) NOT NULL,
    "pub_date" timestamp with time zone NOT NULL
);
--
-- Add field question to choice
--
ALTER TABLE "polls_choice" ADD COLUMN "question_id" integer NOT NULL;
ALTER TABLE "polls_choice" ALTER COLUMN "question_id" DROP DEFAULT;
CREATE INDEX "polls_choice_7aa0f6ee" ON "polls_choice" ("question_id");
ALTER TABLE "polls_choice"
    ADD CONSTRAINT
    "polls_choice_question_id_246c99a640fbbd72_fk_polls_question_id"
    FOREIGN KEY ("question_id")
    REFERENCES "polls_question" ("id")
    DEFERRABLE INITIALLY DEFERRED;
COMMIT;
```

Figura 10: python manage.py sqlmigrate polls 0001

Notemos que:

- El resultado exacto variará según la base de datos que esté utilizando. El ejemplo anterior se genera para PostgreSQL.
- Los nombres de las tablas se generan automáticamente combinando el nombre de la aplicación (encuestas) y el nombre en minúsculas del modelo: pregunta y elección. (Puede anular este comportamiento).
- Las claves primarias (ID) se agregan automáticamente. (También puede anular esto).
- Por convención, Django agrega `_id` al nombre del campo de clave externa. (Sí, también puedes sobrescribirlo).
- La relación de clave externa se hace explícita por una restricción FOREIGN KEY. No se preocupe por las partes DEFERRABLES; esto le dice a PostgreSQL que no aplique la clave foránea hasta el final de la transacción
- Se adapta a la base de datos que está utilizando, por lo que los tipos de campo específicos de la base de datos como `auto_increment` (MySQL), `serial` (PostgreSQL) o `autoincrement` de clave primaria entera (SQLite) se manejan automáticamente. Lo mismo ocurre con las citas de los nombres de campo, por ejemplo, con comillas dobles o comillas simples.
- El comando `sqlmigrate` no ejecuta realmente la migración en su base de datos, sino que lo imprime en la pantalla para que pueda ver lo que **SQL Django** cree que es necesario. Es útil para verificar qué hará Django o si tiene administradores de bases de datos que requieren *scripts SQL* para dichos cambios.

Si está interesado, también puede ejecutar `python manage.py check`; Esto verifica si hay algún problema en su proyecto sin hacer migraciones o tocar la base de datos.

Ahora, debemos correr el comando `migrate` otra vez para crear estos modelos en las tablas de tu base de datos.

```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Rendering model states... DONE
  Applying polls.0001_initial... OK
```

Figura 11: `python manage.py sqlmigrate polls 0001`

El comando `migrate` toma todas las migraciones que no se han aplicado (Django rastrea cuáles se aplican usando una tabla especial en su base de datos llamada `django_migrations`) y las ejecuta en su base de datos, esencialmente, sincronizando los cambios que realizó en sus modelos con el esquema en la base de datos.

Las migraciones son muy potentes y le permiten cambiar sus modelos con el tiempo, a medida que desarrolla su proyecto, sin la necesidad de eliminar su base de datos o tablas y crear nuevas: se especializa en actualizar su base de datos en vivo, sin perder datos. Los cubriremos con más profundidad en una parte posterior del tutorial, pero por ahora, recuerde la guía de tres pasos para realizar cambios en el modelo:

- Creamos/modificamos los modelos en el archivo `models.py`.
- Luego corremos el comando `python manage.py makemigrations` para crear las migraciones para esos cambios.
- Corremos el comando `python manage.py migrate` para aplicar los cambios realizados en la base de datos.

La razón por la que existen comandos separados para realizar y aplicar migraciones es porque comprometerás las migraciones en tu sistema de control de versiones y las enviarás con tu aplicación; no solo facilitan su desarrollo, también pueden ser utilizados por otros desarrolladores y en producción.

### 3.2.4. Jugando con la API

Ahora, entremos al *shell* interactivo de [Python](#) y juguemos con la **API** gratuita que [Django](#) le brinda. Para invocar el *shell* de [Python](#), use este comando:

```
$ python manage.py shell
```

Estamos usando esto en lugar de simplemente escribir *"python"*, porque *manage.py* establece la variable de entorno `DJANGO_SETTINGS_MODULE`, que le da a [Django](#) la ruta de importación de [Python](#) a su archivo `mysite/settings.py`.

Una vez que entramos en la *shell* podemos trabajar con ella realizando [consultas](#) como se indica en la figura:

```
>>> from polls.models import Choice, Question # Import the model classes we just
wrote.

# No questions are in the system yet.
>>> Question.objects.all()
<QuerySet []>

# Create a new Question.
# Support for time zones is enabled in the default settings file, so
# Django expects a datetime with tzinfo for pub_date. Use timezone.now()
# instead of datetime.datetime.now() and it will do the right thing.
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())

# Save the object into the database. You have to call save() explicitly.
>>> q.save()

# Now it has an ID.
>>> q.id
1

# Access model field values via Python attributes.
>>> q.question_text
"What's new?"
>>> q.pub_date
datetime.datetime(2012, 2, 26, 13, 0, 0, 775217, tzinfo=<UTC>)

# Change values by changing the attributes, then calling save().
>>> q.question_text = "What's up?"
>>> q.save()

# objects.all() displays all the questions in the database.
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
```

Getting Help

Language: en

Documentation version: 3.0

Figura 12: Making queries

Ahora, vemos una cosa `<Question: Question object (1)>` no es una representación útil de este objeto. Arreglemos eso editando el modelo `Question` (en el archivo `polls/models.py`) y agregando un método `__str__()` a los modelos `Question` y `Choice`:

```
polls/models.py

from django.db import models

class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text

class Choice(models.Model):
    # ...
    def __str__(self):
        return self.choice_text
```

Figura 13: ●

Es importante agregar los métodos `__str__()` a sus modelos, no solo para su propia conveniencia al tratar con el mensaje interactivo, sino también porque las representaciones de los objetos se utilizan en todo el administrador generado automáticamente por Django .

Agreguemos también un método personalizado a este modelo:

```
polls/models.py

import datetime

from django.db import models
from django.utils import timezone

class Question(models.Model):
    # ...
    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
```

Figura 14: ●

Observe la adición de `import datetime` y de `django.utils import timezone`, para hacer referencia al módulo de fecha y hora estándar de Python y las utilidades relacionadas con la zona horaria de Django en `django.utils.timezone`, respectivamente. Si no está familiarizado con el manejo de zona horaria en Python , puede obtener más información en los documentos de soporte de zona horaria.

Guarde estos cambios e inicie un nuevo *shell* interactivo de Python ejecutando nuevamente `python manage.py shell`:

```
>>> from polls.models import Choice, Question

# Make sure our __str__() addition worked.
>>> Question.objects.all()
<QuerySet [<Question: What's up?>>]

# Django provides a rich database lookup API that's entirely driven by
# keyword arguments.
>>> Question.objects.filter(id=1)
<QuerySet [<Question: What's up?>>]
>>> Question.objects.filter(question_text__startswith='What')
<QuerySet [<Question: What's up?>>]
```



```

# Get the question that was published this year.
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
<Question: What's up?>

# Request an ID that doesn't exist, this will raise an exception.
>>> Question.objects.get(id=2)
Traceback (most recent call last):
...
DoesNotExist: Question matching query does not exist.

# Lookup by a primary key is the most common case, so Django provides a
# shortcut for primary-key exact lookups.
# The following is identical to Question.objects.get(id=1).
>>> Question.objects.get(pk=1)
<Question: What's up?>

# Make sure our custom method worked.
>>> q = Question.objects.get(pk=1)
>>> q.was_published_recently()
True

# Give the Question a couple of Choices. The create call constructs a new
# Choice object, does the INSERT statement, adds the choice to the set
# of available choices and returns the new Choice object. Django creates
# a set to hold the "other side" of a ForeignKey relation
# (e.g. a question's choice) which can be accessed via the API.
>>> q = Question.objects.get(pk=1)

# Display any choices from the related object set -- none so far.
>>> q.choice_set.all()
<QuerySet []>

# Create three choices.
>>> q.choice_set.create(choice_text='Not much', votes=0)
<Choice: Not much>
>>> q.choice_set.create(choice_text='The sky', votes=0)
<Choice: The sky>
>>> c = q.choice_set.create(choice_text='Just hacking again', votes=0)

# Choice objects have API access to their related Question objects.
>>> c.question
<Question: What's up?>

# And vice versa: Question objects get access to Choice objects.
>>> q.choice_set.all()
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>
>>> q.choice_set.count()
3

# The API automatically follows relationships as far as you need.
# Use double underscores to separate relationships.
# This works as many levels deep as you want; there's no limit.
# Find all Choices for any question whose pub_date is in this year
# (reusing the 'current_year' variable we created above).
>>> Choice.objects.filter(question__pub_date__year=current_year)
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>

# Let's delete one of the choices. Use delete() for that.
>>> c = q.choice_set.filter(choice_text__startswith='Just hacking')
>>> c.delete()

```



Para obtener más información sobre las relaciones del modelo, consulte Acceso a objetos relacionados ([Accessing related objects](#)). Para obtener más información sobre cómo utilizar guiones bajos dobles para realizar búsquedas de campo a través de la API, consulte las búsquedas de campo ([field lookups](#)). Para obtener detalles completos sobre la API de base de datos, consulte nuestra referencia de API de base de datos ([Database API reference](#)).

### 3.2.5. Introducción al administrador de Django

#### *Filosofía*

Generar sitios de administración para que su personal o clientes agreguen, cambien y eliminen contenido es un trabajo tedioso que no requiere mucha creatividad. Por esa razón, Django automatiza por completo la creación de interfaces de administración para modelos.

Django fue escrito en un entorno de redacción, con una separación muy clara entre los “editores de contenido” y el sitio “público”.

Los administradores del sitio usan el sistema para agregar noticias, eventos, resultados deportivos, etc., y ese contenido se muestra en el sitio público.

Django resuelve el problema de crear una interfaz unificada para que los administradores del sitio editen contenido.

El administrador no está destinado a ser utilizado por los visitantes del sitio.

Es para los administradores del sitio.

#### Creando un usuario administrador

Crear un usuario administrador Primero, necesitaremos crear un usuario que pueda iniciar sesión en el sitio de administración. Ejecute el siguiente comando:

```
$ python manage.py createsuperuser
```

Al correr este comando nos pedirá una serie de datos, como el nombre del *usuario*, *nuestro mail* y una *contraseña*.

#### Corremos el servidor de desarrollo

El administrador de Django ya viene activado por defecto. Primero debemos correr el servidor una vez creado el administrador:

```
$ python manage.py runserver
```

Ahora, abra un navegador web y vaya a / **admin** / en su dominio local, por ejemplo, **http://127.0.0.1:8000/admin/**. Debería ver la pantalla de inicio de sesión del administrador:

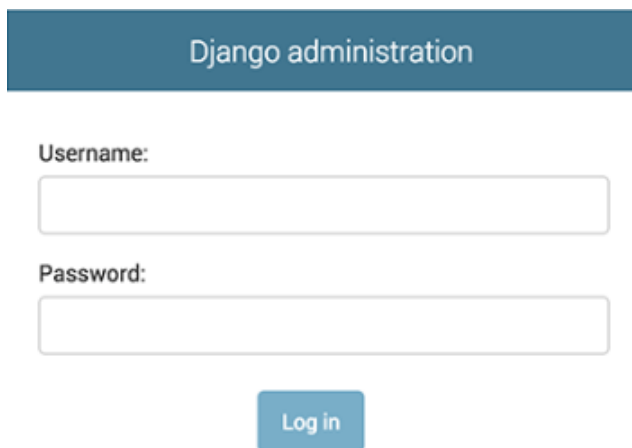


Figura 15: Login del administrador

Dado que la traducción ([translation](#)) está activada de manera predeterminada, la pantalla de inicio de sesión puede mostrarse en su propio idioma, según la configuración de su navegador y si [Django](#) tiene una traducción para este idioma.

## El sitio de administración

Ahora, intente iniciar sesión con la cuenta de *superusuario* que creó en el paso anterior. Debería ver la página de índice de administración de [Django](#):

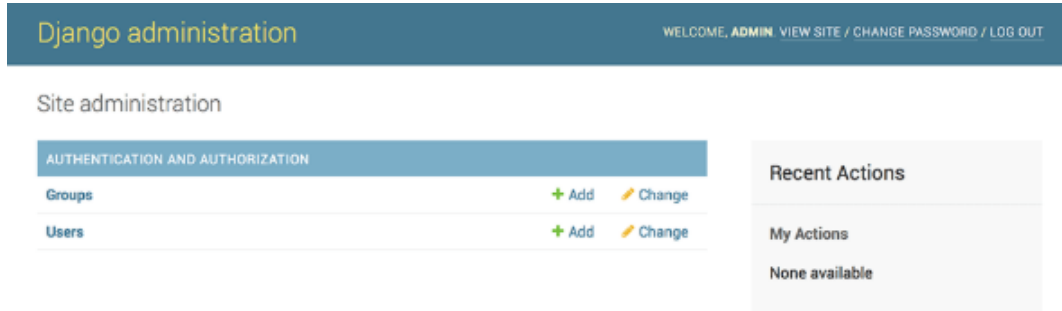


Figura 16: Login del administrador

Debería ver algunos tipos de contenido editable: grupos y usuarios. Estos los proporciona [django.contrib.auth](#), el cual es el *marco de autenticación* enviado por [Django](#).

## Hacer que la aplicación de la aplicacion sea modificable en el administrador

¿Pero dónde está nuestra aplicación de encuestas? No se muestra en la página de índice de administrador.

Solo una cosa más que hacer: necesitamos decirle al administrador que los objetos de Pregunta tienen una interfaz de administrador. Para hacer esto, abra el archivo `polls/admin.py` y editelo para que se vea así:

```
polls/admin.py

from django.contrib import admin

from .models import Question

admin.site.register(Question)
```

Figura 17: Login del administrador

## Explore la funcionalidad de administración gratuita

Ahora que hemos registrado el modelo **Question**, [Django](#) sabe que debería mostrarse en la página de índice de administración:



Figura 18: Login del administrador

Haga clic en “**Question**”. Ahora está en la página de “lista de cambios” para Question. Esta página muestra todas las preguntas en la base de datos y le permite elegir una para cambiarla. Ahí está el “¿Qué pasa?” pregunta que creamos anteriormente:

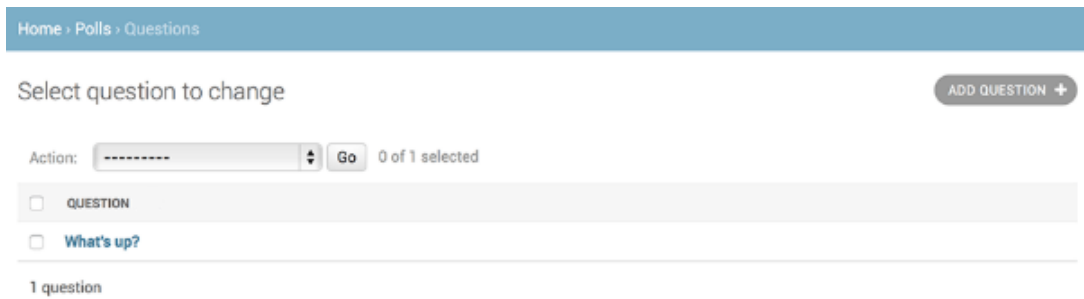


Figura 19: Login del administrador

Haz clic en “¿Qué pasa?” pregunta para editarlo:

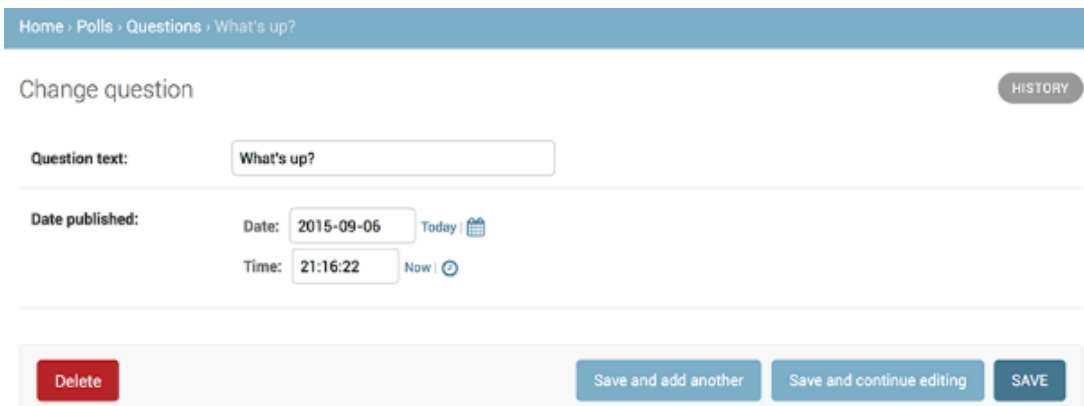


Figura 20: Modificando la Question

Acá debemos tener en cuenta que:

- El formulario se genera automáticamente a partir del modelo de Pregunta.
- Los diferentes tipos de campo de modelo (`DateTimeField`, `CharField`) corresponden al *widget* de entrada **HTML** apropiado. Cada tipo de campo sabe cómo mostrarse en el administrador de Django.
- Cada `DateTimeField` obtiene accesos directos de **JavaScript** gratuitos. Las fechas obtienen un acceso directo “Hoy” una ventana emergente de calendario, y las horas obtienen un acceso directo “Ahora” una ventana emergente conveniente que enumera las horas comúnmente ingresadas.

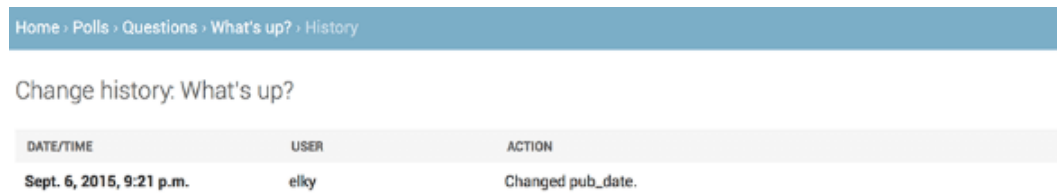
La parte inferior de la página le ofrece un par de opciones:

- Guardar: guarda los cambios y vuelve a la página de lista de cambios para este tipo de objeto.
- Guardar y continuar editando: guarda los cambios y vuelve a cargar la página de administración para este objeto.
- Guardar y agregar otro: guarda los cambios y carga un nuevo formulario en blanco para este tipo de objeto
- Eliminar: muestra una página de confirmación de eliminación

Si el valor de “Fecha de publicación” no coincide con la hora en que creó la pregunta en el Tutorial 1, probablemente significa que olvidó establecer el valor correcto para la configuración `TIME_ZONE`. Cámbielo, vuelva a cargar la página y verifique que aparezca el valor correcto.

Cambie la “Fecha de publicación” haciendo clic en los accesos directos “Hoy” “Ahora”. Luego haga clic en “Guardar y continuar editando”. Luego haga clic en “Historial” en la esquina superior derecha. Verá una página que enumera

todos los cambios realizados en este objeto a través del administrador de Django, con la marca de tiempo y el nombre de usuario de la persona que realizó el cambio:



Home > Polls > Questions > What's up? > History		
Change history: What's up?		
DATE/TIME	USER	ACTION
Sept. 6, 2015, 9:21 p.m.	elky	Changed pub_date.

Figura 21: Modificando la Question

Cuando se sienta cómodo con la API de modelos y se haya familiarizado con el sitio de administración, lea la [parte 3](#) de este tutorial para obtener información sobre cómo agregar más vistas a nuestra aplicación de encuestas.

## 4. Referencias

### 4.1. Refes de las partes de los tutoriales

- [Parte 1](#)
- [Parte 2](#)
- [parte 3](#)

### 4.2. Refes W3School

- [Html](#)
- [css](#)
- [javascript](#)
- [Python](#)
- [Bootstrap](#)

### 4.3. Atajos

- [LANGUAGE\\_CODE](#)
- [TIME\\_ZONE](#)