

Uso de Python + Django Para el Desarrollo Web

Ezequiel Remus

15 de mayo de 2019



Índice

1. Introducción	3
1.1. ¿Qué es Django?	3
1.2. Características Principales	3
2. Que es el Model-Template-View	4
2.1. Archivos Predeterminados Del Proyecto	5
2.1.1. Archivo __init.py__	5
2.1.2. Archivo manage.py	6
2.1.3. Archivo settings.py	6
2.1.4. Archivo urls.py	7
2.2. Archivos Predeterminados De La Aplicación	7
2.2.1. __init__.py	8
2.2.2. models.py	8
2.2.3. views.py	8
2.2.4. test.py	8
3. Desarrollo De Un Proyecto + APP	9
3.1. Crear Proyecto	9
3.2. Creación de Apps y Estructura de Proyecto	13
3.3. Creación de Modelos	15
3.4. Configuración de URLs y views	16
3.5. Sistemas de Plantillas	19
4. Apéndices	22
4.1. Comandos Basicos Windows	22
4.2. Comandos Python (pip)	23

Resumen

El siguiente documento trata el desarrollo básico de un proyecto web bajo el empleo de Python y Django. Este proyecto web corresponde a un proyecto que se ira subiendo a un repositorio Github el cual se puede encontrar en las referencias. Este repositorio estará en constante construcción donde podrán reconocerse las actualizaciones realizadas a través de los commits. Este tendrá tanto código Python + Django como posibles actualizaciones de este documento pdf, el cual también se ira actualizando con el tiempo.

1. Introducción

1.1. ¿Qué es Django?

Django es un framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como Modelo–vista–template (M.V.T) que es una redefinición del modelo M.V.C.(Modelo-Vista.Controlador).

Fue desarrollado en origen para gestionar varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberada al público bajo una licencia BSD en julio de 2005; el framework fue nombrado en alusión al guitarrista de jazz gitano Django Reinhardt. En junio de 2008 fue anunciado que la recién formada Django Software Foundation se haría cargo de Django en el futuro.

La meta fundamental de Django es facilitar la creación de sitios web complejos. Django pone énfasis en el re-uso, la conectividad y extensibilidad de componentes, el desarrollo rápido y el principio No te repitas (DRY, del inglés Don't Repeat Yourself). Python es usado en todas las partes del framework, incluso en configuraciones, archivos, y en los modelos de datos.

1.2. Características Principales

Los desarrolladores de Django lo definen así: *“Django es un framework web de alto nivel que fomenta el desarrollo rápido y el diseño limpio y pragmático”*

Características:

1. **Python:** “Django es un framework web de alto nivel.”eso esta muy claro en la definición, pero ahora agregaremos algo, “Django es un framework web de alto nivel escrito en Python”. Gracias a esto Django hereda todas las características y facilidades que nos da Python, entre ellas escribir código bastante fácil de entender, y sobre todo te permite desarrollar aplicaciones muy rápidas y potentes.
2. **Rapidez:** Django nació en un ambiente periodístico, donde se subian noticias muy rápido, y como los desarrolladores no pudieron estar a ese ritmo decidieron crear algo que sí lo haga, y así fue como nace Django , es por eso que ha sido estructurado de tal manera que tus aplicaciones web se crean muy rápidas.
3. **DRY:** No te repitas!, django utiliza esta filosofía para no crear bloques de código iguales y fomentar la reutilización del mismo.
4. **Admin:** Django es el único framework que “por defecto” viene con un sistema de administración activo, listo para ser utilizado sin ningún tipo de configuración.
5. **ORM:** Para resumir esto, tómalo como una herramienta que te permite realizar consultas SQL a la Base de Datos, SIN UTILIZAR SQL.

2. Que es el Model-Template-View

El patrón **M.V.T**, nace gracias a que los desarrolladores no tuvieron la intención de seguir algún patrón de desarrollo, sino hacer el framework lo más funcional posible.

Para poder entender este patrón debemos realizar primero una analogía utilizando el patrón M.V.C

- El *modelo* en Django sigue siendo **modelo**
- La *vista* en Django se llama **Plantilla** (Template)
- El *controlador* en Django se llama **Vista**

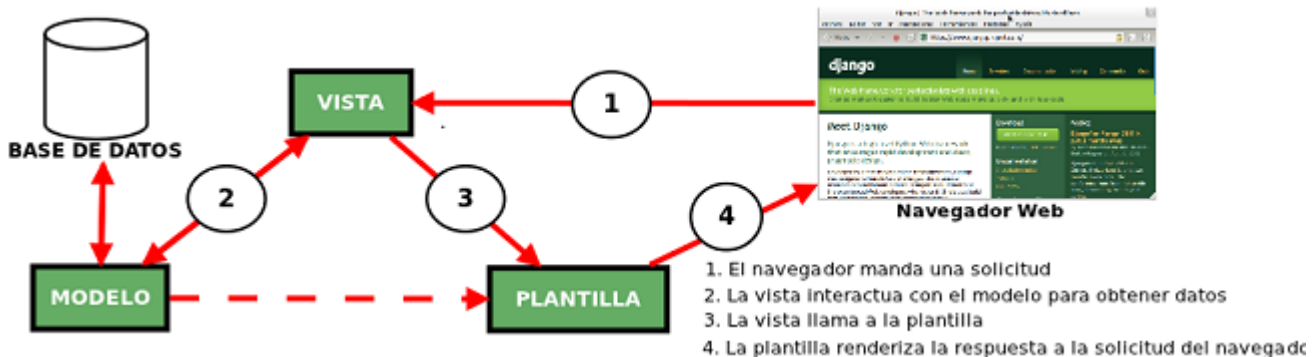


Figura 1: Imagen que visualiza el comportamiento del patrón

Ahora, la pregunta es ¿Qué hace cada uno?:

1. **Modelo:** El modelo define los datos almacenados. Este se encuentra en forma de clases de Python. Cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros (También posee métodos). Todo esto permite indicar y controlar el comportamiento de los datos. Es decir, el modelo es el que manipula o interactúa con la base de datos.
2. **Vista:** La vista se presenta en forma de funciones en Python, su propósito es determinar que datos serán visualizados, entre otras cosas más que iremos viendo conforme avanzamos con el curso. El **ORM** de Django permite escribir código Python en lugar de SQL para hacer las consultas que necesita la vista. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios. Lo más importante a entender con respecto a la vista es que no tiene nada que ver con el estilo de presentación de los datos, sólo se encarga de los datos, la presentación es tarea de la plantilla.
3. **Plantilla(Template):** La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en si no solamente crea contenido en HTML (también XML, CSS, Javascript, CSV, etc).

Por ahora nos enfocaremos a lo básico el HTML. El template recibe los datos de la vista y luego los organiza para la presentación al navegador web. Las etiquetas que Django usa para las plantillas permiten que sea flexible para los diseñadores del frontend, incluso tiene estructuras de datos como if, por si es necesaria una presentación lógica de los datos, estas estructuras son limitadas para evitar un desorden poniendo cualquier tipo de código Python.

Esto permite que la lógica del sistema siga permaneciendo en la vista.

Por otro lado, podemos realizar otro diagrama al implementar un manejo de rutas mediante URLs:



Figura 2: Imagen que visualiza el comportamiento del patrón implementando el mapeo de URLs

Django posee un mapeo de URLs que permite controlar el despliegue de las vistas, esta configuración es conocida como URLConf. El trabajo del URLConf es leer la URL que el usuario solicitó, encontrar la vista apropiada para la solicitud y pasar cualquier variable que la vista necesite para completar su trabajo. El URLConf está construido con expresiones regulares en Python y sigue la filosofía de Python: Explícito es mejor que implícito. Este URLConf permite que las rutas que maneje Django sean agradables y entendibles para el usuario.

2.1. Archivos Predeterminados Del Proyecto

Veremos que al iniciar el proyecto se crean una serie de archivos, los cuales se crean de manera pre-determinada. Es muy importante antes de empezar a hacer nada de código tener bien en claro como están organizados los archivos en un proyecto django y entender cada uno de estos, ya que si mantenemos desorganizados estos archivos se desatará un caos el cual nos complicará la existencia. Vamos a darles un primer vistazo antes de empezar con el desarrollo de un proyecto.

2.1.1. Archivo `__init__.py`

El archivo `__init__.py` es un archivo vacío que le dice a Python que debe considerar este directorio como un paquete de Python. Es decir, el archivo `__init__.py` es utilizado para inicializar paquetes de Python. Imaginemos que tenemos un paquete de python llamado `package`, que además contiene un subpaquete llamado `subpackage`. Esto se podría tener esta pinta:

```
ezequiel@ezequiel-VirtualBox:~/package$ cd ..
ezequiel@ezequiel-VirtualBox:~$ tree package
package
├── archivo1.py
├── archivo2.py
├── archivo3.py
├── __init__.py
└── subpackage
    ├── __init__.py
    └── submodulo.py
```

Figura 3: Empleo de `__init__.py`

En este caso, el archivo `__init__.py` le indica al intérprete de Python que el directorio `package` contiene un módulo, y que debe tratarlo como tal (es decir, hacer que sea posible importar los archivos como parte del módulo).

En general no es necesario poner nada en el archivo `__init__.py`, pero es muy común usarlo para realizar configuraciones e importar cualquier objeto necesario de nuestra librería.

Por ejemplo, en nuestro ejemplo, si el archivo `archivo1.py` contiene una clase llamada `Archivo`, podemos importarla con `__init__.py` para que esté disponible al nivel de paquete. Normalmente para importar esta clase, tendríamos que hacer lo siguiente:

```
from package.archivo1 import Archivo
```

Pero podemos simplificar esto mediante el uso del archivo `__init__.py`:

```
# En el archivo package/__init__.py
from archivo1 import Archivo

# En tu programa que utiliza el paquete package
from package import Archivo
```

Finalmente, para mas información sobre este archivo dejo la [documentación oficial](#) para que puedan darle una ojeada un poco mas profunda.

2.1.2. Archivo `manage.py`

Este archivo contiene una porción de código que permite interactuar con el proyecto de Django de muchas formas. Esta es una utilidad de linea de comandos que te permite interactuar con el proyecto de diversas formas.

Tengamos en cuenta esto, **django-admin** es la unidad de comandos para tareas administrativas. Al crear un proyecto, se crea automaticamente el archivo `manage.py`. En este se crea la variable de entorno **DJANGO_SETTINGS_MODULE** que es la herramienta para decirle a Django como son tus configuraciones, esta variable se encuentra en el archivo `settings.py`.

2.1.3. Archivo `settings.py`

Este archivo contiene todas las configuraciones para el proyecto. Sin duda, uno de los archivos más importantes de todo el proyecto, toda la configuración que usará Django en nuestro proyecto estará dentro de este archivo, por lo que se debe tener mucho cuidado al momento de manipularlo. Por defecto Django te trae solo un archivo `settings.py`, veamos cual es el contenido de este archivo.

Pasemos a explicar las configuraciones más importantes que trae ese archivo.

1. **BASE_DIR** Esta es la variable que tiene por contenido la ruta a una determinada zona de tu proyecto, por defecto lleva la ruta al archivo `settings.py`, si te queda la curiosidad de ver su contenido, puedes imprimirla (`print BASE_DIR`).
2. **SECRET_KEY** Todo proyecto en Django tiene esta variable que lleva una clave secreta, el contenido de esta variable siempre debe estar protegida y nunca escrita como texto plano dentro del archivo, por defecto Django la pone así pero ya es trabajo del desarrollador protegerla.
3. **DEBUG** Variable booleana solo acepta `True` o `False`. `True` cuando tu proyecto está en modo de depuración, esto lo notas fácilmente cuando te sale algún error y ves la pantalla amarilla de Django donde te muestra el detalle del error obtenido. `False` cuando el proyecto ya no se encuentra en modo de depuración, esto se suele utilizar cuando el proyecto ya se encuentra en producción.

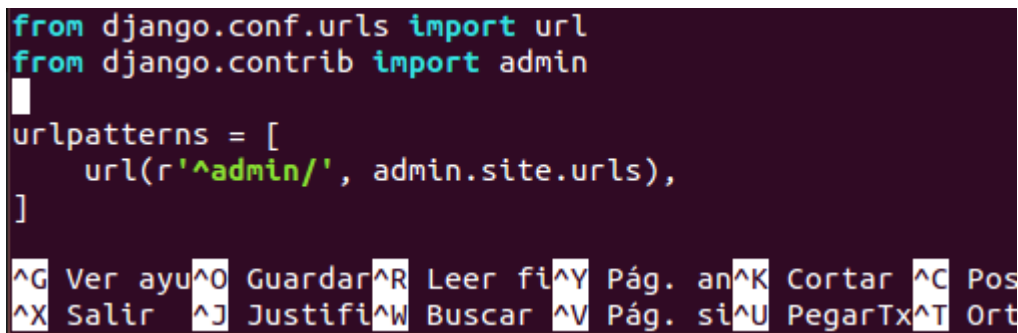
4. **ALLOWED_HOSTS** Host permitidos para el proyecto, mientras el **DEBUG** este en **True** esta variable puede permanecer vacía, cuando **DEBUG** cambie a **False** debes agregar un host obligatorio que puede ser el dominio de tu sitio.
5. **INSTALLED_APPS** Variable donde agregamos las aplicaciones de terceros que vayamos utilizando o las que vayamos creando.
6. **TEMPLATES** También podemos configurar los templates gracias a esta variable, actualmente en la versión 1.8 podemos utilizar 2 motores de templates, los de **django** y **jinja2**.
7. **DATABASES** Variable de configuración de Base de Datos, aquí podemos configurar cualquier motor de base de datos que Django utilice (MySQL, Postgres, Oracle, etc).
8. **STATIC_URL** Variable que utilizamos al momento de enlazar nuestros archivos estáticos en los templates.

Estas son las variables de entorno que mas iremos modificando. Podemos ver la [documentación oficial](#)

2.1.4. Archivo urls.py

Cada página en Internet necesita su propia URL. De esta manera tu aplicación sabe lo que debe mostrar a un usuario que abre una URL. En Django se utiliza algo que se llama URLconf (configuración de URL). URLconf es un conjunto de patrones que Django intentará comparar con la URL recibida para encontrar la vista correcta. Basicamente, este archivo contiene las rutas que están disponibles en el proyecto, manejado por URLConf.

En el archivo `urls.py` del proyecto nos encontramos con algo como esto:



```
from django.conf.urls import url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
]
```

Figura 4: Interior del archivo `urls.py`

Dentro de `urlpatterns` se encuentra la url del administrador (`admin`). Esto significa que para cada URL que empieza con `admin/` Django encontrará su correspondiente view.

Podemos ver una descripción completa de como trabaja en la [documentación oficial](#)

2.2. Archivos Predeterminados De La Aplicación

Las aplicaciones se crean con el comando **django-admin startapp**. Esta debe crearse dentro de el directorio del proyecto. Por ejemplo, si tenemos el proyecto *upaseoporlaciencia* los archivos de la aplicación deberan estar dentro de este, quedandonos algo asi:

```
ezequiel@ezequiel-VirtualBox:~/unpaseoporlaciencia$ ls
db.sqlite3  manage.py  unpaseoporlaciencia  vistaprevia
ezequiel@ezequiel-VirtualBox:~/unpaseoporlaciencia$ cd vistapre
via
ezequiel@ezequiel-VirtualBox:~/unpaseoporlaciencia/vistaprevia$
ls
admin.py    __init__.py  models.py    urls.py
admin.pyc  __init__.pyc models.pyc    views.py
apps.py     migrations   tests.py     views.py~
ezequiel@ezequiel-VirtualBox:~/unpaseoporlaciencia/vistaprevia$
```

Figura 5: Proyecto y app

Observación: Una app es una aplicación web que hace algo – e.g., un sistema de blog, una base de datos de registros públicos o una aplicación simple de encuestas. Un proyecto es una colección de configuración y apps para un sitio web particular. Un proyecto puede contener múltiples app. Una app puede estar en múltiples proyectos

Se puede ver que en el proyecto esta el directorio *unpaseoporlaciencia* que es donde se encuentran los archivos del proyecto y luego esta el directorio vista previa. En este directorio se encuentran los archivos de la app. Vamos a describir detalladamente que hace cada uno.

2.2.1. __init__.py

La misma descripción anterior (líneas arriba).

2.2.2. models.py

En este archivo se declaran las clases del modelo. En general, cada modelo se asigna a una única tabla de base de datos.(Ver: **Modelo**).

Lo básico que hay que entender sobre el funcionamiento del modelo es:

- Cada modelo es una clase de python que a su vez es una subclase de **django.db.models.Model**. Es decir los modelos van a extender de este último.
- Cada atributo del modelo representa un campo de la base de datos

2.2.3. views.py

En este archivo se declaran las funciones de la vista. Es donde ponemos la lógica de nuestra aplicación. Esta solicitará la información del modelo y la pasará al Template.(Ver: **vista**)

2.2.4. test.py

En este archivo se declaran las pruebas necesarias para la aplicación.(Ver: **test**)

3. Desarrollo De Un Proyecto + APP

En esta sección vamos a ver como es el acoplamiento base de una app, en particular utilizaremos un ejemplo base de una pagina sencillo para conocer como se deben manejar los archivos dentro del patron M.V.T.

Observación: *En un futuro voy a realizar un ejemplo mas complejo sobre un blog de ciencias el cual se podra ver en el siguiente repositorio [github](#). En este se encuentran plantillas base de html y css a las que luego le aplicare django y tambien se ira describiendo los pasos en un pdf similar a este para el que quiera saber un poco mas como funciona*

Utilizaremos:

- windows 10
- python 3.4
- django 1.10.1
- postgresql como base de datos

Podemos utilizar un entorno global o un entorno virtual. En este caso utilizaremos un entorno global.

3.1. Crear Proyecto

Como dijimos anteriormente, para crear el proyecto debemos utilizar el comando **django-admin.py startproject <nombreProyecto>**. Desde el cmd podemos verlo de esta manera:

```

C:\Users\ezequ>cd Desktop
C:\Users\ezequ\Desktop>django-admin.py startproject proyecto_django
C:\Users\ezequ\Desktop>
C:\Users\ezequ\Desktop>dir rproyecto_django
El volumen de la unidad C es Windows
El numero de serie del volumen es: EEEC-0F01

Directorio de C:\Users\ezequ\Desktop
No se encuentra el archivo
C:\Users\ezequ\Desktop>dir proyecto_django
El volumen de la unidad C es Windows
El número de serie del volumen es: EEEC-0F01

Directorio de C:\Users\ezequ\Desktop\proyecto_django
14/05/2019  13:29    <DIR>          .
14/05/2019  13:29    <DIR>          ..
14/05/2019  13:29             813 manage.py
14/05/2019  13:29    <DIR>          proyecto_django
                        1 archivos             813 bytes
                        3 dirs  804.342.009.856 bytes libres
C:\Users\ezequ\Desktop>
  
```

Figura 6: Creación y visualización del proyecto

Lo primero que vemos es el empaquetador `manage.py`. Con este vamos a poder crear las migraciones, hacer las migraciones a mi base de datos y correr el servidor. Los comandos para esto son (respectivamente):

- `python manage.py makemigrations APP --empty -n NOMBRE`
- `python manage.py migrate`
- `python manage.py runserver`

Como django ya tiene modelos creados por defecto, antes de crear nuestra aplicación es necesario hacer las migraciones.

Primero, debemos modificar en nuestro caso el archivo `settings.py` del proyecto, ya que por defecto este archivo emplea la base de datos `sqlite` y nosotros emplearemos `postgres`, así debemos modificar la variable **DATABASES**. Si quiciéramos trabajar con `sqlite`, dejamos el archivo como viene por defecto:

```
72
73 # Database
74 # https://docs.djangoproject.com/en/1.10/ref/settings/#databases
75
76 DATABASES = {
77     'default': {
78         'ENGINE': 'django.db.backends.sqlite3',
79         'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
80     }
81 }
```

Figura 7: settings.py por defecto

Ahora, si queremos usar `postgres` debemos modificarlo. En nuestro caso:

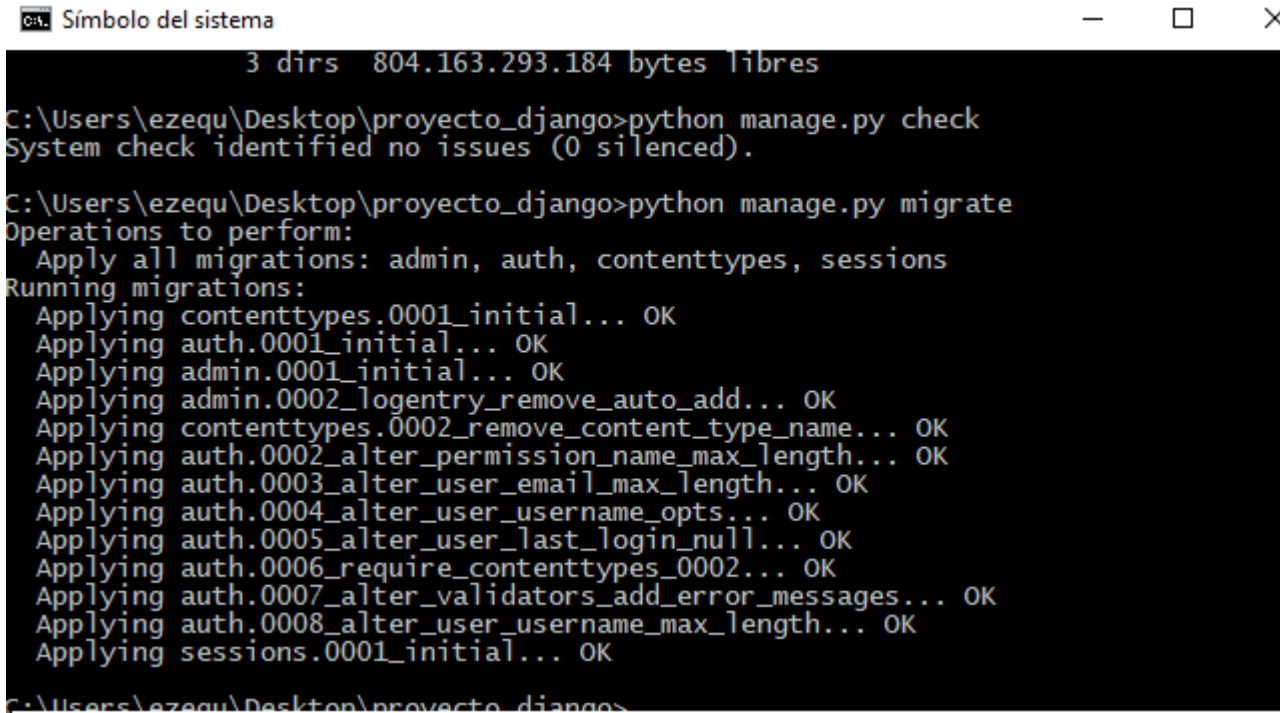
```
72
73 # Database
74 # https://docs.djangoproject.com/en/1.10/ref/settings/#databases
75
76 DATABASES = {
77     'default': {
78         'ENGINE': 'django.db.backends.postgresql_psycopg2',
79         'NAME': 'base_de_datos',
80         'USER': 'postgres',
81         'PASSWORD': 'password',
82         'PORT': '5432',
83     }
84 }
```

Figura 8: settings.py por defecto

- **ENGINE:** indica a django que el backend lo va a hacer con `postgres` o la base de datos que queramos usar.
- **NAME :** va el nombre de la base de datos
- **USER :** Es el usuario que se le da a la base de datos
- **PASSWORD:** Va a ser el password de la base de datos

- **PORT:** aca le indicamos el puerto con el que trabaja la base de datos. En el caso de postgres el puerto por defecto es el 5432

Nos paramos sobre el directorio donde se encuentra el archivo `manage.py` y ejecutamos el comando para migrar:



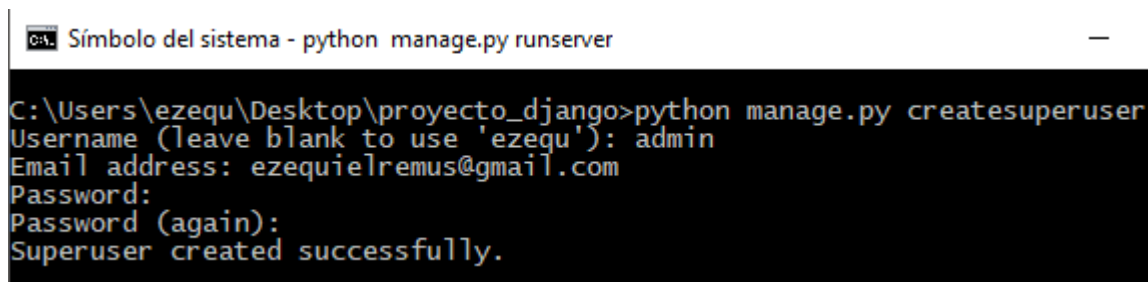
```
3 dirs 804.163.293.184 bytes libres
C:\Users\ezequ\Desktop\proyecto_django>python manage.py check
System check identified no issues (0 silenced).

C:\Users\ezequ\Desktop\proyecto_django>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying sessions.0001_initial... OK
C:\Users\ezequ\Desktop\proyecto_django>
```

Figura 9: chequeo del archivo `manage.py` y migración del proyecto

Usamos el comando **migrate** por el hecho de que solo debemos migrar las tablas creadas por defecto ya que no creamos ningún modelo aun, sino, debieramos usar **makemigrations**.

Luego de migrar, es hora de crear un superusuario, esto se hace con el comando **createsuperuser** de la siguiente forma:



```
Símbolo del sistema - python manage.py runserver
C:\Users\ezequ\Desktop\proyecto_django>python manage.py createsuperuser
Username (leave blank to use 'ezequ'): admin
Email address: ezequielremus@gmail.com
Password:
Password (again):
Superuser created successfully.
```

Figura 10: Creacion del superusuario

Por ultimo, corremos el servidor con el comando `runserver`, el cual al ejecutar nos da la dirección del servidor **http://127.0.0.1:8000**. Al entrar aquí veremos lo siguiente:

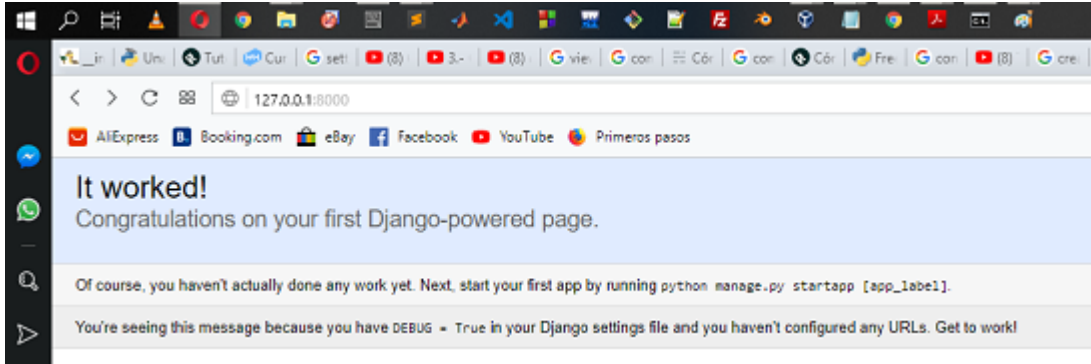


Figura 11: Corremos el servidor

Luego, podemos entrar a nuestro administrador escribiendo **`http://127.0.0.1:8000/admin`** y colocamos los datos de nuestro superusuario.

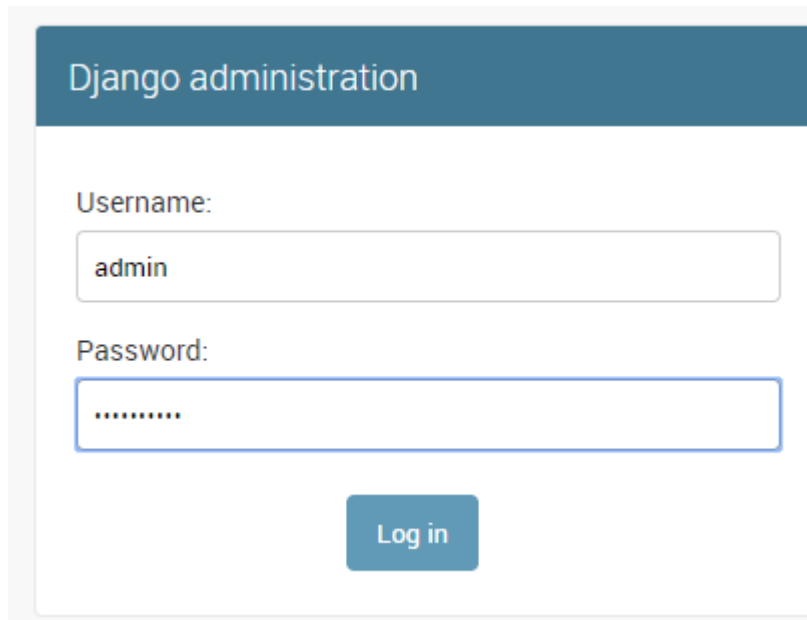


Figura 12: Corremos el servidor /admin

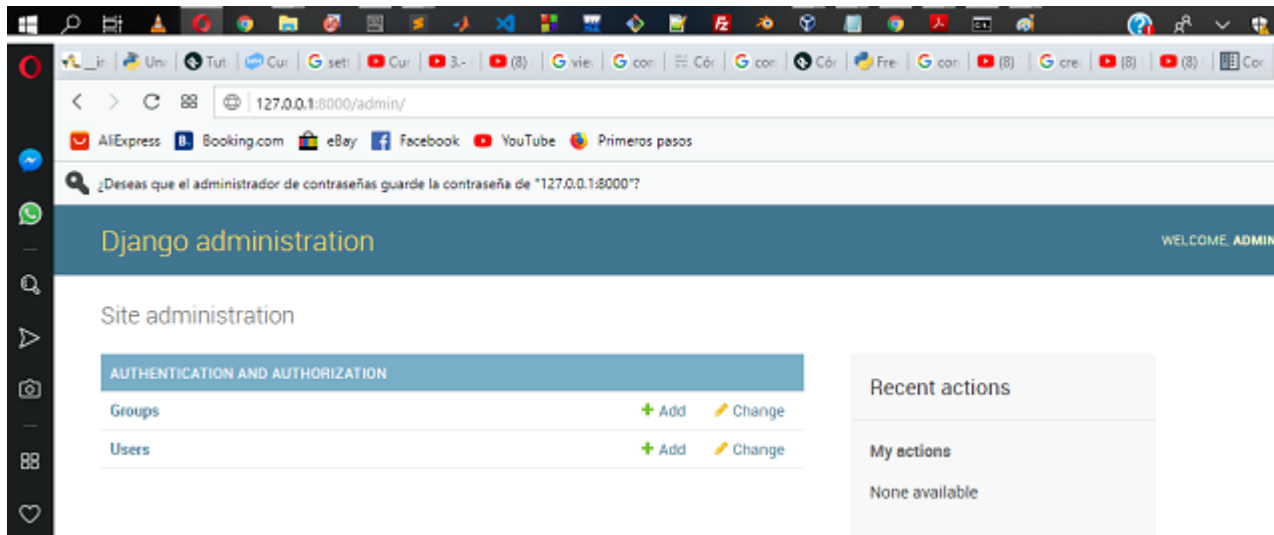


Figura 13: Sitio de administracion de django

En la **figura (13)** se ve el sitio de administracion de django. Acá vamos a poder manipular los modelos, realizar incerciones, actualizaciones, etc.

3.2. Creación de Apps y Estructura de Proyecto

Lo primero a destacar, es que no es lo mismo hablar de la aplicación web y la aplicación de django como la misma cosa. La aplicación web es el proyecto en si y la aplicación django (lo que creamos con `django-admin startapp`) es un modelo o funcionalidad del proyecto.

Imaginemos que queremos crear una aplicación de un refugio de animales. Primero, como hicimos antes crearíamos el proyecto mediante el comando `django-admin startproject <refugio>`. Vamos a ver que se crea el proyecto, luego para organizarnos mas crearemos una carpeta **apps**, donde colocaremos todas las aplicaciones referidas al proyecto refugio. Luego, nos metemos dentro de esta carpeta **apps** y ejecutamos el comando para crear una aplicación `django-admin.py startapp <nombreApp>`. En el caso del refugio podemos crear la aplicacion mascota:

```
C:\Users\ezequ\Desktop\proyecto_django\refugio>cd apps
C:\Users\ezequ\Desktop\proyecto_django\refugio\apps>django-admin.py startapp mas
cotas
C:\Users\ezequ\Desktop\proyecto_django\refugio\apps>dir
El volumen de la unidad C es Windows
El número de serie del volumen es: EEEC-0F01

Directorio de C:\Users\ezequ\Desktop\proyecto_django\refugio\apps
14/05/2019  17:20    <DIR>          .
14/05/2019  17:20    <DIR>          ..
14/05/2019  17:20    <DIR>          mascotas
                0 archivos                0 bytes
                3 dirs 803.001.556.992 bytes libres
```

Figura 14: Creación de la aplicación

Se crean los respectivos archivos de la aplicación.

```

C:\Users\ezequ\Desktop\proyecto_django\refugio\apps\mascotas>dir
El volumen de la unidad C es Windows
El número de serie del volumen es: EEEEC-0F01

Directorio de C:\Users\ezequ\Desktop\proyecto_django\refugio\apps\mascotas

14/05/2019  17:20    <DIR>          .
14/05/2019  17:20    <DIR>          ..
14/05/2019  17:20             63 admin.py
14/05/2019  17:20             91 apps.py
14/05/2019  17:20    <DIR>          migrations
14/05/2019  17:20             57 models.py
14/05/2019  17:20             60 tests.py
14/05/2019  17:20             63 views.py
14/05/2019  17:20              0 __init__.py
                6 archivos             334 bytes
                3 dirs  803.005.927.424 bytes libres
  
```

Figura 15: Archivos de la aplicación

Cabe destacar que es necesario crear un archivo `__init__.py` para el manejo de la carpeta apps, así esta puede ser reconocida como un paquete de python.

Dentro de la carpeta app vamos a poner todas las aplicaciones. Por ejemplo, además de mascotas podemos crear la app adopción la cual tendrá sus propias funcionalidades.

Ahora, para que todo funcione bien, dentro del archivo `settings.py` del proyecto, debemos modificar la variable de entorno `INSTALLED_APPS`. Aquí debemos poner la ruta de donde se ubican las aplicaciones que forman parte del proyecto.

```

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  ]
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      'apps.mascotas',
41  ]
  
```

Figura 16: Modificación del archivo settings.py para el reconocimiento de las aplicaciones que se agreguen

Como creamos un nuevo proyecto, debemos modificar la base de datos del settings, y también podríamos modificar el lenguaje.

```

110
111  LANGUAGE_CODE = 'es-ar'
112
  
```

Figura 17: Modificación del archivo settings.py (Lenguaje)

Luego de esto podemos migrar con el comando `python manage.py migrate` los datos del proyecto a la base de datos.

3.3. Creación de Modelos

Un modelo dijimos que es el archivo que manipula los datos, por lo que podemos definir un modelo como una tabla de nuestra base de datos. Los modelos van a ser clases que vana a extender del `django.db.models.Model`

Dentro de nuestra aplicación mascota, esta el archivo `models.py`. Acá es donde se crean los modelos de la aplicación.

Por ejemplo, en este ejemplo dos modelos sencillos podrian ser:

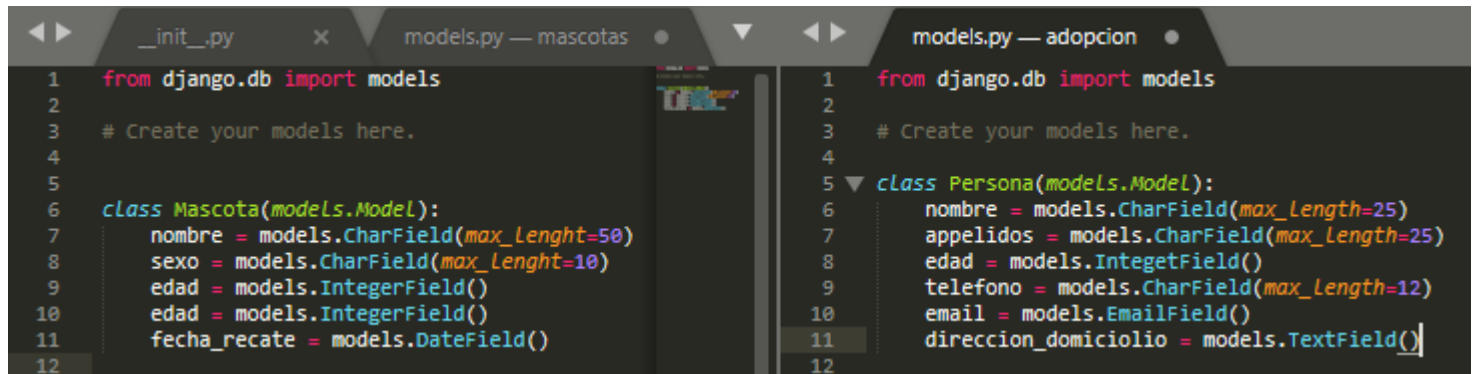


Figura 18: Ejemplos de modelos

Vemos que estos son clases de python. También se estarán preguntando que eso de `models.typeField`. Son básicamente atributos que validan al objeto en cuanto al tipo de dato. Estos se encuentran documentados [En Esta pagina](#)

También se preguntaran ¿Porqué no le pone un id?. Esto es porque django si no se lo ponemos le coloca un campo **id** del tipo Auto-Incrementable en la base de datos (recordemos que los modelos son los que interacúan con dichas tablas). Si nosotros quisiéramos, pordríamos tener un campo particular, como por ejemplo **legajo** y utilizarlo como **primarykey**. Eso se escribiría de la siguiente forma:

```
class Persona(models.Model):
    legajo = models.CharField(primary_key=True)
    nombre = models.CharField(max_length=25)
```

Figura 19: Primary_key

De esta forma, desvinculamos el id autogenerado y definimos un campo como un **primary_key**.

Ahora, cuando terminamos los modelos, lo que es comun hacer es migrar denuevo a la base de datos, pero esta vez se realizara mediante el comando **python manage.py makemigrations**. Es importate que nos paremos en el directorio del proyecto y no de las aplicaciones al hacer las migraciones. Luego de esto migramos con el comando **python manage.py migrate**.

Luego de migrar se creara un archivo llamado **0001.initial.py** el cual se ubica en la carpeta **migrations**. Este contiene codigo que genera python para la interacción con la base de tados.

The screenshot shows a code editor with two files open: `models.py` for the `adopcion` app and `models.py` for the `mascotas` app. The `mascotas` file contains the following code:

```

1  # coding: utf-8 -*-
2  # Generated by Django 1.10.2 on 2019-05-15 01:53
3  from __future__ import unicode_literals
4  from django.db import migrations, models
5
6
7
8  class Migration(migrations.Migration):
9
10     initial = True
11
12     dependencies = [
13     ]
14
15     operations = [
16         migrations.CreateModel(
17             name='Mascota',
18             fields=[
19                 ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
20                 ('nombre', models.CharField(max_length=50)),
21                 ('sexo', models.CharField(max_length=10)),
22                 ('edad', models.IntegerField()),
23                 ('fecha_recate', models.DateField()),
24             ],
25         ),
26     ]

```

To the right, a terminal window shows the execution of Django commands:

```

C:\Users\ezequ\Desktop\proyecto_django\refugio>python manage.py makemigrations
Migrations for 'mascotas':
  apps\mascotas\migrations\0001_initial.py:
    - Create model Mascota
Migrations for 'adopcion':
  apps\adopcion\migrations\0001_initial.py:
    - Create model Persona

C:\Users\ezequ\Desktop\proyecto_django\refugio>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, adopcion, auth, contenttypes, mascotas, sessions
Running migrations:
  Applying adopcion.0001_initial... OK
  Applying mascotas.0001_initial... OK

C:\Users\ezequ\Desktop\proyecto_django\refugio>

```

Figura 20: Código del archivo `0001_initial.py` creado y los comandos de migraciones

Este se crea con el comando **python manage.py makemigrations**. Luego, la migración a la base de datos la hacemos con el comando **python manage.py migrate**.

Este es un ejemplo sencillo. Podemos crear modelos más complejos utilizando relaciones

3.4. Configuración de URLs y views

Lo que primero hay que hacer para poder organizar las URLs, es crear un archivo **urls.py** en cada una de las aplicaciones. En el ejemplo tenemos `mascotas` y `asociados`, por lo que en cada una creamos estos archivos.

Imaginemos que tenemos una vista tan simple como un mensaje en pantalla.

The screenshot shows the `views.py` file for the `mascotas` app with the following code:

```

1  from django.shortcuts import render
2  from django.http import HttpResponse
3  # Create your views here.
4
5
6  def index(request):
7      #Este es un objeto de la clase HttpResponse
8      #Que nos devuelve un texto por pantalla
9      return HttpResponse("Texto")
10
11

```

Figura 21: Archivo `view.py` de la app `mascota`

La pregunta ahora es ¿Cómo hacemos para que las URLs detecten esta vista?. Para esto es que creamos los `urls.py`. Además, debemos notar que en nuestro proyecto también tenemos un archivo **urls.py** creado por defecto que tiene el siguiente código:


```
15
16 from django.conf.urls import url
17 from django.contrib import admin
18
19 urlpatterns = [
20     url(r'^admin/', admin.site.urls),
21 ]
22
```

Figura 22: Código del archivo urls.py creado por defecto

La url que esta por defecto en este archivo es la url perteneciente al admin de python, el que corremos con `http://127.0.1:8000/admin`. En este archivo se van a encontrar las urls globales. Para que nosotros podamos incluir las urls de las aplicaciones en las urls globales, primero debemos crear el archivo `urls.py` en las app. Luego debemos, dentro de estos archivos definir las urls. Esto se hace dentro de la variable `urlpatterns`. Dentro de esta se ponen todas las urls de la aplicación (Cabe destacar que `urlpatterns` es una tupla de python).

Modifiquemos el `urls.py` de la mascota de la siguiente manera:

```
1 from django.conf.urls import url, include
2 from apps.mascotas.view import index
3
4 urlpatterns = [
5     url(r'^$', index),
6 ]
```

Figura 23: Código de urls.py de la app mascota

Acá se puede ver que se importa la función `index` anteriormente creada en el archivo `view.py`, luego dentro de la variable `urlpatterns` definimos la url (`url(regex, view, kwargs=None, name=None)`) en la cual le pasamos la vista `index` como parametro. La vista se va a ejecutar cuando se realice la petición de la url.

Luego, para poder introducir las urls definidas en este archivo en las urls globales del archivo `urls.py` del proyecto, lo que se hace es *crear una url dentro de la variable `urlpatterns` de dicho archivo donde le incluimos la ruta de donde estan las urls de la app `mascotas` de la siguiente forma:*

```
16 from django.conf.urls import url, include
17 from django.contrib import admin
18
19 urlpatterns = [
20     url(r'^admin/', admin.site.urls),
21     url(r'^', include('apps.mascotas.urls')),
22 ]
23
```

Figura 24: Agregado de la url de la aplicación mascota dentro de la variable global del proyecto

Luego, al correr el servidor podremos entrar y ver la salida la cual es simplemente la palabra `Texto`.

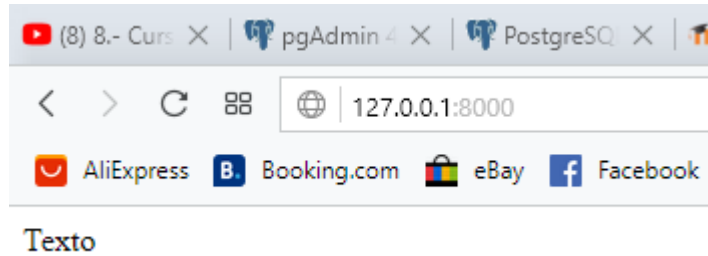


Figura 25: Vista de la url

También, podemos organizar la url mediante el **regex**, que es el primer parámetro pasado a **url()**. Por ejemplo, si modificamos las url globales agregando lo siguiente:

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^mascota/', include('apps.mascotas.urls'))],
```

Figura 26: Modificación de las url globales

Lo que hacemos agregando **mascota/** en el primer parámetro es básicamente modificar el link final que deberás utilizar para entrar a las vistas pertenecientes a la app.

Ahora, para entrar a la vista debemos escribir **http://127.0.0.1:8000/mascota/**

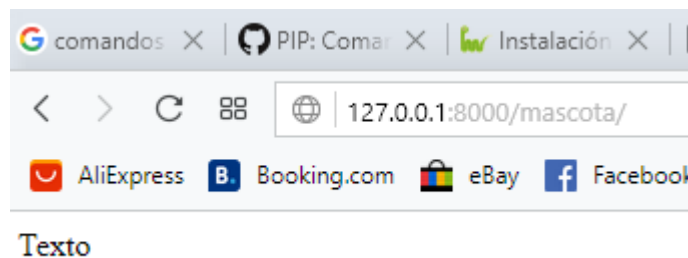


Figura 27: La vista modifica su dirección en la url global

Ahora, esto se da al modificar la global. ¿Que pasa si modificamos el parámetro regex de la url de la aplicación?

Ahora, si por ejemplo, ponemos:

```
1 from django.conf.urls import url
2
3 from apps.mascotas.views import index
4
5 urlpatterns = [
6     url(r'^index', index),
7 ]
```

Figura 28: Agregado de regex en la app mascota

Con esto lo que hacemos es darle otro parámetro de búsqueda a la dirección. En este caso, primero deberíamos pararnos en el servidor, luego en la url global y luego en la url de la app para poder ver la vista. Modificaríamos la dirección de la siguiente manera: **http://127.0.0.1:8000/mascota/index**

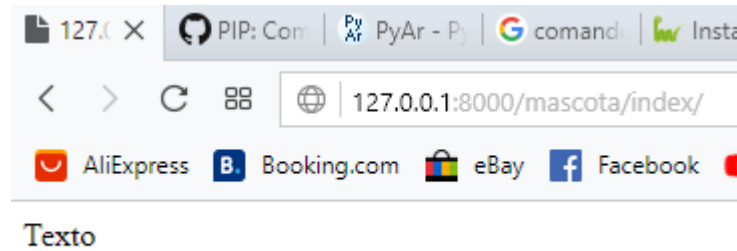


Figura 29: Agregado de regex en la app mascota

3.5. Sistemas de Plantillas

Una plantilla de Django es una cadena de texto que pretende separar la presentación de un documento de sus datos. Una plantilla define rellenos y diversos bits de lógica básica (esto es, etiquetas de plantillas) que regulan cómo debe ser mostrado el documento. Normalmente, las plantillas son usadas para producir HTML, pero las plantillas de Django son igualmente capaces de generar cualquier formato basado en texto. Este sistema de plantillas nos brinda 3 herramientas:

1. **Variables** : A estas las vamos a poder identificar por siempre estar encerradas entre llaves dobles *Variable*. Estas se utilizan para renderizar valores, es decir, cuando queramos mostrar valores que envía la vista al template (plantilla).
2. **Tags** (etiquetas): Los tags los identificamos por estar encerrados entre llaves y signos de porcentaje. Estos nos dan un flujo de control o carga de información externa de los templates. Estos se hacen mediante bucles for (**% for condicion % lo que queremos que bulee % endfor %**), sentencias if (**% if condicion % lo que queremos someter a sentencia % endif %**), cargar archivos en el template (**%load staticfiles %**).
3. **Herencia de Plantillas**: Esta herramienta nos servirá para reducir la duplicación de elementos comunes del template (Recuerdan que hablamos de DRY) como por ejemplo las etiquetas html **title**, **header**, **Navbarm**, etc. Con esto podremos crear un esqueleto que contendrá todo y así podremos heredar estos esqueletos en los templates

Para indicarle a Django la ubicación de los templates debemos ir al archivo **settings.py** del proyecto y modificar la variable de entorno **TEMPLATES**.

Cuando abrimos, el settings.py nos encontramos con esto:

```
56 TEMPLATES = [  
57     {  
58         'BACKEND': 'django.template.backends.django.DjangoTemplates',  
59         'DIRS': [],  
60         'APP_DIRS': True,  
61         'OPTIONS': {  
62             'context_processors': [  
63                 'django.template.context_processors.debug',  
64                 'django.template.context_processors.request',  
65                 'django.contrib.auth.context_processors.auth',  
66                 'django.contrib.messages.context_processors.messages',  
67             ],  
68         },  
69     },  
70 ]  
71
```

Figura 30: Template sin modificar

A este debemos modificarle el **DIRS**, para indicarle donde debe buscar las plantillas. Para esto debemos modificar esto agrgando la siguiente linea:

```
55  
56 TEMPLATES = [  
57     {  
58         'BACKEND': 'django.template.backends.django.DjangoTemplates',  
59         'DIRS': [os.path.join(BASE_DIR, 'templates')],  
60         'APP_DIRS': True,  
61         'OPTIONS': {  
62             'context_processors': [  
63                 'django.template.context_processors.debug',  
64                 'django.template.context_processors.request',  
65                 'django.contrib.auth.context_processors.auth',  
66                 'django.contrib.messages.context_processors.messages',  
67             ],  
68         },  
69     },  
70 ]  
71
```

Figura 31: Template modificado

Este le dice que busque una carpeta que se llama *templates* que esta en la raiz de la carpeta contenedora del proyecto.

Luego el elemento **APP_DIRS**, que esta en True es para que busque en la carpeta de cada una de nuestras aplicaciones, por lo que nos permite organizarnos con una sola carpeta donde la raiz de nuestra carpeta contenedora de nuestro proyecto donde podemos almacenar tooodos nuestros templates o otra manera puede ser en cada aplicación generamos otro directorio que se llame templates y DIR va a buscar tambien en estos.

Referencias

- [1] *StackOverflow*, Link: <https://stackoverflow.com>
- [2] *Contenidos de la documentación de Django*, Link: <https://docs.djangoproject.com/en/2.2/ref/templates/builtins/>
- [3] Django Software Foundation *Tutorial de Django*.
Link: <http://docs.python.org.ar/tutorial/django/download/tutorial-1.5.pdf>
- [4] Adrian Holovaty y Jacob Kaplan-Moss , *El libro de Django*. Link:
<http://bibing.us.es/proyectos/abreproy/12051/fichero/libros%252Flibro-django.pdf>
- [5] Cuenta propia de github .
Link: <https://github.com/remusezequiell?tab=repositories>
- [6] Repositorio de github donde se encuentra el codigo de este proyecto y algunos ejercicios basicos .
Link: <https://github.com/remusezequiell/Python>
- [7] Repositorio en el cual se encontrara la el html y css que se modificara a futuro .
Link: <https://github.com/remusezequiell/IntentoDeBaseBlogHtmlCss>
- [8] Comunidad Python Argentina , .
Link: <http://www.python.org.ar>

4. Apéndices

4.1. Comandos Basicos Windows

CD : Sirve para cambiar de directorio, utilizando la fórmula *cd < RutaDirectorio >* para ir al directorio o carpeta concreta que le digas, o *cd..* (con dos puntos) para salir de una carpeta e ir al nivel superior o carpeta donde estaba alojada.

DIR El comando lista el contenido del directorio o carpeta donde te encuentras, mostrando todas las subcarpetas o archivos que tiene. Con este comando podrás saber si el archivo que buscas está ahí o a qué subcarpeta navegar.

TREE CARPETA Te muestra el árbol de directorios de una carpeta concreta que le digas

CLS Limpia la ventana de la consola de Windows, borrando todo lo que se ha escrito en ella, tanto tus comandos como las respuestas de la propia consola. Se quedará todo como si la acabases de abrir de nuevo.

EXIT Cierra la ventana de la consola de Windows.

HELP Muestra todos los comandos que hay disponibles, poniendo en cada uno una breve descripción en inglés.

COPY ARCHIVO DESTINO Copia uno o más archivos en la dirección que tu elijas.

ROBOCOPY Una función mejorada más rápida y eficiente, y que permite hacer acciones como cancelar o retomar la copia. Muestra también un indicador de progreso, lo que lo convierte en una buena alternativa para copiar archivos pesados.

MOVE ARCHIVO DESTINO Mueve el archivo concreto que quieras del lugar o carpeta en el que está a otra dirección que le digas. Es como copiar, pero sin dejar el archivo en su ubicación original.

DEL ARCHIVO O CARPETA Borra el archivo o carpeta que le indiques.

RENAME ARCHIVO Te permite cambiarle el nombre al archivo que consideres oportuno, e incluso incluyendo su extensión también puedes cambiarla. Aunque será un cambio como el que haces en la interfaz principal de Windows, sin conversión y sin que implique que va a funcionar bien con la nueva extensión.

MD NOMBREDECARPETA Crea una carpeta con el nombre que le asignes en la dirección en la que te encuentres en ese momento.

TYPE ARCHIVO.EXTENSION Te permite crear un archivo desde la propia ventana de comandos. Esto quiere decir que no sólo vas a crear un archivo, sino que también podrás escribir el texto que quieras en su interior.

FORMAT Sirve para formatear una unidad de disco duro.

4.2. Comandos Python (pip)

Instalación pip:

Debian / Ubuntu

Si utilizas Debian o Ubuntu puedes utilizar apt-get para descargar e instalar pip de una manera rápida y simple. Sin embargo, generalmente este paquete se encuentra desactualizado, o al menos no en su última versión, por lo que puede que prefieras utilizar el método convencional.

```
sudo apt-get install python-pip
```

Fedora

Al igual que en **Debian** o **Ubuntu**, generalmente el paquete no se encuentra en su última versión.

```
sudo yum install python-pip
```

Windows, Linux y OS X

Primero, descarga el archivo instalador desde [acá](#) . Si tu navegador lo abre en lugar de descargarlo, presiona *CTRL* + *S* (o *CTRL* + *G*) para guardarlo en tu ordenador. Después, una vez situado con la terminal en donde has guardado get-pip.py, ejecuta el siguiente comando.

```
python get-pip.py
```

Si utilizas Windows y no tienes el directorio de Python en PATH, deberás ejecutar:

```
C:\PythonXY \python.exe get-pip.py
```

Donde *X* e *Y* corresponden al número de versión de Python. También puedes utilizar la ruta completa del archivo:

Windows: C:\PythonXY \python.exe Mis documentos \Descargas\get-pip.py

Linux: python Descargas \get-pip.py

Para comprobar que pip se ha instalado, ejecuta en la terminal:

```
pip
```

Y deberás ver las opciones de uso de la herramienta. Si, en Windows, obtienes "pip" no se reconoce como un comando interno o externo, programa o archivo por lotes ejecutable, deberás acceder manualmente.

```
C:\PythonXY \scripts\pip
```

Intalar Paquetes:

Para instalar un paquete desde **PyPI** utiliza:

```
pip install paquete
```

En donde paquete es el nombre de un módulo, librería, script o framework. Este se debe encontrar en **PyPI**. Ejemplo:

Linux: **pip install django**

Windows: **python -m pip install django**

Luego, para desinstalar paquetes usamos:

pip uninstall paquete

pip freeze

Muestra que paquetes tenemos instalados y estan vinculados con python.

```
C:\Users\ezequ\Desktop\proyecto_django\proyecto_django>pip freeze
DEPRECATION: Python 3.4 support has been deprecated. pip 19.1 will be the last o
ne supporting it. Please upgrade your Python as Python 3.4 won't be maintained a
fter March 2019 (cf PEP 429).
certifi==2019.3.9
chardet==3.0.4
Cython==0.29.7
Django==1.10.2
docutils==0.14
idna==2.8
Kivy-examples==1.10.1
Kivy-Garden==0.1.4
kivy.deps.angle==0.1.6
kivy.deps.gstreamer==0.1.12
Pillow==5.4.1
psycpg2==2.8.2
Pygments==2.3.1
requests==2.21.0
urllib3==1.24.1
virtualenv==16.4.3
C:\Users\ezequ\Desktop\proyecto_django\proyecto_django>
```

Figura 32: pip freeze de mi entorno global

pip list

Este comando sirve para listar paquetes instalados.

```
C:\Users\ezequ\Desktop\proyecto_django\refugio>pip list
DEPRECATION: Python 3.4 support has been deprecated. pip 19.1 wi
thon as Python 3.4 won't be maintained after March 2019 (cf PEP
Package          Version
-----
certifi           2019.3.9
chardet           3.0.4
Cython            0.29.7
Django            1.10.2
docutils          0.14
idna              2.8
Kivy-examples     1.10.1
Kivy-Garden       0.1.4
kivy.deps.angle   0.1.6
```

Figura 33: pip freeze de mi entorno global

Si queremos conocer cuales son los paquetes desactualizados usamos el comando *pip list --outdated*

pip show paquete

Sirve para ver la información de un paquete instalado

```
C:\Users\ezequ\Desktop\proyecto_django\refugio>pip show django
DEPRECATION: Python 3.4 support has been deprecated. pip 19.1 will be the l
thon as Python 3.4 won't be maintained after March 2019 (cf PEP 429).
Name: Django
Version: 1.10.2
Summary: A high-level Python Web framework that encourages rapid development
Home-page: http://www.djangoproject.com/
Author: Django Software Foundation
Author-email: foundation@django project.com
License: BSD
Location: c:\python34\lib\site-packages
```

Figura 34: pip freeze de mi entorno global