TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- Instalando Django.
- El utilitario PIP.
- Instalación en un entorno virtual.
- Django dentro de un entorno virtual de Python.
- Uso de un entorno virtual.
- Instalación de Django.
- Comprobar la instalación de Django.
- Creando un proyecto Django.
- El utilitario django-admin y manage.py.
- Otros comandos y obtener ayuda en tiempo de ejecución.

CREACIÓN DE UN PROYECTO DJANGO

INSTALANDO DJANGO

Django se puede configurar de una manera sencilla y fácil en el computador, con la finalidad de comenzar a desarrollar aplicaciones web. A continuación, se describe cómo podemos configurar el entorno de desarrollo, y revisar una visión general de la configuración.

El entorno de desarrollo es una instalación de Django en el computador local, la cual se utiliza para desarrollar y probar aplicaciones Django antes de desplegarlas al entorno de producción.

Las principales herramientas que el mismo Django proporciona son un conjunto de scripts de Python para crear y trabajar con proyectos Django, junto con un simple servidor web de desarrollo que se utiliza para probar de forma local (es decir, en el computador de desarrollo, no en un servidor Web externo) aplicaciones Web Django con el explorador Web del computador.

También tenemos herramientas periféricas, que forman parte del entorno de desarrollo, como lo es el editor de textos o IDE para editar código, herramienta de gestión del control de fuentes como Git para gestionar con seguridad las diferentes versiones del proyecto.

Django es extremadamente flexible en términos de cómo y dónde puede instalarse y configurarse.

Puede ser:

- Instalado en diferentes sistemas operativos.
- Usado con Python 3 y Python 2.
- Instalado desde las fuentes, Python Package Index (PyPi), y en muchos casos desde la aplicación de gestión de paquetes del computador.
- Configurado para poder ser usado entre varias bases de datos, que pueden también necesitar ser instaladas y configuradas por separado.
- Ejecutado en el entorno Python del sistema principal, o separados dentro de entornos virtuales Python.

Las opciones antes mencionadas requieren configuraciones y adecuaciones particulares para cada una. Las aplicaciones web Django pueden ejecutarse en casi cualquier máquina donde pueda funcionar el lenguaje de programación Python: Windows, Mac OS X, Linux/Unix, Solaris, por mencionar algunos. Django se ejecuta por encima de Python, y puede usarse tanto con Python 2, o con Python 3 (o ambos).

Para seleccionar la versión de Python, se debe tener en cuenta:

- Python 2 es una versión tradicional del lenguaje que no va a tener más características nuevas, pero que mantiene disponible para los desarrolladores un enorme repositorio de bibliotecas de terceros de alta calidad (quizá algunas no disponibles en Python 3).
- Python 3 es una actualización de Python 2, aunque similar, es más consistente y fácil de usar. Python 3 también es el futuro de Python, y continúa evolucionando.
- Es posible soportar ambas versiones usando bibliotecas.
- Por lo general, se recomienda usar la última versión de Python 3, al menos que el sitio dependa de bibliotecas de terceros que sólo están disponibles para Python 2.

Los sitios de descarga para Django son:

- El Python Package Repository (PyPi), usando la herramienta PIP. Este es el mejor modo de obtener la última versión estable de Django.
- Instalar por medio del gestor de paquetes del computador. Las distribuciones de Django que se empaquetan con los sistemas operativos ofrecen un mecanismo de instalación.
- Instalar desde la fuente. Se obtiene y descarga la versión de la última actualización de Python partiendo de las fuentes. No es recomendable para principiantes, pero es necesario si deseas contribuir codificando el propio Django.

Por defecto, Django soporta cuatro bases de datos importantes (PostgreSQL, MySQL, Oracle y SQLite), y existen bibliotecas de la comunidad que proporcionan varios niveles de soporte para otras bases de datos populares SQL y NOSQL.

En el ambiente de desarrollo es recomendable hacer uso de la misma base de datos, tanto para la producción, como para el desarrollo (Django abstrae muchas de las diferencias entre las bases de datos por medio de su Object Relational Mapper - ORM).

EL UTILITARIO PIP

`pip` es un sistema de gestión de paquetes, utilizado para instalar y administrar paquetes de software escritos en Python. Muchos paquetes pueden ser encontrados en el Python Package Index (PyPI). Python 2.7.9, y posteriores (en la serie Python2), Python 3.4, y posteriores, incluyen pip (pip3 para Python3) por defecto.

La ventaja importante de pip es la facilidad de su interfaz de línea de comandos, la cual permite instalar paquetes de software de Python fácilmente por medio de una sola orden de comando:

```
$ pip install nombre-paquete
```

Los usuarios también pueden fácilmente desinstalar algún paquete:

```
$ pip uninstall nombre-paquete
```

El comando `pip` permite gestionar listas de paquetes y sus números de versión correspondientes, a través de un archivo de requisitos. Esto nos permite una recreación eficaz de un conjunto de paquetes en un entorno separado (p. ej. otro computador de desarrollo), o entorno virtual. Se puede conseguir con un archivo correctamente formateado `requerimientos.txt` y la siguiente orden:

```
$ pip install -r requerimientos.txt
```

Con pip es posible instalar un paquete para una versión concreta de Python, sólo es necesario reemplazar `${versión}` por la versión de Python que queramos: 2. 3, 3.9, entre otros:

```
$ pip${version} install nombre-paquete
```

INSTALACIÓN EN UN ENTORNO VIRTUAL

Cuando instalas Python3, obtienes un único entorno global que es compartido con todo el código Python3 en el sistema operativo. Si bien puedes instalar los paquetes que te gusten en el entorno, sólo puedes instalar al mismo tiempo una versión en particular de cada uno.

Si instalas Django dentro del entorno, por defecto/global sólo podrás apuntar a una sola versión de éste en el ambiente de desarrollo. Esto puede generar un problema si se desea crear nuevos sitios (usando la última versión de Django) pero manteniendo los sitios web que dependen de versiones más antiguas.

Como resultado, los desarrolladores experimentados de Python/Django normalmente ejecutan las aplicaciones Python dentro de entornos virtuales Python independientes. De esta forma se habilitan entornos Django diferentes en el mismo ambiente de desarrollo. Por defecto, el mismo equipo de desarrollo Django recomienda el uso de entornos Python virtuales.

Para instalar y usar Django, se debe instalar Python en el ambiente de desarrollo. Si se utiliza Python 3 es necesario tener la herramienta `Python Package Index (pip3)` para gestionar (instalar, actualizar y eliminar) los paquetes/bibliotecas Python usados por Django, y otras aplicaciones de Python.

En Ubuntu, Linux incluye Python 3 por defecto. Para verificar qué versión tenemos instalada, ejecutamos el siguiente comando en la terminal:

```
$ python3 -V
Python 3.8.3
```

Sin embargo, la herramienta `Python Package Index` es la que necesitarás para instalar paquetes de Python 3 (incluido Django), y si no está disponible por defecto, se puede instalar `pip3`, haciendo uso de la terminal `bash`:

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

DJANGO DENTRO DE UN ENTORNO VIRTUAL DE PYTHON

Las bibliotecas que se utilizan para crear entornos virtuales son: `virtualenvwrapper` (Linux and Mac OS X), y `virtualenvwrapper-win` (Windows), y a su vez, utilizan la herramienta `virtualenv`. Las herramientas `wrapper` crean una interfaz consistente para la gestión de interfaces en todas las plataformas.

Después de instalar Python y PIP, puedes instalar `virtualenvwrapper` (que incluye `virtualenv`) usando `pip3` como se muestra.

```
$ sudo pip3 install virtualenvwrapper
```

A continuación, añade las siguientes líneas al final del archivo de inicio de tu `shell` (éste es un archivo oculto llamado `.bashrc`, que se encuentra en tu directorio de inicio del usuario). Esto ajusta la localización de donde deberían vivir los entornos virtuales, la de los directorios de tus proyectos de desarrollo, y la del script instalado con este paquete:

```
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
export PROJECT_HOME=$HOME/Devel
source /usr/local/bin/virtualenvwrapper.sh
```

A continuación, volver a recargar el fichero de inicio, ejecutando el siguiente comando en la terminal:

```
source ~/.bashrc
```

Una vez que hayas instalado `virtualenvwrapper`, o `virtualenvwrapper-win`, trabajar con entornos virtuales es muy similar en todas las plataformas.

Se puede crear un nuevo entorno virtual con el comando `mkvirtualenv`. A medida que se ejecuta este comando, verás que se va poniendo en marcha el entorno (notarás que es ligeramente específico de la plataforma). Cuando se completa el comando, el nuevo entorno virtual estará activo — podrás comprobarlo porque el comienzo del `prompt` será el nombre del entorno entre paréntesis (como se muestra abajo).

```
$ mkvirtualenv my_django_environment
Running virtualenv with interpreter /usr/bin/python3
...
virtualenvwrapper.user_scripts                                creating
/home/ubuntu/.virtualenvs/t_env7/bin/get_env_details
(my_django_environment) ubuntu@ubuntu:~$
```

Una vez que estás dentro del entorno virtual, puedes instalar Django e iniciar el desarrollo.

Nota: De ahora en adelante, en este artículo (y por ende en el módulo) debes asumir que todos los comandos se ejecutan en un entorno virtual Python, tal como el que acabamos de poner en marcha arriba.

USO DE UN ENTORNO VIRTUAL

Estos son los que usarás de forma habitual:

- `deactivate` — Salir del entorno virtual Python actual.

- `workon` — Listar los entornos virtuales disponibles.
- `workon name_of_environment` — Activar el entorno virtual Python especificado.
- `rmvirtualenv name_of_environment` — Borrar el entorno especificado.

INSTALACIÓN DE DJANGO

Una vez que has creado el entorno virtual, y realizado la llamada `workon` para entrar en él, puedes usar `pip3` para instalar Django.

```
pip3 install django
```

Puedes comprobar que está instalado Django ejecutando el siguiente comando (esto sólo comprueba que Python puede encontrar el módulo Django):

Linux/Mac OS X:

```
python3 -m django --version
```

Windows:

```
py-3 -m django --version
```

COMPROBAR LA INSTALACIÓN DE DJANGO

La prueba de arriba funciona, pero no es muy divertida. Una comprobación más interesante es crear un esqueleto de proyecto, y corroborar si ésta funciona. Para hacerlo, navega primero en tu consola de comandos/terminal donde quieras almacenar tus aplicaciones Django. Crea una carpeta para la comprobación de tu sitio, y navega a ella.

```
kdir django_test
cd django_test
django-admin shell
> import django
> print(django.__version__)
```

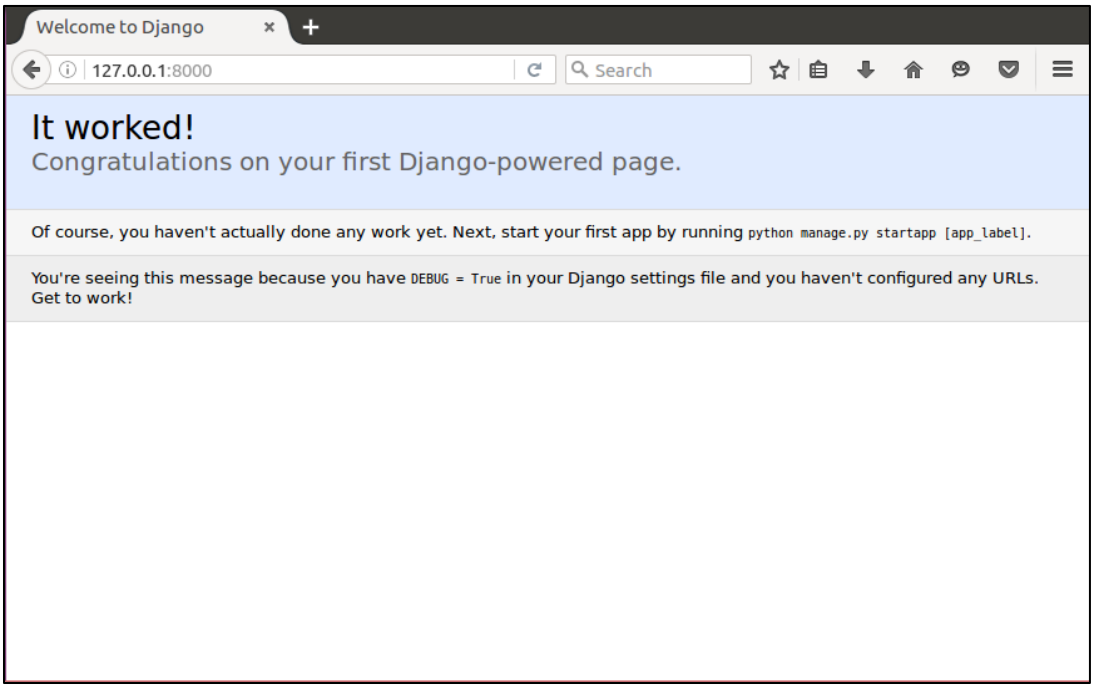
A continuación, puedes crear un nuevo esqueleto de sitio llamado "`mytestsite`" usando la herramienta `django-admin`, tal como se mostrará más adelante. Después de crear el sitio, puedes navegar a la carpeta donde encontrarás el script principal para la gestión de proyectos, llamado: `manage.py`.

```
1 django-admin startproject mytestsite
2 cd mytestsite
```

Podemos arrancar el servidor web de desarrollo desde esta carpeta usando `manage.py`, y el comando `runserver`, como se muestra.

```
 1 $ python3 manage.py runserver
 2 Performing system checks...
 3 System check identified no issues (0 silenced).
 4 You have 13 unapplied migration(s). Your project may not work properly until
 5 you apply the migrations for app(s): admin, auth, contenttypes, sessions.
 6 Run 'python manage.py migrate' to apply them.
 7 September 19, 2021 - 23:31:14
 8 Django version 1.10.1, using settings 'mysite.settings'
 9 Starting development server at http://127.0.0.1:8000/
10 Quit the server with CONTROL-C.
```

Una vez que tengas funcionando el servidor, puedes ver el sitio navegando a la siguiente URL en tu explorador web local: http://127.0.0.1:8000/. Deberías poder observar un sitio parecido a éste:



CREANDO UN PROYECTO DJANGO

¿Cómo crear un proyecto en Django, y cómo comenzar con Django en Linux?

Todos los comandos que se usarán a continuación son los mismos para todas las distribuciones de Linux/Windows.

- Primero, necesitamos crear un directorio donde podamos almacenar los proyectos de Django y las configuraciones. Por lo general, Django se opera con la interfaz del navegador web, por lo que crearemos un directorio.
- Se ejecutan las líneas de comando que se indican a continuación para crear un directorio. Aquí, estamos nombrando el proyecto como `mysite`, la instrucción para crear un proyecto (`django-admin`), comando `startproject`:

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

La estructura de carpetas de un proyecto Django. Veamos lo que el comando `startproject` creó:

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

Estos archivos son:

- `manage.py`: una utilidad de la línea de comandos que le permite interactuar con este proyecto Django de diferentes formas.
- El interior del directorio `mysite/` es el propio paquete de Python para su proyecto. Su nombre es el nombre del paquete de Python que tendrá que utilizar para importar todo dentro de éste (por ejemplo, mysite.urls).
- `mysite/__init__.py`: un archivo vacío que le indica a Python que este directorio debería ser considerado como un paquete Python. Si es un principiante, lea más sobre los paquetes en la documentación oficial de Python.
- `mysite/settings.py`: ajustes/configuración para este proyecto Django. Django settings le indicará todo sobre cómo funciona la configuración.
- `mysite/urls.py`: las declaraciones URL para este proyecto Django; una «tabla de contenidos» de su sitio basado en Django.
- `mysite/wsgi.py`: un punto de entrada para que los servidores web compatibles con WSGI puedan servir su proyecto.

EL UTILITARIO DJANGO-ADMIN Y MANAGE.PY

`django-admin` es la utilidad de línea de comandos de Django para tareas administrativas.

Además, manage.py se crea automáticamente en cada proyecto de Django. Hace lo mismo que `django-admin`, pero también establece la variable de entorno `DJANGO_SETTINGS_MODULE` para que apunte al archivo settings.py de su proyecto.

El script `django-admin` debe estar en la ruta de su sistema si instaló Django a través de pip. Si no está en su camino, asegúrese de tener su entorno virtual activado.

Generalmente, cuando se trabaja en un solo proyecto de Django, es más fácil usar manage.py que `django-admin`. Si necesita cambiar entre varios archivos de configuración de Django, use `django-admin` con `DJANGO_SETTINGS_MODULE`, o la opción de línea de comando `--settings`.

Sintaxis:

```
$ django-admin <comando> [opciones]
$ manage.py <comando> [opciones]
$ python -m django <comando> [opciones]
```

El comando debe ser uno de los comandos enumerados options, que es opcional, ya sea cero o más de las opciones disponibles para el comando dado.

OTROS COMANDOS Y OBTENER AYUDA EN TIEMPO DE EJECUCIÓN

Django viene con una serie de comandos de administración incorporados, que utilizan `python manage.py [command]`.

Éstos son algunos de los más utilizados:

- Obtener una lista con todos los comandos de ayuda disponibles:
  `./manage.py help`
- Ejecutar servidor Django en localhost: 8000; esencial para las pruebas locales:
  `./manage.py runserver`
- Ejecutar una consola de python (o ipython si está instalada), con la configuración de Django de un proyecto (intentar acceder a partes de un proyecto en un terminal de python, sin hacer esto fallará):
  `./manage.py Shell`
- Crear un nuevo archivo de migración de base de datos en función de los cambios que haya realizado en sus modelos. Ver migraciones:
  `./manage.py makemigrations`
- Aplicar migraciones, si no se habían hecho, a una base de datos:
  `./manage.py migrate`
- Ejecutar la suite de prueba de un proyecto. Ver Unidad de Pruebas:

`./manage.py test`

- Tomar todos los archivos estáticos de un proyecto, y colocarlos en la carpeta especificada en `STATIC_ROOT` para que puedan ser servidos en producción:

  `./manage.py collectstatic`

- Permitir crear superusuario:

  `./manage.py createsuperuser`

- Cambia la contraseña de un usuario específico:

  `./manage.py changepassword username`