Apuntes de web2py

Sergio Alvariño salvari@gmail.com

Agosto-2019

Resumen

Apuntes de web2py: Un framework para desarrollo de aplicaciones web

Índice general

1 Introducción

web2py es un *framework* para facilitar el desarrollo de aplicaciones web escrito en Python.

web2py funciona correctamente en Python 3. Su curva de aprendizaje no es tan empinada como la de Django (que es el *framework* de aplicaciones web de

referencia en Python) y en muchos sentidos es más moderno que Django.

web2py tiene una documentación muy completa y actualizada (disponible también en castellano) y sobre todo una comunidad de usuarios y desarrolladores muy activa y que responden con rapidez a las dudas que puedas plantear.

web2py está basado en el modelo MVC

web2py incorpora *Bootstrap 4*

1.1 Referencias

- Página de documentación y recursos de web2py, con enlaces a grupos de usuarios y tutoriales
- Evolución del modelo MVC
- Fat models and thin controllers
- Crítica del mantra

2 Empezar rápido

2.1 Instalación

Vamos a ver el proceso de instalación de una instancia de ***web2py*** en modo *standalone*. ***web2py*** instalado de esta forma es ideal para entornos de desarrollo. Para un entorno de producción puede ser más conveniente instalar ***web2py*** tras un servidor web como Apache o Nginx, pero dependiendo de la carga de trabajo y de como administres tus sistemas puede ser mejor opción usarlo *standalone* también en producción.

1. Creamos un entorno virtual

   Como ya hemos comentado ***web2py*** funciona ya en Python 3. Y en cualquier caso, con Python nunca está de mas encapsular nuestras pruebas y desarrollos en un entorno virtual.[1] Así que creamos el virtualenv que llamaremos *web2py*:

   ```
   mkvirtualenv -p `which python3` web2py
   ```

[1] Los siguientes comandos asumen que tienes instalado *virtualenvwrapper* como recomendamos en la guía de postinstalación de Linux Mint, si no lo tienes te recomendamos crear un virtualenv con los comandos tradicionales

2. Bajamos el programa de la web de Web2py y descomprimimos el framework:

```
# creamos un directorio (cambia el path a tu gusto)
mkdir web2py_test
cd web2py_test

# bajamos el programa de la web y descomprimimos
wget https://mdipierro.pythonanywhere.com/examples/static/web2py_src.zip

# opcionalmente borramos el zip, aunque sería mejor guardarlo
# por si queremos hacer nuevas instalaciones
rm web2py_src.zip
```

3. Generamos certificados para el protocolo *ssl*:

   Para usar con comodidad web2py conviene que nos generemos unos certificados para gestionar el ssl:

```
# nos movemos al directorio de web2py
cd web2py

openssl genrsa -out server.key 2048
openssl req -new -key server.key -out server.csr

Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:A Coruna
Locality Name (eg, city) []:A Coruna
Organization Name (eg, company) [Internet Widgits Pty Ltd]:BricoLabs
Organizational Unit Name (eg, section) []:Division de Hackeo
Common Name (e.g. server FQDN or YOUR name) []:testServer@bricolabs.cc
Email Address []:contacto@bricolabs.cc

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:secret1t05
An optional company name[]:Asociacion BricoLabs
```

   Y ahora ejecutamos:

```
openssl x509 -req -days 365 -in server.csr \
-signkey server.key -out server.crt
```

4. Servidor de base de datos.

Para usar ***web2py*** es imprescindible tener acceso a un servidor de base de datos. Podemos usar *MySQL* o *MariaDB* por ejemplo. Pero para empezar rápidamente vamos a tirar de SQLite, un servidor fácil de instalar potente y versátil. Es importante usar la versión 3 que introduce grandes mejoras sobre el antiguo *SQLite*

```
sudo apt install sqlite3
```

5. Arrancamos el servidor:

   Deberíamos tener los ficheros generados en el paso anterior: `server.key`, `server.csr` y `server.crt`, en el directorio raiz de web2py. Podemos arrancar el servidor con los siguientes parámetros (recuerda activar el entorno virtual si no lo tienes activo):

   ```
   python web2py.py -a 'admin_password' -c server.crt -k server.key \
   -i 0.0.0.0 -p 8000
   ```

   Y ya podemos acceder nuestro server ***web2py***, con nuestro navegador favorito, visitando la dirección https://localhost:8000

Y ahora si que ya tenemos todo listo para empezar a usar ***web2py***. Podemos crear nuestra primera aplicación.

2.1.1 Los detalles tenebrosos

Si tienes mucha prisa por aprender web2py puedes saltarte esta sección e ir directamente a la sección siguiente

Si por el contrario quieres entender exactamente que hemos hecho para poder arrancar el ***web2py*** continuar leyendo puede ser el primer paso.

¿Qué es un *virtualenv*? Python nos permite definir *virtualenv*. Un *virtualenv* es un entorno python aislado. Todos los *virtualenvs* están aislados entre si y mejor todavía son independientes del python del sistema. Esto te permite tener multiples entornos de desarrollo (o producción) cada uno con distintas versiones de python y diferentes librerias python instaladas en cada uno de ellos, o quizás diferentes versiones de las mismas librerias.

¿Que es *virtualenvwrapper*? Es un frontend para usar *virtualenv*, la herramienta nativa de python para gestionar *virtualenvs*. Es completamente opcional, aunque a mi me parece muy cómoda.

¿Qué es todo eso de los certificados? ***web2py*** viene preparado para usar *https* (estas siglas tienen varias interpretaciones: *HTTP over TLS*, *HTTP*

over SSL o *HTTP Secure*). *https* usa comunicaciones cifradas entre tu navegador y el servidor web para garantizar dos cosas: que estás accediendo al auténtico servidor y que nadie este interceptando la comunicación entre navegador y servidor. En particular ***web2py*** exige que se use *https* para conectarse a las páginas de administración. Así que si no generas los certificados podrás arrancar y conectar con ***web2py*** pero no podrás hacer demasiadas cosas.

Para usar *https* hay que hacer varias cosas:

- Generar un CSR (Certificate Signing Request)
- Obtener con ese CSR un certificado SSL de una autoridad certificadora (CA)
- O alternativamente generar nosotros un certificado a partir del CSR

Lo que hemos hecho con los comandos *openssl* ha sido:

- Generar un par de claves (privada y pública) para nuestro servidor (`server.key`)
- Generar con esa clave un CSR (el CSR lleva la información que le hemos metido de nuestro servidor y la clave pública)
- Generar un certificado firmándolo nosotros mismos con esa misma clave como si fueramos la autoridad certificadora.

Esto nos vale para arrancar ***web2py*** aunque nuestro navegador nos dará una alerta de riesgo de seguridad por que no reconoce a la CA.

Más info de *openssl*

¿Qué es un motor de base de datos? ***web2py*** usa un motor (o gestor) de base de datos relacional. Puede usar muchos, incluyendo los más populares como por ejemplo MySQL, Postgres o MariaDB.

Las bases de datos relacionales se basan en relaciones. Las relaciones primarias son tablas que almacenan registros (filas) con atributos comunes (columnas). Las relaciones derivadas se establecen entre distintas tablas mediante consultas (queries) o vistas (views)

web2py te permite gestionar y utilizar las bases de datos a muy alto nivel, así que podras usarlo sin saber practicamente de bases de datos; pero no es demasiado difícil aprender los conceptos básicos y compensa ;-) Todo lo que puedas aprender de bases de datos te ayudará a hacer mejores aplicaciones web.

2.2 Nuestra primera aplicación

Vamos a crear nuestra primera aplicación en web2py.

Si has seguido los pasos de la sección anterior ya tienes el ***web2py*** funcionando y puedes seguir cualquiera de los tutoriales que hay en la red para aprender. El capítulo 3 del libro de ***web2py*** es muy recomendable, y está disponible en castellano, puedes ventilarte los ejemplos que trae explicados en una tarde y son muy ilustrativos.

En esta guía vamos a ver la creación de una aplicación paso a paso. Crearemos una aplicación de inventario para el material de la Asociación BricoLabs, pero lo haremos de manera que también nos valga para uso particular y tener controladas todas nuestras cacharradas.

Este no es un tutorial de diseño profesional de aplicaciones, solo pretendemos demostrar lo fácil que es iniciarse con ***web2py***.

De hecho, no seguiremos un orden lógico en el diseño de la aplicación, si no que intentaremos seguir un orden que facilite conocer el framework.

Sin más rollo, vamos a comenzar con nuestra aplicación:

Crea una aplicación desde el interfaz de administración, en nuestro caso la llamaremos ***cornucopia***.

Nuestro ***web2py*** "viene de serie" con algunas aplicaciones de ejemplo. La propia pantalla inicial es una de ellas la aplicación "Welcome" o "Bienvenido" (dependerá del lenguaje por defecto de tu navegador).

Para crear nuestra aplicación ***cornucopia***:

- Vamos al botón **admin** en la pantalla principal.
- Metemos la password de administración (con la que hemos arrancado el ***web2py*** en la linea de comandos).
- Desde la ventana de administración creamos nuestra nueva aplicación

Inmediatamente nos encontraremos en la ventana de diseño de nuestra nueva aplicación. ***web2py*** nos permite diseñar completamente nuestra aplicación desde aquí, ni siquiera necesitaremos un editor de texto (aunque nada impide usar uno, desde luego).

2.2.1 `private/appconfig.ini`

El primer fichero que vamos a examinar es `private/appconfig.ini` La sección `private` debería estar abajo de todo en la ventana de diseño.

En la sección `[app]` del fichero podemos configurar el nombre de la aplicación y los datos del desarrollador.

En la sección `[db]` fichero configuramos el motor de base de datos que vamos a usar en nuestra aplicación. Por defecto viene configurado *sqlite* así que no vamos a tener que cambiar nada en este sentido.

En la seccion `[smtp]` podemos configurar el gateway de correo que usará la aplicación para enviar correos a los usuarios. Por defecto viene viene la configuración para usar una cuenta de gmail como gateway, solo tenemos que cubrir los valores de usuario y password y la dirección de correo.[2]

2.2.2 El Modelo

En la parte superior de la ventana de diseño (o edición) de nuestra aplicación tenemos la sección `Models`

Figura 1: Menú Modelos

web2py se encarga de crear las tablas necesarias en la base de datos que le hayamos indicado que use.

Al crear la aplicación **_web2py** ha creado en la base de datos todas las tablas relacionadas con la gestión de usuarios y sus privilegios.

Si echamos un ojo al modelo gráfico (*Graphs Models*) veremos las tablas que ***web2py*** ha creado por defecto y las relaciones entre ellas. Estas tablas que ha creado el *framework* son las que se encargan de la gestión de usuarios, sus privilegios y el acceso de los mismos al sistema, es decir la capa de seguridad.

[2] Es aconsejable crear una cuenta de gmail, o cualquier otro servicio de correo que nos guste, para pruebas. Usar tu cuenta de correo personal podría ser muy mala idea

Si vemos el log de comandos de sql (*sql.log*) veremos los comandos que ***web2py*** ha ejecutado en el motor de base de datos.

Y por último si vemos *database administration* podremos ver las tablas creadas en la base de datos, e incluso crear nuevos registros en esas tablas (de momento no lo hagas)

También podemos echar un ojo al contenido del fichero `db.py` o `menu.py` pero por el momento **no** vamos a modificar nada en esos ficheros.

Ahora tenemos que ampliar el modelo y añadir todo lo que consideremos necesario para nuestra aplicación.

2.2.2.1 Diseñando el modelo

Build fat models and thin controllers es uno de los lemas del modelo MVC, no vamos a entrar en detalles de momento pero un modelo bien diseñado nos va a ahorrar muchísimo trabajo al construir la aplicación.

El diseño de bases de datos es una rama de la ingeniería en si mismo, hay camiones de libros escritos sobre el tema y todo tipo de herramientas para ayudar al diseñador. Pero nosotros nos vamos a centrar en usar sólo lo que nos ofrece ***web2py***.

Además como estamos aprendiendo vamos a ver algunas facilidades que nos da ***web2py*** sin proponer ningún proceso de diseño del modelo (recuerda, esto no es un curso de diseño de aplicaciones)

Vamos a definir el modelo (concretamente las tablas) de nuestra aplicación en un nuevo fichero de la sección *Models*, que llamaremos `db_custom` así que pulsamos en el botón *Create*, y creamos el fichero `db_custom`.

web2py parsea todos los ficheros de la sección *Models* por orden alfabético. Esto nos permite separar nuestro código del que viene originalmente con la aplicación. Pero es importante que `db.py` sea siempre el primero alfabeticamente para que se ejecute antes que el resto.

web2py se encarga también de añadir la extensión `.py` al nuevo fichero que estamos creando así que teclea sólo el nombre `db_custom`.

El objetivo de nuestra aplicación es mantener un inventario de “cosas”. Parece lógico que nuestra primera tabla valga para almacenar “cosas”. Así que en el fichero `db_custom.py` añadimos las siguientes lineas y salvamos el fichero:

```
db.define_table('thing',
    Field('id', 'integer'),
```

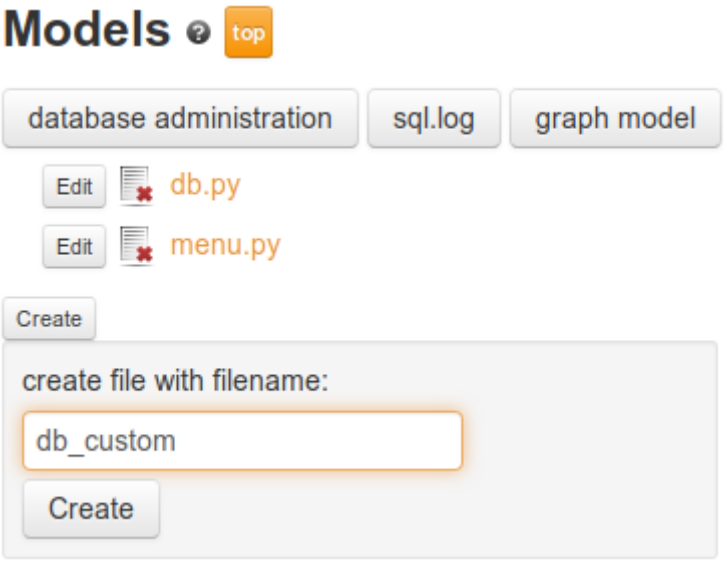

Figura 2: Crear fichero

```
    Field('desc', 'string'),
    migrate = True);
```

2.2.2.2 Tickets de error Ya hemos salvado nuestro fichero, vamos a echar un ojo a nuestra base de datos con el botón *Graph Model*.

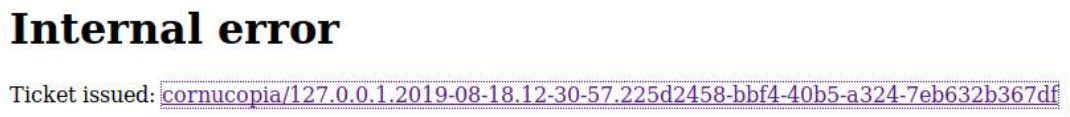

Figura 3: Error interno

¡Tenemos un horror! ¿Qué ha pasado?. Si pinchamos en el link del *Ticket* se abrirá una nueva pestaña en nuestro navegador:

En el ticket tenemos mucha información acerca del error, afortunadamente en este caso es facilito. El nombre del campo `desc` que hemos añadido a nuestra tabla `thing` es una palabra reservada en **todas** las variedades de *SQL* (es el comando para ver la definición de una tabla: *desc tablename*)

Editamos de nuevo nuestro fichero `db_custom.py` y corregimos el contenido:

Error ticket for "cornucopia"

Ticket ID

127.0.0.1.2019-08-18.12-30-57.225d2458-bbf4-40b5-a324-7eb632b367df

<type 'exceptions.SyntaxError'> **invalid table/column name "desc" is a "ALL" reserved SQL/NOSQL keyword**

Version

web2py™ Version 2.18.5-stable+timestamp.2019.04.08.04.22.03
Python Python 2.7.15+: /usr/bin/python (prefix: /usr)

Traceback

```
Traceback (most recent call last):
  File "/home/salvari/work/estudio/web2py/web2py/gluon/restricted.py", line 219, in restricted
    exec(ccode, environment)
  File "/home/salvari/work/estudio/web2py/web2py/applications/cornucopia/models/db_custom.py", line 5, in <module>
    migrate = True);
  File "/home/salvari/work/estudio/web2py/web2py/gluon/packages/dal/pydal/base.py", line 592, in define_table
    table = self.lazy_define_table(tablename, *fields, **kwargs)
  File "/home/salvari/work/estudio/web2py/web2py/gluon/packages/dal/pydal/base.py", line 604, in lazy_define_table
    table = table_class(self, tablename, *fields, **kwargs)
  File "/home/salvari/work/estudio/web2py/web2py/gluon/packages/dal/pydal/objects.py", line 354, in __init__
    check_reserved_keyword(field_name)
  File "/home/salvari/work/estudio/web2py/web2py/gluon/packages/dal/pydal/base.py", line 551, in check_reserved_keyword
    'invalid table/column name "%s" is a "%s" reserved SQL/NOSQL keyword' % (name, backend.upper()))
SyntaxError: invalid table/column name "desc" is a "ALL" reserved SQL/NOSQL keyword

```

Figura 4: Error palabra reservada SQL

```
db.define_table('thing',
    Field('id', 'integer'),
    Field('description', 'string'),
    migrate = True);
```

¡Ahora si! Si pulsamos en el botón de *Graph Model* (después de salvar el nuevo contenido) veremos que ***web2py*** ha creado la nueva tabla en la base de datos. Incluso podríamos empezar a añadir filas (cosas) a nuestra tabla desde el *database administration*

El campo `id` es *casi* obligatorio en todas las tablas que definamos en ***web2py***, siempre será un valor único para cada fila en una tabla y se usará internamente como clave primaria. Podemos usar otros campos como clave primaria pero de momento mantendremos las cosas símples.

Si visitas ahora la sección de administración de la base de datos puedes añadir algunas "cosas" a la nueva tabla.

Figura 5: Añadir destornillador

2.2.2.3 Mejorando la tabla Evidentemente nuestro modelo de "cosa" es demasiado simple, tenemos que añadirle nuevos atributos de distintos tipos para que sea funcional. Pero antes de ir a por todas vamos a ver algunas funciones que nos ofrece ***web2py*** para construir los Modelos.

Vamos a añadir algunos campos más de distintos tipos a nuestro modelo y verlos con un poco de calma.

```
db.define_table('thing',
    Field('id', 'integer'),
    Field('name', 'string'),
```

```
    Field('description', 'string'),
    Field('picture',, 'upload'),
    Field('created_on, 'datetime'),
    migrate = True);
```

Si ahora volvemos al administrador de base de datos podemos comprobar que:

- No hemos perdido las "cosas" que añadimos antes, ***web2py*** ha añadido las nuevas columnas pero ha conservado los valores de las antiguas.
- Podemos editar las "cosas" que habíamos añadido sin mas que hacer click en el `id`
- Si queremos editar (o añadir) una "cosa", ***web2py*** nos ofrece un diálogo para subir la foto de nuestro objeto. Sabe que los atributos de tipo upload son fichero que subiremos al servidor.

  De la misma forma nos ofrece un menú inteligente para añadir el campo `datetime`

Este es el tipo de facilidades que ofrecen los *frameworks* para acelerar el trabajo de crear una aplicación.

Vamos a hacer un pelín más sofisticada nuestra tabla `thing`:

```
db.define_table('thing',
    Field('id', 'integer'),
   Field('name', 'string', requires = IS_NOT_EMPTY(error_message='cannot be empty')),
    Field('description', 'string'),
    Field('qty', 'integer', default=1, label=T('Quantity')),
    Field('picture', 'upload'),
    Field('created_on', 'datetime'),
    format='%(name)s',
    migrate = True);
```

En la linea del `name` hemos añadido un `VALIDATOR`. Se trata de funciones auxiliares que nos permiten comprobar multitud de condiciones y que son extremadamente útiles (iremos viendo casos de uso). En este caso exigimos que el campo `name` no puede estar vacío y además especificamos el mensaje de error que debe aparecer si sucede.

Hemos añadido un atributo `qty` (cantidad), hemos especificado que tenga un valor por defecto de una unidad, y además hemos especificado el `label`.

El `label` se usará en los formularios en lugar del nombre del campo en la base de datos. Si vamos a añadir una nueva "cosa" veremos que en el formulario no

aparece *qty* sino que nos pregunta *Quantity*. Además, y esto es muy importante, hemos asignado el valor de la etiqueta con la función `T()`.

web2py incorpora un sistema completo de internacionalización. Al usar la función `T()` la cadena *Quantity* se ha añadido a todos los diccionarios de traducción (si es que no estaba ya) y solo tenemos que añadir la traducción en el diccionario correspondiente (p.ej. a `es.py`) para que funcione la i18n. Una vez añadida si el idioma por defecto de nuestro navegador es el castellano, en el formulario aparecerá "Cantidad" en lugar de *Quantity*.

Por último hemos añadido el `format` a la definición de la tabla, `format` especifica que cuando nos refiramos a un objeto "cosa" se represente por defecto con su atributo `name`.

2.2.2.4 Relaciones entre tablas Supongamos ahora que queremos tener registrado en nuestro inventario al proveedor de cada una de nuestras cosas. ¿cómo se hace eso?

Vamos a crear una tabla básica con los proveedores:

```
db.define_table('provider',
    Field('id', 'integer'),
    Field('name, 'string'),
    Field('CIF', 'string'),
    Field('email', 'string'),
    Field('phone', 'string'),
    migrate = True);
```

Y ahora a la tabla `thing` le añadimos la referencia a proveedores: ~~~~{python}

db.define_table('thing', Field('id', 'integer'), Field('name', 'string', requires = IS_NOT_EMPTY(error_message='cannot be empty')), Field('description', 'string'), Field('qty', 'integer', default=1, label=T('Quantity')), Field('picture', 'upload'), Field('created_on', 'datetime'), Field('provider_id', 'reference provider', requires=IS_EMPTY_OR(IS_IN_DB(db, 'provider.id', '%(name)s'))), format='%(name)s', migrate = True); ~~~~

3 Secciones en el futuro

3.1 web2py y git

3.2 Instalación con nginx

3.3 Certificados let’s encrypt

```
db.define_table('thing',
    Field('id', 'integer'),
    Field('description', 'string'),
    Field('picture', 'upload'),
    Field('created_on, 'datetime'),
    Field('created_by, 'reference auth_user', default = auth.user_id),
    Field('updated_on, 'datetime'),
    migrate = True);
```