

Mini Manual de Desarrollo Web con Python y Web.Py

1. Introducción

Se va a mostrar una idea clara del funcionamiento del small framework para desarrollo web como siempre de una forma rápida y concreta.

- **Qué es web.py?**

web.py es un framework web para Python que es tan simple como potente. **web.py** es de dominio público, y se puede usar para cualquier propósito sin ningún tipo de restricciones.

- **Para qué sirve web.py?**

Existen otro framework como Django o web2py demasiado robustos, pero para desarrollar de forma rápida un sitio web, web.py es lo recomendable.

- **Requisitos.**

Python, web.py y algún SGBD que en nuestro caso sera **postgreSQL** y su driver para python correspondiente **psycopg2**.

Para nuestros tutos usaremos la ultima versión hasta el momento la 0.35 de web.py

INSTALACIÓN

```
$ wget -c http://webpy.org/static/web.py-0.35.tar.gz
$ tar -xvzf web.py-0.35.tar.gz

# cd web.py-0.35/
# python setup.py install
```

Ejecutamos:

```
$ python
Python 2.5.5 (r255:77872, Nov 28 2010, 19:00:19)
[GCC 4.4.5] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> import web
>>> web.__version__ '0.35'
```

Para crear nuestra primera aplicación “hola mundo” vamos a crear un fichero hola.py:

```
1 # -*- coding: UTF-8 -*-
2 """
3 Primer ejemplo para Web.py version 0.3
4 """
5 import web
```

En la línea 5 importamos el modulo web.py el cual nos cargara todos los objetos del framework.

```
6
7 print web.__version__
8
```

Imprimimos en la linea 7 la versión de web.py que estamos usando. Ahora necesitamos decirle a web.py que URL's son validas para nuestro sitio web.

```
9 # URL:
10 urls = (
```

```

11 # 'regex para url', 'clase donde se envia la petición'
12 '/', 'index'
13 )

```

La primera parte es una expresión regular ('/')que hace referencia a la raíz de nuestro sitio web, indicándole una petición en este caso GET a nuestro servidor para que nos sirva la raíz del server. La segunda parte es el nombre de la Clase que maneja la petición GET de la expresión regular.

Ahora tenemos que crear una solicitud especificando las urls.

```

14
15 # Aplicacion donde se especifican las urls.
16 app = web.application(urls, globals())
17

```

Esto le dice a web.py que se creara una aplicacion con las direcciones URL que están en urls y buscara las clases en el espacio de nombres global de este archivo.

Ahora creamos la clase index que contendrá la definición para los métodos GET o POST según sea el caso.

```

18 # Clase index:
19 class index:
20     def GET(self):
21         return "Hola, Mundo!"
22

```

En la línea 20 definimos la función GET que será llamada por web.py en el momento que se haga una solicitud GET para /.

Por último le indicamos a web.py que empiece a servir las páginas web según la aplicación que hemos creado.

```

23 # corremos nuestro app web.py
24 if __name__ == "__main__":
25     app.run()

```

- **Iniciando el servidor**

En la línea de comando ejecutamos nuestro script como cualquier otro en python.

```

$ python hola.py
0.35

```

```
http://0.0.0.0:8080/
```

Con esto tenemos ejecutando un servidor web en nuestra máquina y solo basta con visitar <http://localhost:8080> donde veremos un mensaje:

Hola, Mundo!

Avanzando un poco más:

```

1 #!/usr/bin/env python
2
3 import web
4 rutas = (
5 '/', 'hola'
6 )
7

```

```

8 app = web.application(rutas, globals())
9
10 class hola:
11     def GET(self):
12         return """
13         <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
14         <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es"
lang="es">
15             <head>
16                 <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
17                 <title>Hola Mundo...</title>
18                 <style type="text/css" media="screen">
19                     h1 {text-align:center; color: #444;}
20                 </style>
21             </head>
22             <body>
23                 <h1>Hola Web.py</h1>
24             </body>
25         </html>
26         """
27
28 if __name__ == '__main__':
29     app.run()

```

Ahora veremos algo de plantillas las cuales nos facilitan mucho el poder integrar python a la web.

- **Plantillas en web.py**

web.py permite que desde dentro de los ficheros de HTML podamos embeber código python al estilo de otros lenguajes de script como php. Para usar plantillas en nuestros proyectos podemos hacer un directorio para almacenarlas.

```
$ mkdir templates
```

```
$ cd templates/
```

Dentro de nuestro directorio de plantillas vamos a crear nuestro fichero html, llamado **index.html**, en el cual podemos escribir etiquetas html y usar el lenguaje de plantillas de web.py.

Para nuestro ejemplo “Hola Mundo”. index.html:

```

1 $def with (nombre)
2     <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml" lang="es" xml:lang="es">
4     <head>
5         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/>
6         <meta name="author" content="Jorge Alonso Toro" />

```

```

7     <meta name="generator" content="web.py" />
8     <title>Hola Mundo</title>
9     <style type="text/css">
10
11         p {
12             font-style: italic;
13             font-family: arial;
14             color: gray;
15             margin: .8em;
16         }
17
18         span {
19             font-weight: bold;
20             font-style: normal;
21         }
22
23         #msg {
24             -moz-border-radius: 5px 5px 5px 5px;
25             background-color: #f4f4ff;
26             border: 1px solid #AACCEE;
27             width: 20em;
28         }
29         #sld {
30             -moz-border-radius: 3px 3px 3px 3px;
31             background-color: #f4f4ff;
32             border: 1px solid #AACCEE;
33             width: 20em;
34         }
35
36     </style>
37 </head>
38 <body>
39 $if nombre:
40     <div id="msg">
41         <p>Solo quer&iacute;a decirte hola<span> $nombre</span></p>
42     </div>
43 $else:
44     <div id="sld">
45         <p>Hola, <span>Mundo</span></p>
46     </div>
47 </body>
48 </html>

```

Nota: Hay que tener mucho **cuidado con la indentación**, tanto del código python como las etiquetas html. Actualmente son requeridos **4 espacios**. En vim lo podés establecer con 'set tabstop' poniendo en 2.

No profundizaremos en el diseño web con (x)html y css pero veremos algunas etiquetas y estilos básicos.

Como podemos notar el lenguaje de plantillas de web.py se asemeja mucho al lenguaje nativo de python a **diferencia de la primera línea la cual contiene los datos pasados a la plantilla cuando ésta es llamada (\$def with(name)) y el signo dolar (\$) colocado delante de cualquier código.**

Volvemos a nuestro hola.py y debajo del import, agregamos:

```

7 # Directorio de plantillas:

```

```
8 render = web.template.render('templates/')
9
```

Esto le indica a web.py donde debe buscar las plantillas. Reemplazamos el contenido de nuestra función GET a:

```
21 # Clase index:
22 class index:
23     def GET(self):
24         nombre = 'Linux'
25         return render.index(nombre)
```

En la línea 25 index es el nombre de la plantilla y 'nombre' es el argumento pasado a esta plantilla. Corramos nuestro script y visitemos nuestro sitio <http://localhost:8080/>:

```
$ python hola.py
```

0.35

```
http://0.0.0.0:8080/
```

Muy bonito pero para nada funcional, algo más funcional puede ser, que la gente pueda ingresar su nombre!. Reemplacemos las líneas 24 y 25.

```
21 # Clase index:
22 class index:
23     def GET(self):
24         i = web.input(nombre=None)
25         return render.index(i.nombre)
```

Corremos nuestro script, si nos dirigimos a <http://localhost:8080/> debemos ver el saludo "Hola, Mundo", pero si visitamos <http://localhost:8080/?nombre=gente> debemos ver el saludo "Solo quería decirte hola gente".

Por supuesto que tener un '?' en nuestra URL es un poco feo. En su lugar podemos cambiar nuestras línea de direcciones (urls).

```
12 # URL:
13 urls = (
14     # 'regex para url', 'clase donde se envía la petición'
15     '/(.*)', 'index'
16 )
```

Y cambiar la definición de nuestra función GET:

```
21 # Clase index:
22 class index:
23     def GET(self, nombre):
24         return render.index(nombre)
```

Con esto le pasamos como argumento a nuestra función GET, lo enviado por el método GET de HTTP. Corremos nuestro script y si visitamos <http://localhost:8080/gente> obtenemos el mensaje "Solo quería decirte hola gente".

- **Formularios en Web.py**

El módulo form de web.py permite generar formas html, obtener entrada del usuario, y validarlo antes de su procesamiento o su inclusión en una base de datos.

El módulo form define dos clases principales:

- **Form:** crea una instancia con una o más entradas, y los validadores opcionales.
- **Input:** crea una instancia con el nombre de una variable, y los argumentos opcionales y validadores.

La clase Input es una subclase de las siguientes entradas html (tipo de elemento html en paréntesis):

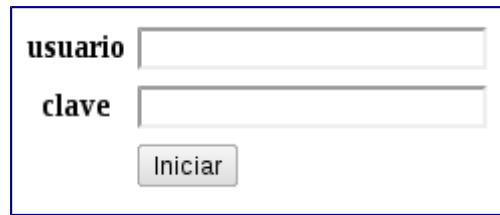
- **Textbox:** Entrada de formulario de una sola línea (`<input type="text"></input>`).
- **Password:** Entrada de formulario de una sola línea que esconde el texto (`<input type="password"></input>`).
- **Textarea:** Entrada de formulario multilínea o múltiples líneas (`<input type="textarea"></input>`).
- **Dropdown:** Lista de entrada excluyente (`<select>` y `<option>`)
- **Radio:** Entrada mutuamente excluyente para algunas opciones (`<input type="radio"></input>`)
- **Checkbox:** Entrada binaria. Si son varias entradas relacionadas, checkbox no es excluyente (`<input type="checkbox"></input>`)
- **Button:** Botón de formulario (button)

Creamos un programa básico de logueo:

```
1 #!/usr/bin/env python
2 # -*- coding: UTF-8 -*-
3 import web
4 from web import form
5
6 urls = (
7     '/', 'ingreso'
8 )
9
10 app = web.application(urls, globals())
11
12 # Creamos nuestro formulario:
13 login = form.Form(
14     form.Textbox('usuario'),
15     form.Password('clave'),
16     form.Button('Iniciar'),
17 )
18
19 class ingreso:
20     def GET(self):
21         f = login()
22         return f.render()
23
24 def main():
25     app.run()
26     return 0
27
```

```
28 if __name__ == "__main__": main()
```

Esto nos mostraría una página así:



A screenshot of a web form. It contains two input fields. The first is labeled 'usuario' and the second is labeled 'clave'. Below these fields is a button labeled 'Iniciar'.

Si reemplazamos la línea 22 “return f.render()” por “print f.render()”, en El Método GET, nos creará la siguiente salida:

```
<table>
  <tr><th><label for="usuario">usuario</label></th><td><input type="text"
id="usuario" name="usuario"/></td></tr>
  <tr><th><label for="clave">clave</label></th><td><input type="password"
id="clave" name="clave"/></td></tr>
  <tr><th><label for="Iniciar"></label></th><td><button id="Iniciar"
name="Iniciar">Iniciar</button></td></tr>
</table>
```

- **Características de los elementos de entrada**

Las entradas de formulario soportan varios atributos adicionales.

form.Textbox("fistname", form.notnull, class_="Entrada_de_texto", pre="pre", post="post", description="string", value="", id="nombre_de_id")

- *fistname*: hace referencia al atributo name de los elementos de formulario.
- *form.notnull*: poner validadores primero, seguido por los atributos opcionales.
- *class_="textEntry"*: da un nombre de clase CSS, para el cuadro de texto.
- *pre="pre"*: text antes del cuadro de texto.
- *pos="post"*: text inmediatamente después del cuadro de texto.
- *description="string"*: Describe el campo. Por defecto establece (firstname).
- *value=""*: valor por defecto. Puede establecer un valor por defecto.
- *id="nombre_de_id"*: especifica el id.
- *size="pixeles"*: Tamaño en pixeles del control.
- *maxlength="num"*: Máximo número de caracteres para los controles de texto y de password.

Por ejemplo, si modificamos nuestro formulario, agregándole:

```
12 # Creamos nuestro formulario:
13 login = form.Form(
14     form.Textbox('usuario', form.notnull, description="Username",
class_="textEntry", \
15     value="nombre de usuario", id="cajatext", post="despues",
pre="antes", size="10"),
16     form.Password('clave', description="Password", maxlength="4",
size="10"),
17     form.Button('Iniciar'),
```

18)

Este ejemplo dibujaría algo como:

Username	antes	<input type="text" value="nombre de usuario"/>	despues
Password	<input type="password"/>		
<input type="button" value="Iniciar"/>			

Nos debe aparecer en html:

```
<table>
  <tr><th><label    for="cajatext">Username</label></th><td>antes<input
name="usuario" value="nombre de usuario" class="textEntry" type="text"
id="cajatext" size="10"/>despues</td></tr>
  <tr><th><label for="clave">Password</label></th><td><input maxlength="4"
type="password" id="clave" name="clave" size="10"/></td></tr>
  <tr><th><label    for="Iniciar"></label></th><td><button    id="Iniciar"
name="Iniciar">Iniciar</button></td></tr>
</table>
```

- **Características de los elementos dropdown o desplegables**

Estas listas se crean con tuplas que permiten una descripción única y valor por cada elemento de la lista desplegable. P. e. un programa que muestre una lista desplegable:

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 #
4 import web
5 from web import form
6
7 urls=(
8     '/listas', 'Lista'
9 )
10
11 app = web.application(urls, globals())
12
13 lista = form.Form(
14     form.Dropdown('milista', \
15     [('valor1', '1000'), ('valor2', '2000')])
16 )
17
18 class Lista:
19     def GET(self):
20         l = lista()
21         return l.render()
22
23
24 def main():
25     app.run()
26     return 0
27
28 if __name__ == "__main__": main()
```

Si visitamos <http://localhost:8080/listas>. Veremos algo como:

milista 1000 ▼

- **Características de los formularios**

Además de los validadores individuales de form.py se puede realizar validaciones de comparación de campos de nuestro formulario. Los datos validadores se pasan a una list. Por ejemplo, un programa que valide si la contraseña es correcta.

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 #
4 import web
5 from web import form
6
7
8 urls=(
9     '/suscribe', 'Suscribe'
10 )
11
12 app = web.application(urls, globals())
13
14 formulario = form.Form(
15     form.Textbox('username', maxlength="10"),
16     form.Password('clave', maxlength="8", \
17     post=" se requiere un passwod de 8 caracteres"),
18     form.Password('valida_clave', maxlength="8", description="Repita la
clave"),
19     form.Button("Enviar"),
20     validators = [form.Validator("La contraseña no coincide", \
21     lambda i: i.clave == i.valida_clave)]
22 )
23
24 class Suscribe:
25     def GET(self):
26         f = formulario()
27         return f.render()
28
29 def main():
30     app.run()
31     return 0
32
33 if __name__ == "__main__":
34     main()
```

Un ejemplo, si creamos nuestra plantilla llamada *formulario_1.html*.

```
1 $def with (form)
2
3 <form>
4     $:form.render()
5     <input type="submit" />
6 </form>
```

Ahora creamos nuestro programa para que nos cree un formulario con el método GET, llamado formulario.py.

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 import web
4 from web import form
5
6 urls = (
7     '/', 'index'
8 )
9
10 plantilla = web.template.render('./templates/')
11
12 app = web.application(urls, globals())
13
14 myform = form.Form(
15     form.Textbox("nombre"),
16     form.Textbox("id",
17         form.notnull,
18         form.regexp('\d+', 'Debe ser un dígito'),
19         form.Validator('Debe ser más de 5', lambda x:int(x)>5)),
20     form.Textarea('observacion'),
21     form.Checkbox('reenviar'),
22     form.Dropdown('prioridad', ['baja', 'media', 'alta']))
23
24
25 class index:
26     def GET(self):
27         form = myform()
28         return plantilla.formulario_1(form)
29
30
31 if __name__ == "__main__":
32     app.run()

```

Cuando lo ejecutemos nos mostrará algo como esto:



En esta segunda imagen vemos cómo enviamos los datos al hacer clic en Enviar.



Esto nos creara una salida html:

```

<form> <table> <tr><th><label for="nombre">nombre</label></th><td><input
type="text"      id="nombre"      name="nombre"/></td></tr>      <tr><th><label

```

```

for="id">id</label></th><td><input                                type="text"                                id="id"
name="id"/></td></tr>                                <tr><th><label
for="observacion">observacion</label></th><td><textarea id="observacion"
name="observacion"></textarea></td></tr>                                <tr><th><label
for="reenviar_">reenviar</label></th><td><input                                type="checkbox"
id="reenviar_" value="" name="reenviar"/></td></tr>                                <tr><th><label
for="prioridad">prioridad</label></th><td><select id="prioridad"
name="prioridad">                                <option value="baja">baja</option>                                <option
value="media">media</option>                                <option value="alta">alta</option>                                </select>
</td></tr> </table> <input type="submit" /> </form>

```

EJEMPLO FUNCIONAL (MODIFICADO):

Modificando nuestro ejemplo, para que no solo muestre el formulario, si no que lo evalúe e imprima en pantalla los datos capturados.

Creemos una nueva plantilla llamada `formulario_2.html`.

```

1 $def with (form)
2
3 <!-- cuando le doy envia me llama al método POST -->
4 <form name="main" method="post">
5     <!-- valid(self, value), es un método de Validator(self, msg, test,
jstest=None)
6     y de regexp(self, rexp, msg), que hacen parte del type Textbox -->
7     $if not form.valid: <p class="error">Intenta de Nuevo</p>
8     $:form.render()
9     <input type="submit" />
10 </form>

```

Y creamos un nuevo programa con un método POST, que capture los datos enviado por el formulario.

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 import web
4 from web import form
5
6 urls = (
7     '/', 'index'
8 )
9
10 plantilla = web.template.render('./templates/')
11
12 app = web.application(urls, globals())
13
14 myform = form.Form(
15     form.Textbox("nombre"),
16     form.Textbox("id",
17         form.notnull,
18         form.regexp('\d+', 'Debe ser un dígito'),
19         form.Validator('Debe ser más de 5', lambda x:int(x)>5)),
20     form.Textarea('observacion'),
21     form.Checkbox('reenviar'),
22     form.Dropdown('prioridad', ['baja', 'media', 'alta']))
23
24
25 class index:

```

```

26 # Método de Llegada
27 def GET(self):
28     form = myform()
29     return plantilla.formulario_2(form)
30
31 # Método POST
32 def POST(self):
33     form = myform()
34     if not form.validates():
35         return plantilla.formulario_2(form)
36     else:
37         # form.d.nombre y form['nombre'].value son formas equivalente
38         # de extraer los argumentos validados del formulario.
39         return "Gran éxito! Nombre: %s, ID: %s" % (form.d.nombre,
form['id'].value)
40
41
42 if __name__ == "__main__":
43     app.run()

```

La salida sería la siguiente, si no ingresamos ningún dato y hacemos clic en “enviar”.

Intenta de Nuevo

nombre

id Required

observacion

reenviar ☐

prioridad baja ▼

Enviar

su respectivo html:

```

<!-- cuando le doy enviar me llama al método POST -->
<form name="main" method="post">
<p>Intenta de Nuevo</p>
<table>
  <tr><th><label for="nombre">nombre</label></th><td><input type="text"
id="nombre" value="" name="nombre"/></td></tr>
  <tr><th><label for="id">id</label></th><td><input type="text" id="id"
value="" name="id"/><strong>Required</strong></td></tr>
  <tr><th><label for="observacion">observacion</label></th><td><textarea
id="observacion" name="observacion"></textarea></td></tr>
  <tr><th><label for="reenviar_">reenviar</label></th><td><input
type="checkbox" id="reenviar_" value="" name="reenviar"/></td></tr>
  <tr><th><label for="prioridad">prioridad</label></th><td><select
id="prioridad" name="prioridad">
    <option selected="selected" value="baja">baja</option>
    <option value="media">media</option>
    <option value="alta">alta</option>
  </select>
</td></tr>
</table>
<input type="submit" />

```

</form>

Podemos llenar los datos.



Y luego hacer clic en “Enviar”.



Si no son dígitos en el campo “id”.



- **Formularios para bases de datos**

Una vez que los datos del formulario se han publicado, puede ser fácilmente puesto en una base de datos (si el esquema de base de datos tiene los nombres coherentes con su formulario webpy). Por ejemplo:

```
class Agregar:
    def POST(self):
        f = formulario()
        if f.validates():
            web.insert('nombre_tabla_db', **f.d)
            # No debe usar web.insert('nombre_tabla_db', **web.input()) porque los
            # datos malintencionados podrían presentarse también
        else:
            render.foo(f)
```

Esto debemos tenerlo en cuenta para el manejo de bases de datos con web.py.

- **Bases de Datos con web.py**

Antes de adentrarnos en el manejo de bases de datos, es necesario estar seguros de que contamos con las librerías apropiadas para hacerlo.

Si usas Postgres: Necesitamos [Psycopg2](#).

Si usas MySQL: Necesitamos [MySQLdb](#).

En este apartado usaremos PostgreSQL y Psycopg2. Para instalar PostgreSQL y psycopg2 puedes apoyarte en el administrador de paquetes de tu distro si estás en GNU/Linux o mirar la guía de “[Cómo Instalar y configurar PostgreSQL en Debian](#)” o “Instalar PostgreSQL desde Las fuentes en Linux”. Para comprobar si tenemos la librería psycopg2 instaladas, podés usar el CLI de Python:

```
>>> import psycopg2
>>> psycopg2.__version__
'2.2.1 (dt dec mx ext pq3)'
>>>
```

- **Creando nuestra Base de Datos.**

Para desarrollar nuestro capítulo de databasing, necesitamos crear una base de datos en nuestro SGBD.

A. Creando una Base de Datos en Postgres:

```
# su - postgres
$ psql
postgres=# CREATE DATABASE myweb owner=postgres encoding='utf-8';
CREATE DATABASE
postgres=#
```

B. Creamos una simple tabla en nuestra base de datos:

```
postgres=# \c myweb
myweb=# CREATE TABLE todo (
id serial primary key, title text,
created timestamp default now(), done boolean default 'f');
```

NOTICE: CREATE TABLE creará una secuencia implícita «todo_id_seq» para la columna serial «todo.id»

NOTICE: CREATE TABLE / PRIMARY KEY creará el índice implícito «todo_pkey» para la tabla «todo»

```
CREATE TABLE
```

```
myweb=# \d
```

Listado de relaciones

Esquema	Nombre	Tipo	Dueño
public	todo	tabla	postgres
public	todo_id_seq	secuencia	postgres

(2 filas)

```
myweb=#
```

C. Agregamos una fila a nuestra tabla:

```
myweb=# INSERT INTO todo (title) VALUES ('Aprender Web.py');
```

```
myweb=# SELECT * from todo;
```

id	title	created	done
1	Aprender Web.py	2011-11-06 08:42:20.366723	f

(1 fila)

D. Modificamos la contraseña para el usuario postgres:

```
postgres=# ALTER USER postgres WITH ENCRYPTED PASSWORD 'qwerty';
```

■ Conectando Web.py con la Base de Datos.

Para que **web.py** se conecte con la base de datos que acabamos de crear, solo es necesario crear un objeto database. Así:

```
dB = web.database(dbn='postgres', user='usuario', pw='contraseña',  
db='dbname')
```

dbn: nombre SGBD.

user: usuario de la base de datos.

pw: contraseña del usuario de la base de datos.

db: Nombre de la Base de datos.

• Testeando que la conexión funcione.

```
>>> import web  
>>> dB = web.database(dbn='postgres', user='postgres', pw='qwerty',  
db='myweb')  
>>> dB.select('todo')  
0.04 (1): SELECT * FROM todo  
<web.utils.IterBetter instance at 0x8d4d96c>  
>>>
```

Si tenemos un error de autenticación, debes modificar la autenticación de “ident” a “md5” en el fichero pg_hba.conf de postgresQL y recordar haber cambiado la contraseña de nuestro usuario postgres.

• Creando un pequeño manejador de bases de datos en Web.py.

```
1 #!/usr/bin/env python  
2 # -*- coding: utf-8 -*-  
3 #  
4  
5 import web  
6  
7 # Templates  
8 rend = web.template.render('templates')  
9  
10 # URL's  
11 urls = (  
12     '/', 'index',  
13 )  
14  
15 # Database  
16 dB = web.database(dbn='postgres', user='postgres', pw='qwerty',  
db='myweb')  
17  
18 # App  
19 app = web.application(urls, globals())  
20
```

```

21
22 class index:
23     def GET(self):
24         todos = dB.select('todo')
25         return rend.index(todos)
26
27
28 if __name__ == "__main__": app.run()
29

```

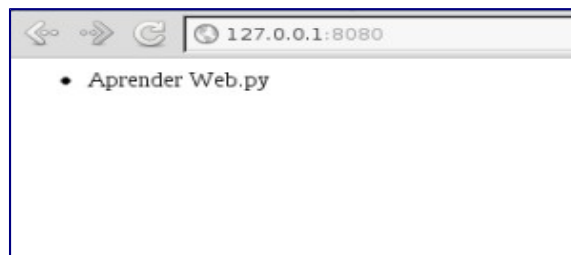
Creamos la plantilla index.html.

```

1 $def with (todos)
2 <ul>
3 $for todo in todos:
4     <li id="t$todo.id">$todo.title</li>
5 </ul>
6

```

Creo que explicar este código sobra ya que todo debe estar más que claro. Ejecutamos nuestro pequeño manejador de DB.



- **Agregando datos a nuestra Base de Datos.**

Ahora solo nos falta poder ingresar datos a la base de datos. Para eso creamos una URL para ingresar datos (puede ser "add").

```

10 # URL's
11 urls = (
12     '/', 'index',
13     '/add', 'add'
14 )

```

Al final de nuestro index.html creamos un pequeño formulario para enviar datos al servidor:

```

7 <form method="post" action="add">
8 <p><input type="text" name="title" /></p>
9 <input type="submit" value="Agregar" />
10 </form>
11

```

Agreguemos nuestra clase para manejar la url /add.

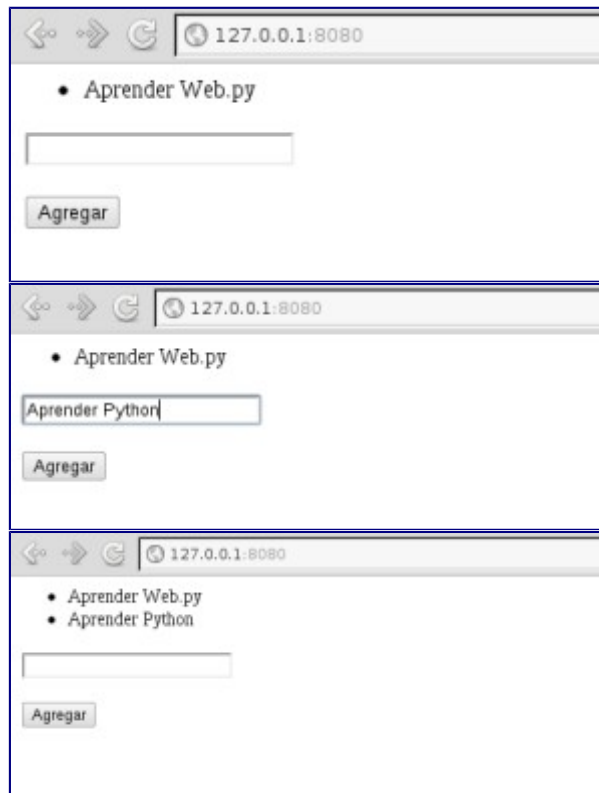
```

28 class add:
29     def POST(self):
30         i = web.input()
31         n = dB.insert('todo', title = i.title)
32         raise web.seeother('/')
33

```


Note que hemos usado el método POST para enviar datos a la base de datos.

Si ejecutamos nuestro pequeño manejador de bases de datos en web.py.



`web.input()`: le da acceso a todas las variables enviadas por el usuario a través de un formulario.
`dB.insert`: inserta los valores en la base de datos “todo” y devuelve el ID de la nueva fila.
`web.seeothers`: redirige a los usuarios a la URL indicada.

- **Notas Adicionales:**

Para poder acceder a los datos de varios elementos de idéntico nombre, de un formulario. Por ejemplo: una serie de check-boxes todos con el atributo `name="name"`. usar un formarto de lista `"post_data = web.input(name[])"`.

`dB.update`: `db.update` funciona igual que `db.insert` pero en lugar de devolver la ID que se necesita (o una cadena de cláusula `WHERE`) después de el nombre de la tabla.

`web.input`, `dB.query`, y otras funciones en `web.py` retornan “Objetos de almacenamiento” que son como los diccionarios, excepto que usted haga `d.foo` además de `d["foo"]`. Esto realmente hace un código más limpio.

- **Web.py en Producción.**

Web.py cuenta con herramientas que ayudan en la depuración. Cuando corres el servidor interno de `web.py`, esto arranca la aplicación en modo depuración. En el modo depuración cualquier cambio al código y las plantillas son automáticamente cargados así como los mensajes de error; Algo que no quisiéramos en nuestro servidor de producción.

Si querés deshabilitar el modo de depuración, vos podes agregar la siguiente línea antes de crear tu aplicación/plantillas:

```
web.config.debug = False
```

Bibliografía