¡La serpiente se ha vuelto apache!

MOD_PYTHON

Mod_Python, SQlite y Cheetah forman un cocktail explosivo que nos permitirá crear nuestro propio entorno de desarrollo de aplicaciones web. **POR JOSÉ MARÍA RUÍZ**

Día tras día aparecen nuevos frameworks que prometen hacer de la creación de páginas web dinámicas un juego de niños. Cada framework viene generalmente acompañado por un nuevo gurú, genio o ídolo que nos viene a decir que ha creado la última y definitiva herramienta que acabará con todos nuestros sufrimientos. Por desgracia, para él o ella, siempre aparece al cabo de unos meses una nueva solución que promete todo lo anterior y aún más.

Tarde o temprano puede llegar a cansar todo este bullicio sin sentido. ¿Qué framework escoger? ¿Qué ventajas tienen? ¿Cuánto tiempo tardaré en aprenderlo? ¿Cuando termine de aprenderlo aparecerá una nueva versión que mande al garete todo lo que he aprendido?

Por eso en este artículo vamos a ver cómo construir una aplicación web empleando muchas de las técnicas que nos ofrecen los frameworks, pero sin hacer uso de ninguno de ellos: construiremos el nuestro propio desde casi cero. Una vez armados con este conocimiento creo que al lector o lectora le será mucho más sencillo separar el grano de la paja cuando considere emplear un framework, y de paso puede que descubra que hay mucho más humo del que cabría esperar.

El Problema con los Frameworks

El software libre se ha especializado fundamentalmente en la creación de editores de texto y de librerías genéricas. Es un auténtico infierno enzarzarse en un debate con otro programador sobre qué editor o librería es el mejor. Y en parte el problema está en la palabra «mejor». La evolución no es la supervivencia del más fuerte. Si ese fuese el caso, sólo un tipo de organismos dominaría toda la Tierra, y probablemente no sería el ser humano.

La evolución es el proceso por el cual los organismos o sistemas sobreviven si son los mejores adaptados en base a las circunstancias específicas en las que viven. Por supuesto que estas circunstancias pueden cambiar no sólo por causas externas, ¡sino por la propia aparición de nuevos organismos!.

El resultado es que el dogmatismo y la tiranía son prácticamente antievolutivas. Por eso existen tantísimas alternativas para el mismo problema en software libre. Todo esto

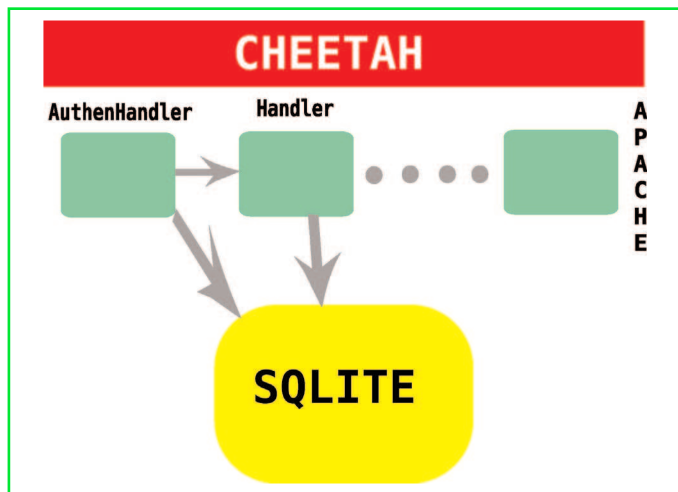


Figura 1: Esquema del paradigma MVC.

nos puede llevar a un relativismo terrible, en el que la opinión de todo el mundo cuenta y nada llega a desarrollarse.

Pero curiosamente, observamos cómo hay ciertos programas o librerías que se imponen de una u otra manera. En el mundo de los frameworks gráficos está claro que GTK y QT dominan el entorno. Lo mismo ocurre con las bases de datos, MySQL y Postgresql, o …

El mundo de la web es distinto. La web es «fácil», y casi cualquiera puede acceder a ella. De hecho, es tan «fácil» que todo el mundo tiene su propio framework, sin que la evolución haya decidido aún cuál ha ganado. Más que nos pese, la evolución sí ha marcado un patrón: PHP es el rey de la Web. PHP, tal cual, sin framework, ni forks, ni disputas. Quizá con envidia por parte de otros lenguajes, quizá con ataques debido a su simplicidad que roza lo burdo. Pero bueno… así es el lenguaje de programación *C*, que también domina la creación de sistemas.

Python, sin embargo, está inmerso en un continuo devenir de frameworks. Django, Turbogears, …

A mí, personalmente, tanto movimiento me marea. Y cuando la marea te arrastra lo mejor es asirse a una roca estable. Así que la pregunta podría ser ¿Si todo se mueve, dónde está el punto fijo (como dirían los matemáticos)? Pues la mayor parte de los frameworks se asientan en un servidor web, y el más utilizado es *Apache*.

Por más que no nos guste es *Apache* quien marca lo que un framework puede y no puede hacer, puesto que al fin y al cabo es *Apache* el que nos conecta con el exterior. Pero *Apache* no es un objeto pasivo, no es un esclavo, en realidad es mucho más potente de lo que podamos imaginar si no lo conocemos realmente. Vamos a ver cómo Python, mediante *mod_python*, hace uso de la potencia que *Apache* puede proporcionarle y descubriremos que es posible que lo hayamos subestimado.

Apache, Ese Gran Desconocido

Nuestra relación con *Apache*, ver Recurso [1], suele ser escueta. Lo instalamos, nos vamos al directorio de turno donde alberga los ficheros html, copiamos algún fichero en él, editamos algo del fichero de configuración, lo arrancamos y nos desentendemos. ¡Craso error! *Apache* es un programa maduro, cuyo código se ha puesto a prueba en innumerables situaciones.

Apache no sólo sirve para procesar ficheros html estáticos. Posee un inmenso sistema de módulos que nos permiten extender su funcionalidad hasta el infinito. Pero este artículo no versa sobre Apache, sino sobre cómo aprovecharlo con *mod_python*. Para ello debemos entender cómo funciona realmente.

El concepto principal que debemos aclarar es que Apache es una máquina muy bien diseñada. Apache divide el proceso de procesamiento de peticiones http en una serie de pasos con el objetivo de poder controlar al milímetro lo que está ocurriendo. Pero antes de enfrentarnos a Mod_Python es preciso que veamos qué es *MVC*.

MVC

Estas tres siglas se han hecho de oro en los últimos años y, como veremos, esconden en realidad un principio realmente simple. MVC viene a significar Modelo Vista Controlador. Es un acrónimo que representa una manera de diseñar programas informáticos que se remonta a los años 70 del siglo pasado (ver Figura 1).

El concepto es realmente simple: dividir el código del programa de forma que las funciones o partes del mismo que trabajan directamente con la base de datos, que generan las pantallas o html y que guardan la lógica del programa, se encuentren en sitios separados y sólo tengan contacto entre sí mediante interfaces abstractos bien definidos.

Imaginemos que tenemos una empresa en la que hay un almacén, una administración y una serie de vendedores. Llega un cliente que es atendido por el vendedor y solicita la compra de un producto. El vendedor podría vendérselo, ir al almacén, buscarlo, traerlo y dárselo, cobrando la cantidad que considere es el precio del producto.

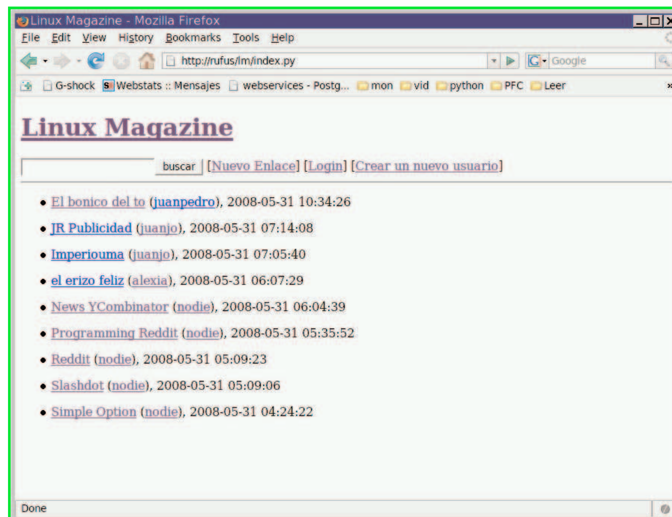


Figura 2: Nuestra Web terminada.

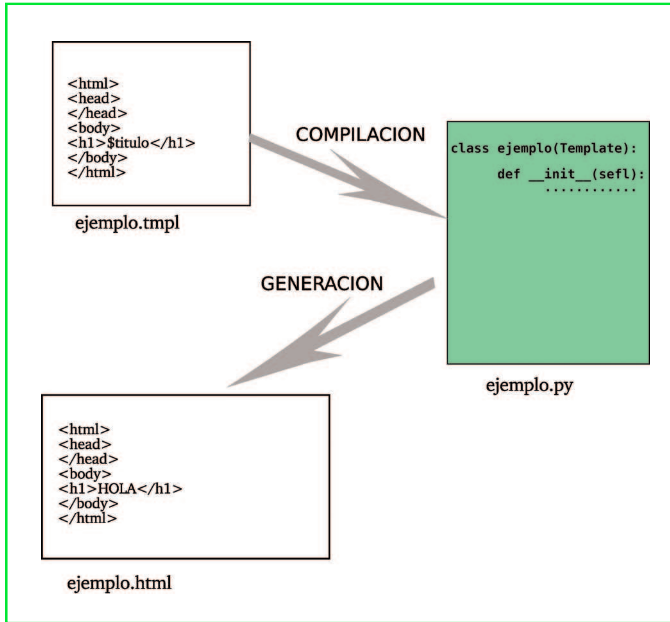


Figura 3: Esquema de las plantillas Cheetah.

Tabla 1: Eventos

EVENTO	ACCIÓN
PostReadRequestHandler	Después de leer la petición
TransHandler	Traducción de petición a una ruta de fichero
HeaderParserHandler	Cuando se reconocen las cabeceras
InitHandler	Primer handler llamado
AccessHandler	Control de restricciones de acceso
AuthenHandler	Autenticación de acceso
AuthzHandler	Autenticación específica para un recurso
TypeHandler	Gestión de tipos de documentos
FixupHandler	Modificación de cabeceras
Handler	Generación de los datos a devolver
LogHandler	Registro de eventos en ficheros de log
CleanupHandler	Antes de que se destruya el objeto Request

Esta manera de trabajar puede funcionar a pequeña escala, con pocos vendedores y si todos se conocen, pero en cuanto la empresa crezca, aparecerán una serie de problemas. Los empleados comenzarán a ver que no hay mucho control, debido a que cualquiera puede hacer cualquier cosa y, ya sea por malicia o desconocimiento, sobrevendrán los conflictos.

Por eso las empresas dividen el trabajo. Lo normal es que el vendedor «venda» el producto, una vez vendido se lo comunique a un superior que verá si realmente se puede vender ese producto (puede que el cliente no tenga crédito o no cumpla las condiciones de venta). El superior representaría la capa de Control y es el que toma las decisiones. Si todo va bien, dará orden al almacén de traer el producto. Almacén comprobará el inventario y si el producto está en condiciones de ser vendido, que haciendo las veces del Modelo, es quien almacena los datos y se dedica a manipularlos. El almacén dará el producto al vendedor, que representa la Vista, y éste se lo entregará al cliente.

El diseño, por tanto, se rompe en 3 pasos: la parte con la que el cliente interactúa (Vista), la toma de decisiones (Controlador), y la manipulación de los datos (Modelo). La experiencia ha demostrado que, tanto en las organizaciones humanas como en el diseño de programas, esta arquitectura funciona bastante bien. Nuestro diseño seguirá el patrón MVC.

Nuestra aplicación web consistirá en un sistema para que distintos usuarios añadan direcciones web con un título a una lista, un buscador y un sistema de control de sesiones (entrada, salida y creación de nuevos usuarios); la página será parecida a la que aparece en la Figura 2.

Mod_Python

Mod_Python, ver Recurso [2], es un módulo para Apache diseñado por Gregory Trubetskoy, ver Recurso [3], que nos permite conectar Python directamente con Apache. El diseño de Apache se compone de una especie de cadena de montaje en la que existen puntos de control a los que podemos conectarnos y responder a eventos producidos en ellos usando Python.

Tenemos que configurar Apache para que haga uso de Mod_Python, aunque normalmente al instalar el módulo se realizan cambios en el fichero de configuración de Apache para activarlo. Aún así es necesario comunicar a Apache en qué directorio queremos que actúe Mod_Python y de qué manera:

```
<Directory
/usr/local/www/ data/prueba>
 AddHandler mod_python .py
 PythonHandler index
 PythonAuthenHandler index
 PythonDebug On
 AuthType Basic
 AuthName "Area Restringida"
 require valid-user
</Directory>
```

Tenemos que incorporar un código como el anterior a nuestro fichero de configuración para Apache (el nombre del fichero y su localización puede cambiar con la distribución que empleemos). Es importante que adaptemos la ruta del directorio a la que usemos en nuestro sistema.

Lo primero que haremos será proteger nuestra aplicación contra usuarios externos mediante una contraseña, de forma que durante el desarrollo no sea posible acceder a ella. La manera más sencilla de hacerlo es mediante autenticación. El protocolo AUTH de HTTP es un estándar que soportan todos los navegadores.

En Mod_Python existen una serie de funciones que Apache llamará en el caso de que existan. El nombre de estas funciones se relaciona directamente con las etapas de procesado de peticiones de HTTP de Apache. En nuestro programa sólo emplearemos dos de estas funciones, *authenhandler* y *handler*, pero el lector puede comprobar el resto en la Tabla [1].

Todos estas funciones, que llamaremos *handlers* a partir de ahora, aceptan un solo parámetro: un objeto de la clase *Request*. Este objeto encapsula la petición y todos los datos que trae consigo. En el Lis-

Listado 1: Código del Programa

```
001 from mod_python import apache
002 from mod_python import util
003 from mod_python import Cookie
004 import sqlite3
005 from Cheetah.Template import
    Template
006 import cgi
007 import time
008 import re
009
010 class DB(object):
011  """Controla la conexión con
    la base de datos smith"""
012
013  def __init__(self):
014   self.db =
    sqlite3.connect("/tmp/enlaces
    .db")
015
016  def getClave(self,nombre):
017   datos = self.sql("select
    clave from usuarios where
    nombre = ?",  (nombre,))
018   return datos[0][0]
019
020  def
    getEnlacesUsuario(self,usuari
    o):
021   datos = self.sql("""select
    usuarios.nombre,
    enlaces.url,enlaces.titulo,
    enlaces.fecha
022                       from
    enlaces join usuarios on
    usuarios.id = enlaces.usuario
023         where usuarios.nombre
    = ?
024         order by
    enlaces.fecha
    desc;""",(usuario,))
025
026   resultado = []
027
028   for fila in datos:
029
    resultado.append({'usuario' :
    fila[0],
030        'url' : fila[1],
031        'titulo' : fila [2],
032        'fecha' : fila [3] })
033
034   return resultado
035
036  def getUsuario(self,id):
037   sql = """select nombre from
    usuarios where id = ?"""
038   cur = self.db.cursor()
039   cur.execute(sql
    ,(int(id),))
040   datos = cur.fetchone()
041   cur.close()
042   return datos
043
044  def
    nuevoEnlace(self,titulo,url,u
    suario = 1):
045   cur = self.db.cursor()
046   cur.execute("insert into
    enlaces
    (titulo,url,usuario,fecha)
    values
    (?,?,?,DATETIME('now'))",
047
    (titulo,url,int(usuario)))
048   self.db.commit()
049   cur.close()
050
051  def
    nuevoUsuario(self,nombre,clav
    e):
052   cur = self.db.cursor()
053   cur.execute("insert into
    usuarios (nombre,clave)
    values (?,?)",
054        (nombre,clave))
055   self.db.commit()
056   cur.close()
057
058  def
    buscarEnlaces(self,cadena):
059   datos = self.sql("""select
    usuarios.nombre,
    enlaces.url,enlaces.titulo,
    enlaces.fecha
060                       from
    enlaces join usuarios on
    usuarios.id = enlaces.usuario
061      where enlaces.titulo
    like ? order by enlaces.fecha
    desc;""",('%'+cadena+'%',))
062
063   resultado = []
064
065   for fila in datos:
066
    resultado.append({'usuario' :
    fila[0],
067        'url' : fila[1],
068        'titulo' : fila [2],
069        'fecha' : fila [3] })
070
071   return resultado
072
073  def getEnlaces(self):
074   datos = self.sql("""select
    usuarios.nombre,
    enlaces.url,enlaces.titulo,
    enlaces.fecha
075   from enlaces join usuarios
    on usuarios.id =
    enlaces.usuario order by
    enlaces.fecha desc limit
    50;""")
076
077   resultado = []
078
079   for fila in datos:
080
    resultado.append({'usuario' :
    fila[0],
081        'url' : fila[1],
082        'titulo' : fila [2],
083        'fecha' : fila [3] })
084
085   return resultado
086
087  def sql(self,cadena,datos =
    {}):
088   """Ejecuta una sentencia
    SQL y devuelve fetchall()."""
089   cur = self.db.cursor()
090   cur.execute(cadena, datos)
091   datos = cur.fetchall()
092   cur.close()
093   return datos
094
095  def
    login(self,nombre,clave):
096   sql = """select id from
    usuarios where nombre = ? and
    clave = ?"""
097   cur = self.db.cursor()
098   cur.execute(sql
    ,(nombre,clave))
099   datos = cur.fetchone()
100   cur.close()
101   return datos[0]
102
103 def authenhandler(req):
104  """Handler de autenticación,
    gestiona el acceso seguro a
    la web."""
105
106  user = req.user
107  clave =
    db.getClave(req.user)
108  pw = req.get_basic_auth_pw()
109
```

Listado 1: Código del Programa

```
110  if pw == clave:
111   return apache.OK
112  else:
113   return
    apache.HTTP_UNAUTHORIZED
114
115 def getId(req):
116  cookies =
    Cookie.get_cookies(req,Cookie
    .MarshalCookie,secret="gusild
    raja")
117  if cookies.has_key('id'):
118   usuario =
    cookies['id'].value
119
120   nombre =
    db.getUsuario(usuario)[0] if
    db.getUsuario(usuario) else
    ""
121
122   return {'id':
    usuario,'nombre': nombre}
123  else:
124   return{'id': -1, 'nombre':
    ''}
125
126 def index(req):
127
128  req.content_type =
    "text/html"
129  t =
    Template(file="/usr/local/www
    /data/templates/index.tmpl")
130  t.registrado = getId(req)
131  t.titulo = "Linux Magazine"
132  t.resultados =
    db.getEnlaces()
133  req.write(str(t))
134
135 def buscar(req):
136  req.content_type =
    "text/html"
137  t =
    Template(file="/usr/local/www
    /data/templates/index.tmpl")
138  t.titulo = "Linux Magazine
    :: Busqueda"
139  t.registrado = getId(req)
140  parametros =
    cgi.parse_qs(req.args)
141
142  if 'q' in parametros:
143   t.resultados =
    db.buscarEnlaces(parametros['
    q'][0])
144  else:
145   t.resultados = []
146
147  req.write(str(t))
148
149 def enlacesUsuario(req):
150  req.content_type =
    "text/html"
151  t =
    Template(file="/usr/local/www
    /data/templates/index.tmpl")
152  usuario =
    cgi.parse_qs(req.args)['q'][0
    ]
153  t.registrado = getId(req)
154  t.titulo = "Linux Magazine
    :: Enlaces del usuario " +
    usuario
155  t.resultados =
    db.getEnlacesUsuario(usuario)
156  req.write(str(t))
157
158 def areIn (elementos,lista):
159  return reduce(lambda x,y: x
    and y, map(lambda x: x in
    lista, elementos))
160
161 def pantallaNuevoEnlace(req):
162  req.content_type =
    "text/html"
163  t =
    Template(file="/usr/local/www
    /data/templates/nuevoEnlace.t
    mpl")
164  t.registrado = getId(req)
165  t.titulo = "Linux Magazine
    :: Nuevo enlace"
166  req.write(str(t))
167
168 def errorUsuario(req):
169  req.content_type =
    "text/html"
170  t =
    Template(file="/usr/local/www
    /data/templates/errorUsuario.
    tmpl")
171  t.titulo = "Linux Magazine
    :: Error de acceso"
172  req.write(str(t))
173
174 def pantallaLogin(req):
175  req.content_type =
    "text/html"
176  t =
    Template(file="/usr/local/www
    /data/templates/login.tmpl")
177  t.titulo = "Linux Magazine
    :: login"
178  req.write(str(t))
179
180 def login(req):
181  if req.method == "POST":
182   parametros =
    cgi.parse_qs(req.read())
183
184   if
    areIn(('nombre','clave'),para
    metros):
185    nombre =
    parametros['nombre'][0]
186    clave =
    parametros['clave'][0]
187
188    id =
    db.login(nombre,clave)
189    if  id:
190
    Cookie.add_cookie(req,Cookie.
    MarshalCookie('id',id,'gusild
    raja'))
191
    util.redirect(req,"index.py")
192    else:
193
    util.redirect(req,"errorUsuar
    io.py")
194   else:
195    pantallaLogin(req)
196
197  else:
198   pantallaLogin(req)
199
200 def logout(req):
201
    Cookie.add_cookie(req,Cookie.
    Cookie('id',"-1"))
202
    util.redirect(req,"index.py")
203
204 def
    pantallaNuevoUsuario(req):
205  req.content_type =
    "text/html"
206  t =
    Template(file="/usr/local/www
    /data/templates/nuevoUsuario.
    tmpl")
207  t.titulo = "Linux Magazine
    :: Crear nuevo usuario"
208  req.write(str(t))
209
210 def nuevoUsuario(req):
211  if req.method == "POST":
```

Listado 1: Código del Programa

```
212   parametros =
    cgi.parse_qs(req.read())
213   if
    areIn((‘nombre’,’clave1’,’cla
    ve2’),parametros):
214
215    nombre =
    parametros[‘nombre’][0]
216    clave =
    parametros[‘clave1’][0]
217    clave2 =
    parametros[‘clave2’][0]
218
219    if clave == clave2:
220
    db.nuevoUsuario(nombre,clave)
221
    util.redirect(req,”login.py”)
222    else:
223
    util.redirect(req,”errorUsuar
    io.py”)
224
225   else:
226    pantallaNuevoUsuario(req)
227  else:
228   pantallaNuevoUsuario(req)
229
230 def nuevoEnlace(req):
231  usuario = getId(req)[‘id’]
232
233  if usuario != ”-1”:
234   if req.args:
235    parametros =
    cgi.parse_qs(req.args)
236    if
    areIn((‘url’,’titulo’),parame
    tros):
237
    db.nuevoEnlace(parametros[‘ti
    tulo’][0],
238
    parametros[‘url’][0],
239            usuario)
240
241
    util.redirect(req,”index.py?u
    su=”+str(usuario))
242    else:
243     pantallaNuevoEnlace(req)
244   else:
245    pantallaNuevoEnlace(req)
246  else:
247
    util.redirect(req,”errorUsuar
    io.py”)
248
249 rutas = [ (“.*/index.py”,
    index),
250    (“.*/buscar.py”,  buscar),
251
    (“.*/enlacesUsuario.py”,enlac
    esUsuario),
252    (“.*/nuevoEnlace.py”,
    nuevoEnlace),
253    (“.*/errorUsuario.py”,
    errorUsuario),
254    (“.*/login.py”, login),
255    (“.*/logout.py”, logout),
256    (“.*/nuevoUsuario.py”,
    nuevoUsuario),
257    ]
258
259 def handler(req):
260  global db
261
262  del(db) # limpiamos la
    conexión con la base de datos
263
264  db = DB()
265
266  for patron,funcion in rutas:
267   if
    re.match(patron,req.uri):
268    funcion(req)
269    return apache.OK
270
271  # Si no se encuentra....
272  index(req)
273  return apache.OK
274
275 ###############################
276 db = DB()
```

tado [1] aparece el código de nuestro programa, y podemos comprobar cómo tanto *handler* como *authenhandler* aceptan un parámetro *req*. Pero ahora concentrémonos en *authenhandler*. Su código es muy sencillo. Cuando Apache encuentra un handler de autenticación como éste, lo primero que hace es presentar al usuario de la aplicación web una ventana en la que le pedirá tanto su nombre de usuario como su clave.

Listado 2: Esquema de la Base de Datos

```
01 CREATE TABLE usuarios (id
   integer primary key, nombre
   varchar, clave varchar);
02 CREATE TABLE enlaces (id
   integer primary key, url
   varchar, titulo varchar,
   usuario integer, fecha
   timestamp);
```

Esta acción se realiza de forma automática, nosotros sólo recogeremos el fruto de esa interacción: el nombre y la clave del usuario. Para ello accedemos a la variable *req.user* del objeto *Request* que aceptamos como parámetro. Para recoger la clave es preciso invocar *get_basico_auth_pw()* antes de comprobar el nombre del usuario, puesto que es esta función la que se encarga de traer ambos datos de forma segura. La ventana que nos mostrará el navegador será parecida a la que se muestra en la Figura 4.

Nuestro handler se encarga de la autenticación sustituyendo a Apache en el proceso. En nuestro caso comprobamos que el nombre del usuario y la clave existen en nuestra base de datos, y si se da el caso, devolvemos el valor *apache.OK*; en caso contrario, *apache.HTTP_UNATHORIZED*. Estos valores son constantes que define Apache, en base a las cuales realizará ciertas acciones. Podemos ver todos los valores posibles en el Recurso [5].

Pasemos a la función *handler*. Cuando el objeto *Request* se pasa a *handler* ya se ha recorrido la mayor parte del proceso de procesado de peticiones. *handler* es el núcleo de la aplicación, puesto que en este punto ya podemos trabajar normalmente. En nuestro caso eliminamos la conexión preexistente a la base de datos si la hubiese y volvemos a conectar para asegurarnos una conexión limpia, después pasamos a delegar el procesado de la petición a otras funciones basándonos en la ruta que se nos pide. Usamos una expresión regular para ver si la ruta se corresponde con alguna de las que controlamos. En caso contrario, pasamos el control a la función por defecto *index()*. De nuevo indicamos a Apa-

che que todo ha ido bien mediante el valor apache.OK.

El esquema de trabajo de las distintas funciones es más o menos el mismo. Se comprueba la presencia de una *cookie* o de unos argumentos esperados, y en base a ambas variables se toman unas acciones, que generalmente consistirán en redirigir la petición a otra función, realizar alguna interacción con la base de datos o pintar una pantalla por defecto.

Procesado de Parámetros

Los parámetros GET se pueden recoger gracias a la variable *req.args*. Pero no todo está solucionado, porque *req.args* nos devuelve la cadena de texto con los parámetros, no los propios parámetros. Podemos emplear la función *cgi.parse_qs()* del módulo *cgi*. Esta función rompe la cadena de parámetros GET en variables y genera un diccionario con el nombre de los parámetros y sus valores.

Los parámetros POST no tienen mayor complicación. En el código de *login(req)* podemos ver cómo se trabaja con ellos. Simplemente comprobamos el método con el que se nos están enviando los parámetros, y en el caso de que sea POST, los leemos usando el método *req.read()*, pasando a procesarlos como con los parámetros GET.

Las Cookies

El procesado de las Cookies se realiza gracias a la clase *Cookie*. El trabajo con ella es bastante simple. Posee dos métodos: *get_cookies()* y *add_cookies()*, que nos permiten recoger y añadir información a una cookie respectivamente. Ambas funciones operan sobre el objeto *req*. Como medida es aconsejable cifrar la información presente en las cookies, para lo cual empleamos la función *Cookie.MarshalCookie*, que nos permite serializar (convertir en un array de bytes) cualquier objeto que se preste a ello, eliminando el problema de decidir cómo representarlo en la cookie, y de paso lo firma empleando *md5* para asegurar la integridad de los datos. Para ello se le pasa un tercer parámetro tanto a *get_cookies()* como a *add_cookies()* llamado *secret* con la clave que emplearemos en la firma digital. El método *get_cookies()* devuelve un diccionario con los datos presentes en la cookie.

En nuestra aplicación pasamos como cookie el parámetro *id*, que almacenará el número que identificará al usuario en cuestión cuando se registre en la aplicación. Para facilitar el trabajo he creado la función *getId()*, que recoge la cookie y extrae de ella el parámetro *id*.

SQlite

Para el almacenado de los datos empleamos la base de datos SQLite, que viene de serie con Python 2.5. Esta base de datos es pequeña y bastante conveniente cuando queremos trabajar sin muchas complicaciones. Se accede a ella a través de la interfaz DBAPI 2.0 de Python que homogeneíza los distintos drivers de acceso a las diferentes bases de datos que existen. El código para la creación de la base de datos aparece en el Listado [2].

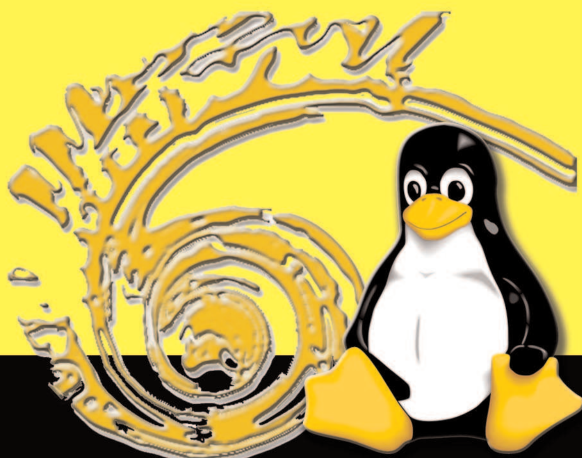
He aislado el código que interactúa con la base de datos, el *Modelo*, en su propia clase de forma que todo acceso sea controlado. Este objeto se comunica con los *controladores* mediante métodos que o bien devuelven una lista de diccionarios con el resultado de una consulta o realizan algún tipo de modificación en la base de datos.

Cheetah

Muy bien, tenemos tanto Modelo (sqlite3) como Controlador (mod_python), ahora necesitamos la Vista. Mod_Python viene de serie con una librería llamada PSP, Python Server Pages, pero la veo algo limitante, puesto que fue diseñada para la generación de HTML. Como queremos tener unas herramientas potentes y versátiles, vamos a utilizar en su lugar *Cheetah*, ver Recurso [4], una librería que permite generar y rellenar plantillas empleando un lenguaje propio. De hecho, Cheetah es muy potente, como tendremos ocasión de comprobar, y podremos utilizarla para generar código fuente, XML o incluso ¡Python!

Cheetah se basa en el concepto de *templates* o plantillas, ver Figura 3. Estas plantillas pueden representar cualquier tipo de fichero texto, dentro del cual existe código *Cheetah*, que genera más texto. Las plantillas son compiladas a ficheros Python que contienen clases con variables y métodos que nos permiten rellenar las plantillas desde Python de forma totalmente transparente. Veamos cómo funciona.

Cualquiera de las funciones «Controladoras» nos puede servir de ejem-



Asociación Linux Español

foros noticias documentación descargas

linux.es

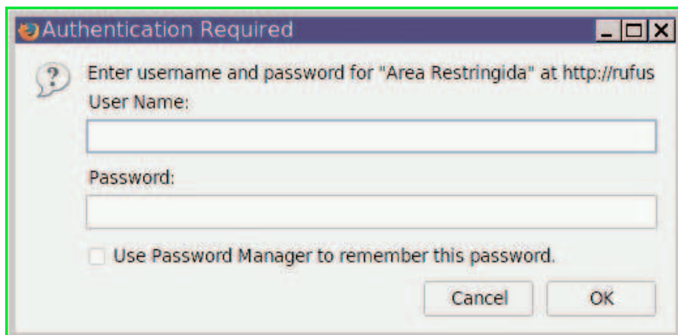


Figura 4: Diálogo de autorización con Auth.

plo, fijémonos en la función *index()*. En ella creamos un objeto de tipo *Template()* al que le pasamos una ruta a un fichero de texto que contiene una plantilla *Cheetah*. Una vez tenemos nuestro objeto *Template* podemos acceder a las variables que hemos establecido en la plantilla como si se tratasen de variables propias del objeto *Template*, por ejemplo accedemos a *t.titulo* para poder cambiar el título. Rellenamos los distintas variables que sean necesarias, y podemos generar el código html mediante, por ejemplo, la función *str()* sobre el objeto *Template*. Pero ¿qué pasa con las variables que hemos rellenado?.

En el momento de generar el código html *Cheetah* interpreta las variables y comienza a ejecutar el código de generación. Tomemos como ejemplo la plantilla *index.tmpl* que vemos en el Listado [2]. Esta plantilla se compone de otras plantillas que añadimos mediante la orden *Cheetah include*. Todas las sentencias de *Cheetah* comienzan con el símbolo «#». En el caso de *include* necesita la ruta del fichero a incluir.

Las variables que sustituimos en el Controlador aparecen ahora aquí con el mismo nombre pero antecedidas por un «$», así *$titulo* representa a *req.titulo*. Pero lo realmente interesante de *Cheetah* es que posee un lenguaje completo. La variable *$registrado*, resultado de la ejecución en el controlador de la función *getId()*, es un diccionario que contiene el *id* del usuario y su *nombre*, y que desde *Cheetah* podemos tratarla como un diccionario de Python. No sólo eso, podemos introducir una sentencia *if* para generar código condicionalmente.

El lenguaje que posee *Cheetah* se parece mucho a Python, pero no es Python. Así, en *Cheetah* los condicionales *#if* acaban con *#end if* en lugar de con la desaparición de una tabulación. En la plantilla *listarEnlaces.tmpl*, ver Listado [3], podemos ver cómo se emplea un bucle *for* para iterar sobre una lista de diccionarios. De esta forma generamos el código HTML que muestra los distintos enlaces que nuestros usuarios han introducido. Si el lector comprende el código Python, entonces no tendrá ningún problema en trabajar con *Cheetah*

Listado 3: index.tmpl

```
01 #include
   ”/usr/local/www/data/templates
   /cabecera.tmpl”
02
03   <h1><a
   href=”index.py”>$titulo</a></h
   1>
04   <form method=”get”
   action=”buscar.py”>
05   <input type=”text” name=”q”
   /><input type=”submit”
   value=”buscar” />
06    [<a
   href=”nuevoEnlace.py”>Nuevo
   Enlace</a>]
07   #if ($registrado[‘id’] !=
   ”-1”  and $registrado[‘id’] !=
   -1)
08   [<a
   href=”logout.py”>Logout</a>
   $registrado[‘nombre’]]
09   #else
10   [<a
   href=”login.py”>Login</a>]
11   [<a
   href=”nuevoUsuario.py”>Crear
   un nuevo usuario</a>]
12   #end if
13   </form>
14
15   #include
   ”/usr/local/www/data/templates
   /listarEnlaces.tmpl”
16
17 #include
   ”/usr/local/www/data/templates
   /pie.tmpl”
```

Listado 4: listarEnlaces.tmpl

```
01 <hr />
02
03   #if $resultados == []
04        <h3>No hay
   resultados</h3>
05   #else
06   <ul>
07   #for $resultado in
   $resultados
08         <li><p><a
   href=”$resultado[‘url’]”>$resu
   ltado[‘titulo’]</a> (<a
   href=”enlacesUsuario.py?q=$res
   ultado[‘usuario’]”>$resultado[
   ‘usuario’]</a>),
   $resultado[‘fecha’]</p></li>
09   #end for
10   </ul>
11   #end if
```

Conclusión

La creación de aplicaciones web siempre se presenta como un proceso complicado que requiere de la ayuda de todo tipo de frameworks. Pero la realidad es que es posible crear aplicaciones web potentes con las herramientas adecuadas y sin tener que recurrir a las sofisticadas librerías que aparecen cada dos por tres. Quizá el canto de sirenas de la Web 2.0 puede desviarnos de nuestro rumbo y hacer que nos perdamos, pero siempre podremos agarrarnos a una roca sólida como es Apache. ■

RECURSOS

[1] Sitio web de Apache: *http://www.apache.org*

[2] mod_python: *http://www.modpython.org*

[3] Pagina de Gregory (Grisha) Trubetskoy, creador de mod_python: *http://www.ispol.com/home/grisha/*

[4] Las plantillas Cheetah: *http://www.cheetahtemplate.org/*

[5] Resumen de Request Handller: *http://www.modpython.org/live/current/doc-html/pyapi-handler.html*