Pyxer 0.5.2

Yet another Python Framework

dirk.holtwick@gmail.com
http://www.pyxer.net

Table of Contents

Introduction

The **Pyxer** Server is a very simple Python Web Framework that aims to makes starting a new project as easy as it can be. It still works respecting the MVC concept but the files can be mixed in one directory. For a high end solution you should consider using Pylons, TurboGears, Django and similar.

This work is inspired by <http://pythonpaste.org/webob/do-it-yourself.html>.

Technical background

The Google App Engine (GAE) in version 1.1 offers a very restricted Python environment and the developer as to ship arround a lot of problems. Pyxer helps on this point by also providing solutions that work with the great WSGI Framework Paster by Ian Bicking. So you get the best from both sides: GAE and Paster. To achieve this some other great third party tools are used like WebOb and VirtualEnv (also by Ian Bicking) and for templating the html5lib is used.

Installation

Install Pyxer using easy_install:

```
$ easy_install pyxer
```

All required packages (webob, html5lib, beaker) should be installed automatically if needed.

If you want to use Google AppEngine install it separately too.

Quick start

Create a new project

At first set up a new Pyxer project using the Pyxer command line tool like this:

```
$  pyxer init myexample
```

In the newly created directory "myexample" you will find a directory structure like this one (under Windows "bin" will be called "Scripts"):

```
bin/
public/
lib/
```

Place your files in the "public" directory.

Start the server

To start the server you may choose between the Paster-Engine:

```
$ xpaster serve
```

or the GAE-Engine:

```
$ xgae serve
```

Or use Pyxer command line tool again to use the default engine (which is WSGI from Python standard lib):

```
$ pyxer serve
```

But you may also use Pyxer without using the command line tools e.g. like this:

```
$ paster serve development.ini
```

"Hello World"

For a simple "Hello World" just put an "index.html" file into the "public" directory with the following content:

```
Hello World
```

This works just like a static server. To use a controller put a file "__init__.py" into that directory with the following content:

```
@controller
def index():
    return "Hello World"
```

About controllers

Controller, templates and static files are placed in the same directory (usually "public"). First **Pyxer** looks for a matching controller. A controller is defined in the "__init__.py" file and decorated by using `@controller` which is defined in `pyxer.base`.

```
from pyxer.base import *

@controller
def index():
    return "Hello World"
```

Using templates

This example can be called like `/` or `/index`. To use a Pyxer template with this file you may use the `render()` function or just return `None` (that is the same as not returning anything) and the matching template will be used, in this case `index.html`. The available object in the template are the same as used by Pylons: `c` = context, `g` = globals and `h` = helpers.

```
from pyxer.base import *

@controller
def index():
    c.title = "Hello World"
```

By default the Kid templating language is used and output is specified as "xhtml-strict". You may want to change that for certain documents e.g. to render a plain text:

```
from pyxer.base import *

@controller(output="plain")
def index():
    c.title = "Hello World"
```

Or use another template:

```
from pyxer.base import *

@controller(template="test.html", output="html")
def index():
    c.title = "Hello World"
```

Or use your own renderer:

```
from pyxer.base import *

def myrender():
    result = request.result
    return "The result is: " + repr(result)

@controller(render=myrender)
def index():
    return 9 + 9
```

Using JSON

To return data as JSON just return a dict or list object from your controller:

```
from pyxer.base import *

@controller
def index():
   return dict(success=True, msg="Everything ok")
```

Using sessions

Session are realized using the Beaker package. You may use the variable `session` to set and get values. To store the session data use `session.save()`. Here is a simple example of a counter:

```
from pyxer.base import *

@controller
def index():
    c.ctr = session.get("ctr", 0)
    session["ctr"] = c.ctr + 1
    session.save()
```

??? XXX

1. #Looks for a controller (foo/bar/__init__py:bar)
    1. If the controller returns a dictionary this will be applied to template (step 2)
2. Looks for the template (foo/bar.html)

Publish to GAE

To publish your project to GAE you may also use the Pyxer command line tool. First check if your "app.yaml" file contains the right informations like the project name and version infos. Then just do like this:

```
$ xgae upload
```

Be aware that Pyxer first needs to fix the paths to be relative instead of absolute to make them work on the GAE environment. If you choose not to use Pyxer for uploading you have to do this fix up explicitly **before** you upload your applikation, like this:

```
$ xgae fix
```

Templating

XXX Kid or how to configure

DEPRECATED:

Pyxer offers yet another templating language that is very close to Genshi and Kid. As the former did not work with Google AppEngine at the moment of birth of **Pyxer** the new templating tools had been implemented. It is based on html5lib.

Variables and expressions

The default templating works similar to most known other templating languages. Variables and expressions are realized like `$<varname>` (where `<varname>` may contain dots!) and `${<expression>}`:

```
Hello ${name.capitalize()}, you won $price.
$item.amount times $item.name.
```

Commands

Commands are the same as used by Python. The only difference is, that you have to tell that a block ends by calling the command `end` (not needed for `else` and `elif`):

```
<% def address(name): %>
  ABCDEF Company
  <% if name: %>
    Your contact: $name
  <% else: %>
    Unknown contact!
  <% end %>
<% end %>
<% for name in ['Tom', 'Fred']: %>
  ${ address(name) }
<% end %>
```

Includes and extends

You can insert content of other templates by using “include”:

```
<% include(filename) %>
```

More sophisticated is the extension of a file.

```
<% extends('b.html') %>
<% def content(): %>
  Here goes the content
<% end %>
```

File “a.html”

```
<% content() %>
```

File “b.html”

HTML Attriutes

These are also known form many templating languages like Genshi. They are used like this:

```
<div py:if=”name.startswith('tom ')”>Welcome $name</div>
```

xxx

```
<div py:for=”name in sorted(c.listOfNames)”>Welcome $name</div>
```

xxx

Advanced

Python virtual environment

To make deployment of GAE projects easy a virtual environment (VM) is created. If you start GAE via `xgae` or paster via `xpaster` these virtual environments will automatically be used. Pyxer determines the root of the VM by looking for the `app.yaml` file. If you have to enter the VM for installing packages or for other reasons you may to it like this:

```
$ pyxer vm
(vm)$ easy_install html5lib
(vm)$ exit
```

You may also use the usual functions as described in virtualenv by Ian Bicking <http://pypi.python.org/pypi/virtualenv/>.

```
$ Scripts\activate.bat
$ easy_install SomePackageName
$ deactivate
```

And for other Unix like system like this:

```
$ source bin/activate
$ easy_install SomePackageName
$ deactivate
```

Development of Pyxer under GAE

If you decide to develop Pyxer you may run into the following problem: each project comes with an own virtual machine (VM)and its own installation of Pyxer in it. So if you change the development version it will have no effect on your installation. Therefore a command "pyxer" is added that synchronizes the Pyxer installation in the VM with the development version:

```
$ pyxer pyxer
```

BTW: To install the development version using SetupTools do like this:

```
$ cd <Path_to_development_version_of_Pyxer>
$ python setup.py develop
```

You will have to repeat this each time the version of Pyxer changes, because otherwise the command line tools do not work.

Writing test cases

Since a Pyxer project is based in Paster writing test cases is quite the same. The most simple test looks like this. (We asume that the test file to be placed in the root of the project. For normal testing you have do add root to `sys.path` and modify the `loadapp` argument.):

```
from paste.deploy import loadapp
from paste.fixture import TestApp
import os.path

app = TestApp(loadapp('config:%s' % os.path.abspath('development.ini')))
res = app.get("/")
assert ('<body' in res)
```

For more informations look here <http://pythonpaste.org/testing-applications.html>.

Use within Eclipse

xxx

Use Google App Engine Launcher on Mac OS

xxx

Engines

XXX

Pyxer uses support different so called "engines" to publish a project. Most of them need own configurations and a well prepare environment to work fine. These are very specific to each of these engines and **Pyxer** tries to make the setup as easy as possible

Paster

```
$ xpaster serve --reload
```

Google Appengine

```
$ xgae serve
```

Links

Some usfull links:

1. XXX