Drink! Documentation

Release 0.0.10

Fabien Devaux

Sep 27, 2017

Contents

Note: alpha software !

Drink looks like a lightweight & easy to install Web CMS. Under the ground this is only a simple test application for an in-progress rapid web framework for applications that have standard requirements.

CHAPTER 1

Short description

With drink, without coding power, you get a simple to use & install CMS, a with nice javascript interface.

With python & javascript skill, you get a very simple API to develop your own website, with no SQL requirements and “builtin” objects storage, automatic but overridable edition & rendering of objects, etc... it comes with a bunch of useful classes you can hack and fork.

CHAPTER 2

Technical description

100% written in python, WSGI compliant, exposing an object database (think about a tree of dict-derived classes).

On top of that, you have simple generic templates you can override if you prefer (instead of writing python). Then, you profite from an important layer of javascript, using some of the client power to reduce bandwith usage. Json and plain-text are used for communication.

Quickstart

Pre-requisites

You should be able to execute a terminal emulator (like `cmd.exe`, `xterm`, etc...) and have notions with python. Even if the steps are detailed, some experience with virtualenv may help.

If you skip the virtual environment part, it might require additional work on permissions. Feel free to use virtualenv.

Installing

It is recommanded to use a virtualenv for your drink installation:

```
% virtualenv --no-site-packages --distribute drink_env
% . drink_env/bin/activate
```

Then, you can install drink, either from *PyPi* (`easy_install drink`) or from sources:

```
% wget -O master.zip https://github.com/fdev31/drink/zipball/master
% unzip master.zip
% cd fdev31-drink-*
% python setup.py install
```

Creating a project

Do not forget to activate your virtual environment:

```
% . drink_env/bin/activate
```

And chose a folder of your own, that can hold several projects and run `drink make`:

```
% cd ~/web_apps/
% drink make
Project folder: test_project
Additional python package with drink objects
(can contain dots): test_extensions
Ip to use (just ENTER to allow all):
HTTP port number (ex: "80"), by default 0.0.0.0:5000 will be used:
Objects to activate:
a gtd-like tasklist - tasks : y
a wiki-like web page in markdown format - markdown : y
a tool to find objects in database - finder : y
a filesystem proxy, allow sharing of arbitrary folder or static websites - filesystem
↪: y
Additional root item name (just ENTER to finish):
Project created successfully.

You can now go into the /home/fab/web_apps/test_project folder and run

- drink db (to start the database daemon)
- drink run (to start the web server)

If you run with DEBUG=1 in environment, templates and python code should reload
↪automatically when changed.
For static files changes, no restart is needed.
```

Running

Go into the project folder (same name as you answered), and type `drink start`:

```
% cd ~/web_apps/test_project
% drink start
```

Exploring

Debug mode

Running drink in debug mode just consists in setting the `DEBUG` environment variable to `1`. In this mode, you gain some features:

- Python code reloads automatically (with a bit of luck...)
- HTML Templates are reloaded on every access
- You have a flood of messages about everything happening
- You get a nice interactive debugger in case of programming errors

This can be set in two ways, by editing the `drink.ini` file and using **debug** as the server.backend, as shown by this snippet:

```
[server]
backend = debug
```

You can also set that behavior directly from the command line:

```
% DEBUG=1 drink start
```

Important files

The database configurations is hold by `database/zeo.conf`, you might want to read ZEO & Zope3 docs in google to know in what way you can customize this, but the defaults should be fine.

Action's icons are in `static/actions/` folder, you can also replace `static/page.css` file if you want a custom CSS.

The `templates/main.html` file can be easily customized as well, but this folder is mainly interesting to add your own custom templates (`main.html` should be a good base).

An empty python module is also created automatically to fit the information you gave at *make* time. Feel free to add your objects inside it, if you add a file `blog_application` to that folder, you need to activate it in the `drink.ini` file:

[objects] markdown= blog_application=

Developpers' documentation

Routing

See drink as a tree of items stored in a database. Since the database is structured (a single trunk, then branches, more branches etc... until "leafs", the children-most items), it is easy to map it as HTTP URLs this way: http://domain.com/trunk/branch/sub-branch/sub-sub-branch/leaf

In our case, the trunk is "/" and the first "public" branch *id* is "pages". This way, most URLs will start with "/pages/". The final "/" indicates we want to access some item (or object or element, call it whatever you prefer). If there is no final *slash* at the end of the URL, then it's a item action or property (ex: *view*, *edit*, *list*, etc... or *struct*, *actions*, etc...).

This way, if you add an item called "My Calendar" under "Pages", the automatically-crafted url will be `/pages/my-calendar/`, if you want to link directly to its *edit* form, then use `/pages/my-calendar/edit`. If you experiment some troubles, you probably made in inconsistant usage of trailing slash.

Database access

Objects can be safely retrieved (with permission checks) using `drink.get_object()`

Rendering

All objects inheriting `drink.Page` should implement a *view* method with the same prototype as the default function which is `drink.default_view()`.

In case you need a simple but correct handling of unauthorized accesses, it is recommanded to return the value of `drink.unauthorized()` call in your handler. It will either show an error message or an authentication screen.

Text conversion

In case you have random kind of input (url encoded, latin string, utf-8 str, unicode) and want to ensure you can work with it as unicode data then you may use `drink.omni()`.

To render file sizes as readable strings for humans, just use `drink.bytes2human()`.

Adding upload capabilities

Creating a new uploadable type

First, your page type you contain a properly filled `classes` property and the user must have permissions to add content to the desired `Page`.

Then, your object must be registered using `drink.add_upload_handler()`.

API index

Base classes

`Page`
`ListPage`

Base functions & data

`make_app`
`classes`
`get_object`
`omni`
`default_view`
`add_upload_handler`
`request`
`response`
`bytes2human`
`db`
`unauthorized`
`init`

Database backends

`zdb`
`dumbdb`

WSGI Building block

`make_app`

Properties types

types	Define names for all type symbols known in the standard interpreter.
objects.markdown.MarkdownEditor	

Builtin object types

generic.WebFile
generic.Settings
markdown.MarkdownPage
tasks.TODO
tasks.TODOList
users.User
users.UserList
users.Group
users.GroupList
finder.ObjectBrowser
filesystem.Filesystem
sonic.SonicHome

CHAPTER 3

Contact

Email

Use my email for bug-reporting, urgent requests, support, or anything. My contact is fdev31 <AT> gmail <DOT> com.

Bug reporting

Use Bitbucket's Bugtracker.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search