
zunzuncito Documentation

Release 0.1.20

Nicolas Embriz

August 31, 2015

1	Table of Contents	3
1.1	Zunzuncito	3
1.2	Download	4
1.3	Install	4
1.4	Quick Start	4
1.5	ZunZun class	8
1.6	Request class	18
1.7	Response class	19
1.8	API Resource	20
1.9	_catchall resource	26
1.10	tools	29
1.11	HTTP status codes	33
1.12	zunzuncito Package	35
1.13	WebOb	35
1.14	Jinja2	36
1.15	gevent	38
1.16	Changelog	40
1.17	Issues	41
1.18	FAQ	41

micro-framework for creating REST API's.

Table of Contents

1.1 Zunzuncito

micro-framework for creating REST API's.

1.1.1 Design Goals

- Keep it simple and small, avoiding extra complexity at all cost. [KISS](#)
- Create routes on the fly or by defining regular expressions.
- Support API versions out of the box without altering routes.
- Thread safety.
- Via decorator or in a defined route, accept only certain [HTTP methods](#).
- Follow the single responsibility [principle](#).
- Be compatible with any WSGI server, example: [uWSGI](#), [Gunicorn](#), [Twisted](#), etc.
- Tracing Request-ID “rid” per request.
- Compatibility with Google App Engine. [demo](#)
- [Multi-tenant Support](#).
- Ability to create almost anything easy, example: Support [chunked transfer encoding](#).

1.1.2 What & Why ZunZuncito

ZunZuncito is a [python](#) package that allows to create and maintain [REST API's](#) without hassle.

The simplicity for sketching and debugging helps to develop very fast; versioning is inherit by default, which allows to serve and maintain existing applications, while working in new releases without need to create separate instances. All the applications are WSGI [PEP 333](#) compliant, allowing to migrate existing code to more robust frameworks, without need to modify the existing code.

Why ?

“The need to upload large files by chunks and support resumable uploads trying to accomplish something like the [nginx upload module](#) does in pure python.”

The idea of creating ZunZuncito, was the need of a very small and light tool (batteries included), that could help to create and deploy REST API's quickly, without forcing the developers to learn or follow a complex flow but, in contrast, from the very beginning, guide them to properly structure their API, giving special attention to “versioned URI's”, having with this a solid base that allows to work in different versions within a single ZunZun instance without interrupting service of any existing API resources.

1.2 Download

Zip File: <https://github.com/nbari/zunzuncito/zipball/master>

Tar Ball: <https://github.com/nbari/zunzuncito/tarball/master>

View on GitHub: <https://github.com/nbari/zunzuncito>

See also:

Install

1.3 Install

The easy way is by using `pip`:

```
$ pip install zunzuncito
```

If you don't have `pip`, after downloading the sources, you can run:

```
$ python setup.py install
```

Clone the repository:

```
$ git clone https://github.com/nbari/zunzuncito.git
```

1.4 Quick Start

URI parts:

```
http://api.zunzun.io/v0/get/client/ip
|
|_____|\_|_|_|_|_|_|_|_|_|_|_|
|           | | | \_|_|_|_|_|_|_|_|
|           |   version | | | |
|   host (default)   resource |path|
|                           |
|                           path[0] |
|                           |
|                           path[1]
```

ZunZun translates that URI to:

```
my_api.default.v0.zun_get.zun_client.zun_client
```

This is the directory structure:

API directory structure**app.py** application python file.**my_api** The **root** directory of the API.**default** The **vroot** directory.**v0** Directory for default API resources or for version **0** when specified.**v1** Directory for API resources version **1****get** API resource**client** Path

```

1 /home/
2   |--zunzun/
3     |--app.py
4     |--my_api
5       |--__init__.py
6       |--default
7         |--__init__.py
8         |--v0
9           |--__init__.py
10          |--zun_default
11            |--__init__.py
12            |--zun_default.py
13            |--zun_get
14            |--__init__.py
15            |--zun_get.py
16            |--zun_client
17              |--__init__.py
18              |--zun_client.py
19            |--zun_hasher
20              |--__init__.py
21              |--zun_hasher.py
22          |--v1
23            |--__init__.py
24            |--zun_default
25              |--__init__.py
26              |--zun_default.py
27            |--zun_hasher
28              |--__init__.py
29              |--zun_hasher.py

```

Inside directory `/home/zunzun` there is a file called **app.py** and a directory **my_api**.

For a very basic API, contents of file **app.py** can be:

```

1 import zunzuncito
2
3 root = 'my_api'
4
5 versions = ['v0', 'v1']
6
7 hosts = {'*': 'default'}
8
9 routes = {'default':[
10     ('/(md5|sha1|sha256|sha512) (/.*)?', 'hasher', 'GET, POST'),
11     ('/.*', 'default')
12 ]}
13

```

```
14 app = zunzuncito.ZunZun(root, versions, hosts, routes)
15
16 # For appending the Request-ID header on GAE
17 # app = zunzuncito.ZunZun(root, versions, hosts, routes, rid='REQUEST_LOG_ID')
```

- line 3 defines the “document root” for your API
- line 7 gives multitenant support, in the example all “*” is going to be handled by the ‘default’ **vroot**
- line 11 contains a regex matching all the requests, it is at the bottom because in the routes, order matters.

The contents of the **my_api** contain python modules (API Resources) for example the content of module `zun_default/zun_default.py` is:

```
1 from zunzuncito import tools
2
3
4 class APIResource(object):
5
6     @tools.allow_methods('get, head')
7     def dispatch(self, request, response):
8
9         request.log.debug(tools.log_json({
10             'API': request.version,
11             'URI': request.URI,
12             'method': request.method,
13             'vroot': request.vroot
14         }, True))
15
16         data = {}
17         data['about'] = ("Hi %s, I am zunzuncito a micro-framework for creating"
18             " REST API's, you can read more about me in: "
19             "www.zunzun.io") % request.environ.get('REMOTE_ADDR', 0)
20
21         data['Request-ID'] = request.request_id
22         data['URI'] = request.URI
23         data['Method'] = request.method
24
25         return tools.log_json(data, 4)
```

See also:

[Basic template](#)

1.4.1 How to run it

Zunzuncito is compatible with any WSGI server, next are some examples of how to run it with `uWSGI`, and `Gunicorn`, `Twisted`.

uWSGI

Listening on port 8080:

```
uwsgi --http :8080 --wsgi-file app.py --callable app --master
```

Listening on port 80 with 2 processes and stats on `http://127.0.0.1:8181`:

```
uwsgi --http :80 --wsgi-file app.py --callable app --master --processes 2 --threads 2 --stats 127.0.0.1
```

Using a .ini file

TRACK_ID

route-run adds a custom tracking ID, see [uwsgi InternalRouting](#)

```
1 [uwsgi]
2 http = :8080
3 route-run = addvar:TRACK_ID=${uwsgi[uuid]}
4 route-run = log:TRACK_ID = ${TRACK_ID}
5 master = true
6 processes = 2
7 threads = 1
8 stats = 127.0.0.1:8181
9 harakiri = 30
10 wsgi-file = app.py
11 callable = app
```

For this case, to append to all your responses the **Request-ID** header run the app like this:

```
app = zunzuncito.ZunZun(root, versions, hosts, routes, rid='TRACK_ID')
```

Gunicorn

Listening on port 8080:

```
gunicorn -b :8080 app:app
```

Listening on port 8080 with 2 processes:

```
gunicorn -b :8080 -w2 app:app
```

1.4.2 GAE

To have a ZunZun instance up and running in Google App Engine you can use the following configuration.

Contents of the **app.yaml** file:

main.app

script main is the main.py file **app** is the instance of zunzun

```
1 application: <your-GAE-application-id>
2 version: 1
3 runtime: python27
4 api_version: 1
5 threadsafe: yes
6
7 handlers:
8 - url: /favicon\.ico
9   static_files: favicon.ico
10  upload: favicon\.ico
```

```
11
12 - url: /*
13   script: main.app
```

Note: When using GAE the global unique identifier per request is: [REQUEST_LOG_ID](#)

For this case, to append to all your responses the **Request-ID** header run the app like this:

```
app = zunzuncito.ZunZun(root, versions, hosts, routes, rid='REQUEST_LOG_ID')
```

1.5 ZunZun class

ZunZun is the name of the `class` that will parse all the incoming request and route them to a proper `APIResource` class to process the requests.

```
1 import zunzuncito
2
3 root = 'my_api'
4
5 versions = ['v0', 'v1']
6
7 hosts = {'*': 'default'}
8
9 routes = {'default':[
10     ('/(md5|sha1|sha256|sha512)(/.*)?', 'hasher', 'GET, POST'),
11     ('/.*', 'default')
12 ]}
13
14 app = zunzuncito.ZunZun(root, versions, hosts, routes, rid='TRACK_ID', debug=True)
```

1.5.1 Root

The `root` argument is the name of the directory containing all your sources.

```
1 import zunzuncito
2
3 root = 'my_api'
4
5 versions = ['v0', 'v1']
6
7 hosts = {'*': 'default'}
8
9 routes = {'default':[
10     ('/(md5|sha1|sha256|sha512)(/.*)?', 'hasher', 'GET, POST'),
11     ('/(.*\.(gif|png|jpg|ico|bmp|css|otf|eot|svg|ttf|woff))', 'static')
12 ]}
13
14 app = zunzuncito.ZunZun(root, versions, hosts, routes)
```

Making an analogy, you can see `root` as the `DocumentRoot` of the application.

```
1 /home/
2   |--zunzun/
3     |--app.py
```

```

4  |--my_api
5      |--__init__.py
6      |--default
7          |--__init__.py
8          |--v0
9              |--__init__.py
10             |--zun_default
11             | |--__init__.py
12             |--zun_default.py
13             |--zun_hasher
14             |--__init__.py
15             |--zun_hasher.py
16         |--v1
17             |--__init__.py
18             |--zun_default
19             | |--__init__.py
20             |--zun_default.py
21             |--zun_hasher
22             |--__init__.py
23             |--zun_hasher.py

```

- In this case the `my_api` directory, is the root

1.5.2 Versions

The `versions` argument must be a list of names representing the available API versions.

```

1  import zunzuncito
2
3  root = 'my_api'
4
5  versions = ['v0', 'v1']
6
7  hosts = {'*': 'default'}
8
9  routes = {'default':[
10     ('/my/?.*', 'ip_tools', 'GET'),
11     ('/(md5|sha1|sha256|sha512) (/.*)?', 'hasher', 'GET, POST')
12 ]}
13
14 app = zunzuncito.ZunZun(root, versions, hosts, routes)

```

Request example

When no version is specified on the URI request, the default version is the first element off the list, example:

```
curl -i http://api.zunzun.io/my/ip
```

Or:

```
curl -i http://api.zunzun.io/v0/my/ip
```

The output could be something like:

```

1  HTTP/1.1 200 OK
2  Request-ID: 52ad9da400ff0dda875ef62f7d0001737e7a756e7a756e6369746f2d617069000131000100
3  Content-Type: application/json; charset=UTF-8

```

```
4 Vary: Accept-Encoding
5 Date: Sun, 15 Dec 2013 12:16:37 GMT
6 Server: Google Frontend
7 Cache-Control: private
8 Alternate-Protocol: 80:quic,80:quic
9 Transfer-Encoding: chunked
10
11 {
12     "ip": "89.181.199.57"
13 }
```

Now if we change the version, notice the v1:

```
curl -i http://api.zunzun.io/v1/my/ip
```

The output could be something like:

```
1 HTTP/1.1 200 OK
2 Request-ID: 52ada62f00ff06ee2d1086b0d00001737e7a7a756e7a756e6369746f2d617069000131000100
3 Content-Type: application/json; charset=UTF-8
4 Vary: Accept-Encoding
5 Date: Sun, 15 Dec 2013 12:53:03 GMT
6 Server: Google Frontend
7 Cache-Control: private
8 Alternate-Protocol: 80:quic,80:quic
9 Transfer-Encoding: chunked
10
11 {
12     "inet_ntoa": 1505085241,
13     "ip": "89.181.199.57"
14 }
```

How it works internally

The API directory structre looks like:

```
1 /home/
2   |--zunzun/
3     |--app.py
4     |--my_api
5       |--__init__.py
6       |--default
7         |--__init__.py
8         |--v0
9           |--__init__.py
10          |--zun_default
11            |--__init__.py
12            |--zun_default.py
13            |--zun_ip_tools
14              |--__init__.py
15              |--zun_ip_tools.py
16            |--v1
17              |--__init__.py
18              |--zun_default
19                |--__init__.py
20                |--zun_default.py
21              |--zun_ip_tools
```

```

22 |--__init__.py
23 |--zun_ip_tools.py

```

The directories `v0` and `v1` have the same structure, but the contents of the `.py` scripts change.

In this example, `v0` returns the IP of the request, while `v1` besides returning the IP it also returns the `inet_atom`

1.5.3 Hosts

The `hosts` argument contains a dictionary of domains and roots.

A very basic API, contents of file `app.py` can be:

Hosts dictionary elements

```

* wildcard matching all HTTP_HOSTS
default root

```

```

1 import zunzuncito
2
3 root = 'my_api'
4
5 versions = ['v0', 'v1']
6
7 hosts = {'*': 'default'}
8
9 routes = {'default':[
10     ('/my/?.*', 'ip_tools', 'GET')
11 ]}
12
13 app = zunzuncito.ZunZun(root, versions, hosts, routes, debug=True)

```

To support [multi-tenancy](#) the `hosts` dictionary is needed.

A [dictionary structure](#) is formed by **key: value** elements, in this case the key is used for specifying the ‘host’ and the value to specify the **vroot**

Hosts structure & vroot

The wildcard character `*` can be used, for example:

```

1 hosts = {
2     '*': 'default',
3     '*.zunzun.io': 'default',
4     'ejemplo.org': 'ejemplo_org',
5     'api.ejemplo.org': 'api_ejemplo_org'
6 }

```

- line 2 matches any host `*` and will be served on vroot `‘default’`
- line 3 matches any host ending with `zunzun.io` and will be served on vroot `‘default’`
- line 4 matches host `ejemplo.org` and will be server on vroot `‘ejemplo_org’`
- line 5 matches host `api.ejemplo.org` and will be served on vroot `‘api_ejemplo_org’`

Notice that the vroot values use `_` as separator instead of a dot, this is to prevent conflicts on how python read files. for example this request:

```
http://api.ejemplo.org/v0/gevent
```

Internally will be calling something like:

```
import my_api.api_ejemplo_org.v0.zun_gevent.zun_gevent
```

Directory structure

The API directory structure for this example would be:

virtual roots (vroot)

default *line 2, 3*

ejemplo_org *line 4*

api_ejemplo_org *line 5*

```
1 /home/
2  |--zunzun/
3    |--app.py
4    |--my_api
5      |--__init__.py
6      |--default
7        |--__init__.py
8        |-- `--v0
9          |--__init__.py
10         |-- `--zun_default
11           |--__init__.py
12           |-- `--zun_default.py
13 |--ejemplo_org
14   |--__init__.py
15   |-- `--v0
16     |--__init__.py
17     |-- `--zun_default
18       |--__init__.py
19       |-- `--zun_default.py
20 |--api_ejemplo_org
21   |--__init__.py
22   |-- `--v0
23     |--__init__.py
24     |--zun_gevent
25     |-- |--__init__.py
26     |-- |-- `--zun_gevent.py
27     |-- `--zun_default
28     |-- |--__init__.py
29     |-- |-- `--zun_default.py
```

1.5.4 Routes

The `routes` argument must be a dictionary containing defined routes per **vroot**.

```
1 import zunzuncito
2
```



```

3 root = 'my_api'
4
5 versions = ['v0', 'v1']
6
7 hosts = {
8     '*': 'default',
9     'domain.tld': 'default',
10    '*.domain.tld': 'default',
11    'beta.domain.tld': 'beta'
12 }
13
14 routes = {'default':[
15     ('/(md5|sha1|sha256|sha512)(/.*)?', 'hasher', 'GET, POST')
16 ], 'beta':[
17     ('/upload/?.*', 'upload', 'PUT, POST'),
18     ('/.*', 'default')
19 ]}
20
21 app = zunuzcito.ZunZun(root, versions, hosts, routes, debug=True)

```

Note: By default, if no **routes** specified, the requests are handled by matching the URI request with an valid **API Resource**, you only need to specify **routes** if want to handle different URI requests with a single **API Resource**

Example

The request:

```
http://api.zunuz.io/v0/env
```

Will be handled by the python custom module `zun_env/zun_env.py`

But all the following GET requests:

- `http://api.zunuz.io/v0/md5/frebsd`
- `http://api.zunuz.io/v0/sha1/frebsd`
- `http://api.zunuz.io/v0/sha256/frebsd`
- `http://api.zunuz.io/v0/sha512/frebsd`

And also this POST requests:

```

curl -i -X POST http://api.zunuz.io/v0/md5 -d 'frebsd'

curl -i -X POST http://api.zunuz.io/v0/sha1 -d 'frebsd'

curl -i -X POST http://api.zunuz.io/v0/sha256 -d 'frebsd'

curl -i -X POST http://api.zunuz.io/v0/sha512 -d 'frebsd'

```

Will be handled by the pythom custom module `zun_hasher/zun_hahser.py`, this is because a specified route:

```
('/(md5|sha1|sha256|sha512)(/.*)?', 'hasher', 'GET, POST')
```

You can totally omit routes and handle all by following the API *directory structure*, this can give you more fine control over you API, for example in the previous example you could create modules for every hash algorithm, and have independent modules like:

```
1  `--v0
2     |--__init__.py
3     |--zun_md5
4         |--__init__.py
5         |--zun_md5.py
6     |--zun_shal
7         |--__init__.py
8         |--zun_shal.py
9     |--zun_sha256
10         |--__init__.py
11         |--zun_sha256.py
12     |--zun_sha512
13         |--__init__.py
14         |--zun_sha512.py
```

Note: Defining routes or using the directory structure is a design choice, some times having all in one module can be easy to maintain, while other times having a module for each specific task, would be preferred.

See also:

The `zun_` prefix

The flow

When a new request arrive, the ZunZun router searches for a `vroot` declared on the hosts dictionary matching the current `HTTP_HOST`.

Once a `vroot` is found, the ZunZun router parses the `REQUEST_URI` in order to accomplish this pattern:

```
/version/api_resource/path
```

The router first analyses the URI and determines if it is versioned or not by finding a match with the current specified versions in case no one is found, fallback to the default which is always the first item on the versions list in case one provided, or `v0`.

After this process, the `REQUEST_URI` becomes a list of resources - something like:

```
['version', 'api_resource', 'path']

# for http://api.zunzun.io/v0/env
['v0', 'env']

# for http://api.zunzun.io/v0/sha256/freebsd
['v0', 'sha256', 'freebsd']
```

The second step on the router is to find a match within the `routes` dictionary and the local modules.

In case a list of `routes` is passed as an argument to the ZunZun instance, the router will try to match the `API_resource` with the items of the `routes` dictionary. If no matches are found it will try to find the module in the root directory.

Routes dictionary structure

In the above example, the `routes` dictionary contains:

root	regular expression	API Resource	HTTP methods
default	/(/md5 sha1 sha256 sha512)(/.*)?	hasher	'GET, POST'
beta	/upload/?.*	upload	'PUT, POST'
beta	/.*	default	

Translating the table to code:

```

1 routes = {}
2 routes['default'] = [
3     ('(/md5|sha1|sha256|sha512)(/.*)?', 'hasher', 'GET, POST')
4 ]
5 routes['beta'] = [
6     ('/upload/?.*', 'upload', 'PUT, POST'),
7     ('/.*', 'default')
8 ]

```

Warning: Regular expressions have priority, for example the regex `(/.*)` will **catch-all** the request, that's why in our example is the last regex, since order is important.

Directory structure

The API directory structure for the examples presented here is:

API directory structure

```

default root directory
beta root directory

```

```

1 /home/
2   |--zunzun/
3     |--app.py
4     |--my_api
5       |--__init__.py
6       |--default
7         |--__init__.py
8         |--v0
9           |--__init__.py
10          |--zun_default
11            |--__init__.py
12            |--zun_default.py
13            |--zun_env
14              |--__init__.py
15              |--zun_env.py
16              |--zun_hasher
17                |--__init__.py
18                |--zun_hasher.py
19            |--v1
20              |--__init__.py
21              |--zun_default
22                |--__init__.py
23                |--zun_default.py
24              |--zun_hasher
25                |--__init__.py
26                |--zun_hasher.py
27            |--beta

```

```

28 |--__init__.py
29 `--v0
30 |   |--__init__.py
31 |   |--zun_default
32 |   |   |--__init__.py
33 |   |   `--zun_default.py
34 |   `--zun_upload
35 |       |--__init__.py
36 |       `--zun_upload.py

```

1.5.5 Prefix

The prefix argument is the string that should be appended to all the names of the python modules.

```

1 import zunzuncito
2
3 root = 'my_api'
4
5 versions = ['v0', 'v1']
6
7 hosts = {'*': 'default'}
8
9 routes = {'default':[
10     ('/(md5|sha1|sha256|sha512)(/.*)?', 'hasher', 'GET, POST')
11 ]}
12
13 app = zunzuncito.ZunZun(root, versions, hosts, routes, prefix='zzz_')

```

Note: The default prefix is zun_

Directory structure

The directory containing the sources for the application would look like:

```

1 /home/
2   |--zunzun/
3   |   |--app.py
4   |   |--my_api
5   |   |   |--__init__.py
6   |   |   |--default
7   |   |   |   |--__init__.py
8   |   |   |   |--v0
9   |   |   |   |   |--__init__.py
10  |   |   |   |   |--zzz_default
11  |   |   |   |   |   |--__init__.py
12  |   |   |   |   |   `--zzz_default.py
13  |   |   |   |   `--zzz_hasher
14  |   |   |   |       |--__init__.py
15  |   |   |   |       `--zzz_hasher.py
16  |   |   `--v1
17  |   |       |--__init__.py
18  |   |       |--zzz_default
19  |   |       |   |--__init__.py
20  |   |       |   `--zzz_default.py
21  |   |       `--zzz_hasher

```

```

22 |--__init__.py
23 `--zzz_hasher.py

```

- In this case the **my_api** directory, is the root and all modules (API Resources) start with **zzz_**

Note: The idea of the **prefix** is to avoid conflicts with current python modules

See also:

[pep 395](#), [python import](#)

1.5.6 Rid

The `rid` argument, contains the name of the environ variable containing the request id if any, for example when using GAE:

```
app = zunzuncito.ZunZun(root, versions, hosts, routes, rid='REQUEST_LOG_ID')
```

This helps to add a Request-ID header to all your responses, example when you make a request like:

```
curl -i http://api.zunzun.io/md5/python
```

The response is something like:

```

1 HTTP/1.1 200 OK
2 Request-ID: 52b041e500ff018603acd9c1c60001737e7a756e7a756e6369746f2d617069000131000100
3 Content-Type: application/json; charset=UTF-8
4 Vary: Accept-Encoding
5 Date: Tue, 17 Dec 2013 12:21:57 GMT
6 Server: Google Frontend
7 Cache-Control: private
8 Alternate-Protocol: 80:quic,80:quic
9 Transfer-Encoding: chunked
10
11 {
12   "hash": "23eeeb4347bdd26bfc6b7ee9a3b755dd",
13   "string": "python",
14   "type": "md5"
15 }

```

This can help to trace the request in both server/client side.

If you do not specify the `rid` argument the **Request-ID** will automatically be generated using an **UUID** , so for example you can run the app like this:

```
app = zunzuncito.ZunZun(root, versions, hosts, routes)
# notice there is no rid argument
```

And the output will be something like:

```

1 HTTP/1.1 200 OK
2 Request-ID: 44f237e7-a330-4fb3-ba61-be5abd960688
3 Content-Type: application/json; charset=UTF-8
4 Vary: Accept-Encoding
5 Date: Tue, 17 Dec 2013 12:21:57 GMT
6 Server: Google Frontend
7 Cache-Control: private
8 Alternate-Protocol: 80:quic,80:quic

```

```
9 Transfer-Encoding: chunked
10
11 {
12 "hash": "23eeeb4347bdd26bfc6b7ee9a3b755dd",
13 "string": "python",
14 "type": "md5"
15 }
```

1.5.7 Debug

The python `logging` module is used for creating logs

To enable debugging, set the `debug` argument to **True**, this will log how the API is handling the requests, besides setting the loglevel to **DEBUG** you can run the app like this:

```
app = zunzuncito.ZunZun(root, versions, hosts, routes, debug=True)
```

Note: The default loglevel is **INFO**

1.6 Request class

When receiving a `request` a **request object** is created and passed as an argument to the `dispatch` method of the `APIResource` class

The first argument for the `dispatch` method is the request object:

```
1 from zunzuncito import tools
2
3 class APIResource(object):
4
5
6     def dispatch(self, request, response):
7         """ your code goes here """
```

1.6.1 Request object contents

Name	Description
<code>log</code>	logger intance
<code>request_id</code>	The request id
<code>environ</code>	The <code>wsgi environ</code>
<code>URI</code>	<code>REQUEST_URI</code> or <code>PATH_INFO</code>
<code>host</code>	The host name.
<code>method</code>	The request method (GET, POST, HEAD, etc)
<code>path</code>	list of URI elements
<code>resource</code>	Name of the API resource
<code>version</code>	Current version
<code>vroot</code>	Name of the vroot

1.6.2 Example

```

1 from urlparse import parse_qs
2 from zunzuncito import tools
3
4 class APIResource(object):
5
6
7     @tools.allow_methods('get, post')
8     def dispatch(self, request, response):
9
10         """
11         log this request
12         """
13         request.log.info(tools.log_json({
14             'API': request.version,
15             'Method': request.method,
16             'URI': request.URI,
17             'vroot': request.vroot
18         }, True))
19
20         if request.method == 'POST':
21             data = dict(parse_qs(request.environ['wsgi.input'].read(), True))
22         else
23             data = dict(parse_qs(request.environ['QUERY_STRING'], True))
24
25         data = {k: v.decode('utf-8') for k, v in data.items()}
26
27         return tools.log_json(data)

```

1.7 Response class

All requests need a [response](#), the response class creates an object for every request, the one can be used to send custom headers or HTTP status codes.

The second argument for the dispatch method is the response object:

```

1 from zunzuncito import tools
2
3 class APIResource(object):
4
5
6     def dispatch(self, request, response):
7         """ your code goes here """

```

1.7.1 Response object contents

Name	Description
log	logger intance.
request_id	The request id.
headers	A CaseInsensitiveDict instance, for storing the headers.
status	Default 200 an int representing an HTTP status code.
start_response	The <code>start_response()</code> Callable.
extra	A list for repeated headers used with the <code>add_header</code> method

1.7.2 add_header

If you need to create multiple headers using the same key for example to set up cookies you should use the `add_header` method.

1.7.3 print_str(response)

Returns all the headers to be sent

1.7.4 Example

```
1 from Cookie import SimpleCookie
2 from zunzuncito import tools
3
4
5 class APIResource(object):
6
7     def __init__(self):
8         self.headers['Content-Type'] = 'text/html; charset=UTF-8'
9
10    def dispatch(self, request, response):
11
12        response.headers.update(self.headers)
13
14        cookie = SimpleCookie()
15        cookie.load(request.environ['HTTP_COOKIE'])
16
17        cookie['session'] = session
18        cookie["session"]["path"] = "/"
19        cookie["session"]["expires"] = 12 * 30 * 24 * 60 * 60
20        for morsel in cookie.values():
21            response.add_header('Set-Cookie', morsel.OutputString())
22
23        try:
24            name = request.path[0]
25        except Exception:
26            name = ''
27
28        if name:
29            return 'Name: ' + name
30
31        response.status = 406
32        """ print all headers """
33        print str(response)
34        return []
```

1.8 API Resource

1.8.1 APIResource class

`APIResource` is the name of the class that the `ZunZun` instance will call to handle the incoming requests.

APIResource.dispatch

The ZunZun instance always will call the `dispatch` method that belongs to the `APIResource` class.

```

1 from zunzuncito import tools
2
3
4 class APIResource(object):
5
6     def dispatch(self, request, response):
7
8         request.log.debug(tools.log_json({
9             'API': request.version,
10            'URI': request.URI,
11            'method': request.method,
12            'vroot': request.vroot
13        }, True))
14
15        # print all the environ
16        return tools.log_json(request.environ, 4)

```

For example, the following request:

```
http://127.0.0.1:8080/v0/upload
```

Is handled by the custom python module `zun_upload/zun_upload.py` which contents:

```

1 """
2 upload resource
3
4 Upload by chunks
5
6 @see http://www.grid.net.ru/nginx/resumable_uploads.en.html
7 """
8 import os
9 from zunzuncito import tools
10
11
12 class APIResource(object):
13
14     @tools.allow_methods('post, put')
15     def dispatch(self, request, response):
16         try:
17             temp_name = request.path[0]
18         except:
19             raise tools.HTTPException(400)
20
21         """rfc2616-sec14.html
22         see http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html
23         see http://www.grid.net.ru/nginx/resumable_uploads.en.html
24         """
25         content_range = request.environ.get('HTTP_CONTENT_RANGE', 0)
26
27         length = int(request.environ.get('CONTENT_LENGTH', 0))
28
29         if content_range:
30             content_range = content_range.split()[1].split('/')
31

```

```
32     index, offset = [int(x) for x in content_range[0].split('-')]
33
34     total_size = int(content_range[1])
35
36     if length:
37         chunk_size = length
38     elif offset > index:
39         chunk_size = (offset - index) + 1
40     elif total_size:
41         chunk_size = total_size
42     else:
43         raise tools.HTTPException(416)
44 elif length:
45     chunk_size = total_size = length
46     index = 0
47     offset = 0
48 else:
49     raise tools.HTTPException(400)
50
51 stream = request.environ['wsgi.input']
52
53 body = []
54
55 try:
56     temp_file = os.path.join(
57         os.path.dirname('/tmp/test_upload/'),
58         temp_name)
59
60     with open(temp_file, 'a+b') as f:
61         original_file_size = f.tell()
62
63         f.seek(index)
64         f.truncate()
65
66         bytes_to_write = chunk_size
67
68         while chunk_size > 0:
69             # buffer size
70             chunk = stream.read(min(chunk_size, 1 << 13))
71             if not chunk:
72                 break
73             f.write(chunk)
74             chunk_size -= len(chunk)
75
76         f.flush()
77         bytes_written = f.tell() - index
78
79         if bytes_written != bytes_to_write:
80             f.truncate(original_file_size)
81             f.close()
82             raise tools.HTTPException(416)
83
84     if os.stat(temp_file).st_size == total_size:
85         response.status = 200
86     else:
87         response.status = 201
88         body.append('%d-%d/%d' % (index, offset, total_size))
89
```

```

90         request.log.info(tools.log_json({
91             'index': index,
92             'offset': offset,
93             'size': total_size,
94             'status': response.status,
95             'temp_file': temp_file
96         }, True))
97
98         return body
99     except IOError:
100         raise tools.HTTPException(
101             500,
102             title="upload directory [ %s ]doesn't exist" % temp_file,
103             display=True)

```

Note: All the custom modules must have the **APIResource** class and the method **dispatch** in order to work

1.8.2 dispatch method

The `dispatch` method belongs to the `APIResource` class and is called by the `ZunZun` instance in order to process the requests.

Basic template

```

1 from zunzuncito import tools
2
3 class APIResource(object):
4
5
6     def dispatch(self, request, response):
7         """ your code goes here """

```

Status codes

The default `HTTP` status code is 200, but based on your needs you can change it to fit you response very eazy by just doing something like:

```
response.status = 201
```

Headers

As with the status codes, same happens with the `HTTP` headers, The default headers are:

```
Content-Type: 'application/json; charset=UTF-8'
Request-ID: <request_id>
```

For updating/replacing you just need to do something like:

```
response.headers['my_custom_header'] = str(uuid.uuid4())
```

Example

```
1 from zunzuncito import tools
2
3 class APIResource(object):
4
5     def __init__(self, api):
6         self.headers = {'Content-Type': 'text/html; charset=UTF-8'}
7
8     def dispatch(self, request, response):
9
10        try:
11            name = self.api.path[0]
12        except:
13            name = ''
14
15        if name:
16            response.headers['my_custom_header'] = name
17        else:
18            response.status = 406
19
20        response.headers.update(self.headers)
21
22        return 'Name: ' + name
```

The output for:

```
curl -i http://api.zunzun.io/status_and_headers
```

```
1 HTTP/1.1 406 Not Acceptable
2 Request-ID: 52e78a1500ff0f217359e91eb90001737e7a756e7a756e6369746f2d617069000131000100
3 Content-Type: application/json; charset=UTF-8
4 Vary: Accept-Encoding
5 Date: Tue, 28 Jan 2014 10:44:38 GMT
6 Server: Google Frontend
7 Cache-Control: private
8 Alternate-Protocol: 80:quic,80:quic
9 Transfer-Encoding: chunked
10
11 Name:
```

The output for:

```
curl -i http://api.zunzun.io/status_and_headers/foo
```

```
1 HTTP/1.1 200 OK
2 Request-ID: 52e78a9300ff0f3fe44a7e4fbf0001737e7a756e7a756e6369746f2d617069000131000100
3 my_custom_header: foo
4 Content-Type: application/json; charset=UTF-8
5 Vary: Accept-Encoding
6 Date: Tue, 28 Jan 2014 10:46:44 GMT
7 Server: Google Frontend
8 Cache-Control: private
9 Alternate-Protocol: 80:quic,80:quic
10 Transfer-Encoding: chunked
11
12 Name: foo
```

See also:

pep 0333

1.8.3 @allow_methods decorator

The @allow_methods decorator when applied to the dispatch method works like a filter to specified [HTTP methods](#)

Example

```

1 from zunzuncito import tools
2
3 class APIResource(object):
4
5     @tools.allow_methods('post, put')
6     def dispatch(self, request, response):
7         """ your code goes here """

```

In this case all the request that are not **POST** or **PUT** will be rejected

1.8.4 path

path is the name of the variable containing a [list](#) of elements of the URI after has been parsed.

Suppose the incoming request is:

```
http://api.zunzun.io/v1/gevent/ip
```

ZunZun instance will convert it to:

```
['v1', 'gevent', 'ip']
```

where:

```

root = default
version = v1
resource = gevent
path = ['ip']

```

for the incoming request:

```
http://api.zunzun.io/gevent/aa/bb
```

this will be generated:

```

root = default
version = v0
resource = gevent
path = ['aa', 'bb']

```

Example

```

1 from zunzuncito import tools
2
3
4 class APIResource(object):
5

```

```

6  def dispatch(self, request, response):
7
8      try:
9          name = request.path[0]
10     except Exception:
11         name = ''
12
13     return 'Name: ' + name

```

1.8.5 URI module

Every resource has a module, this is made with the intention to have more order and give more flexibility.

URI parts

```

http://api.zunuz.io/v0/get/client/ip
|         |         |         |         |
|         |         |         |         |
|         |         |         |         |
host (default)  version | resource | path |
                    |         |         |
                    path[0] |         |
                        |         |
                        path[1]

```

ZunZun translates that URI to:

```
my_api.default.v0.zun_get.zun_client.zun_client
```

Example

The request <http://api.zunuz.io/get/client/> will be handled by the file `zun_cilent.py` notice that that the URI ends with an /

If the request where <http://api.zunuz.io/get/client> without the ending slash it will be handled by `zun_get.py`.

See also:

[URI scheme](#), [Uniform_resource_locator](#)

1.9 _catchall resource

ZunZun process request only for does who have an existing module, for example using the following directory structure notice we only have 2 modules, `zun_default`, `zun_hassher` and `zun_gevent`.

```

1 /home/
2   |--zunzun/
3     |--app.py
4     |--my_api
5       |--__init__.py
6       |--default
7         |--__init__.py
8         |--v0

```

```

9      |--__init__.py
10     |--zun_default
11     | |--__init__.py
12     | `--zun_default.py
13     |--zun_hasher
14     | |--__init__.py
15     | `--zun_hasher.py
16     `--zun_gevent
17         |--__init__.py
18         `--zun_gevent.py

```

The routes uses regular expressions to match more then one request into one module, for example:

```

1 routes = {'default':[
2     ('/(md5|sha1|sha256|sha512) (/.*)?', 'hasher', 'GET, POST'),
3     ('/.*', 'notfound')
4 ]}

```

For example the `zun_hasher` module will handle all the request for URI containing:

```
/md5, /sha1, /sha256, /sha512
```

But the regex:

```
"/.*"
```

Will match anything and send it to the `zun_notfound` module, the problem with this regex, is that since it will catch anything, if you make a request for example to:

```
api.zunzun.io/gevent
```

It will be processed by the `zun_notfound` since the regex caught the request.

A solution to this, could be to add a route for the `gevent` request, something like this:

```

1 routes = {'default':[
2     ('/(md5|sha1|sha256|sha512) (/.*)?', 'hasher', 'GET, POST'),
3     ('/gevent/?.*', 'gevent'),
4     ('/.*', 'notfound')
5 ]}

```

That could solve the problem, but forces you to have a route for every module, the more modules you have, the more complicated and difficult becomes to maintain the routes dictionary.

So, be available to have regular expressions mathing many to one module, to continue serving directly from modules that don't need a regex, and to also have a catchall that does not need require a regex and it is only called when all the routes and modules options have been exhausted, the `_catchall` module was created.

1.9.1 How it works

The only thing required, is to create a module with the name `__catchall`, if using the default prefix, it would be `zun__catchall`.

- Notice the double `__catchall` underscore.

See also:

The `zun` prefix

1.9.2 Directory structure

```

1 /home/
2   |--zunzun/
3     |--app.py
4     |--my_api
5       |--__init__.py
6       |--default
7         |--__init__.py
8         |--v0
9           |--__init__.py
10            |--zun_default
11              |--__init__.py
12              |--zun_default.py
13              |--zun_hasher
14                |--__init__.py
15                |--zun_hasher.py
16                |--zun_gevent
17                  |--__init__.py
18                  |--zun_gevent.py
19                  |--zun__catchall
20                    |--__init__.py
21                    |--zun__catchall.py

```

When processing a request, if not module is found either in the routes or in the directory structure, if the `__catchall` module is found, it is going to be used for handling the request, if not it will just return an HTTP 501 Not Implemented status.

1.9.3 Example

The following example, will handle all *not found* modules for the incoming request, and redirect to `'/'`.

```

1 """
2 catchall resource
3 """
4
5 from zunzuncito import tools
6
7
8 class APIResource(object):
9
10     def __init__(self):
11         self.headers = {'Content-Type': 'text/html; charset=UTF-8'}
12
13     def dispatch(self, request, response):
14         request.log.debug(tools.log_json({
15             'API': request.version,
16             'URI': request.URI,
17             'rid': request.request_id,
18             'vroot': request.vroot
19         }, True))
20
21         response.headers.update(self.headers)
22
23         raise tools.HTTPException(302, headers={'Location': '/'}, log=False)

```


1.10 tools

tools is a module that containing a set of classes and functions that help to process the reply of the request more easy.

```

1 from zunzuncito import tools
2
3 class APIResource(object):
4
5     def dispatch(self, request, response):
6         """ your code goes here """

```

1.10.1 HTTPException

The HTTPException class extends the `HTTPError` class, the main idea of it, is to handle possible errors and properly reply with the corresponding HTTP status code.

```
HTTPException(status, title=None, description=None, headers=None, code=None, display=False, log=True)
```

Note: The headers argument must be a dictionary containing HTTP header fields

Redirect example

```

1 from zunzuncito import tools
2
3
4 class APIResource(object):
5
6     def dispatch(self, request, response):
7         try:
8             name = request.path[0]
9         except:
10            raise tools.HTTPException(
11                302,
12                headers={'Location': '/home'},
13                log=False)

```

Bad request example

```

1 from zunzuncito import tools
2
3
4 class APIResource(object):
5
6     def dispatch(self, request, response):
7         try:
8             name = request.path[0]
9         except:
10            raise tools.HTTPException(400)

```

the line:

```
raise tools.HTTPException(400)
```

For a request like:

```
curl -i http://api.zunzun.io/exception
```

Will reply with this:

```
1 HTTP/1.1 400 Bad Request
2 Request-ID:
3 52c597a700ff0229fef9f477280001737e7a756e7a756e6369746f2d617069000131000100
4 Content-Type: application/json; charset=UTF-8
5 Date: Thu, 02 Jan 2014 16:45:27 GMT
6 Server: Google Frontend
7 Content-Length: 0
8 Alternate-Protocol: 80:quic,80:quic
```

This is because the request URI is missing the path and should be something like:

```
curl -i http://api.zunzun.io/exception/foo
```

That will return something like:

```
1 HTTP/1.1 200 OK
2 Request-ID:
3 52c597d200ff0d89f81dcec4280001737e7a756e7a756e6369746f2d617069000131000100
4 Content-Type: application/json; charset=UTF-8
5 Vary: Accept-Encoding
6 Date: Thu, 02 Jan 2014 16:46:11 GMT
7 Server: Google Frontend
8 Cache-Control: private
9 Alternate-Protocol: 80:quic,80:quic
10 Transfer-Encoding: chunked
11
12 my_api.default.v0.zun_exception.zun_exception
```

Note: If you only pass the integer HTTP status code to the `HTTPException`, only the response headers will be sent.

Body response

Besides only replying with the headers you may want to give a more informative / verbose message, the `HTTPException` accept the following arguments:

```
HTTPException(status, title=None, description=None, headers=None, code=None, display=False, log=True)
```

For example the following snippet of code taken from `zun_exception.py`:

```
1 if name != 'foo':
2     raise tools.HTTPException(
3         406,
4         title='exception example',
5         description='name must be foo',
6         code='my-custom-code',
7         display=True)
```

When the request is:

```
curl -i http://api.zunzun.io/v0/exception/naranjas
```

Notice that the path in this case is:

```
path = ['naranjas']
```

Will reply with something like:

```

1 HTTP/1.1 406 Not Acceptable
2 Request-ID: 52c59bdf00ff0b7042cbfd5d120001737e7a756e7a756e6369746f2d617069000131000100
3 Content-Type: application/json; charset=UTF-8
4 Vary: Accept-Encoding
5 Date: Thu, 02 Jan 2014 17:03:27 GMT
6 Server: Google Frontend
7 Cache-Control: private
8 Alternate-Protocol: 80:quic,80:quic
9 Transfer-Encoding: chunked
10
11 {
12     "code": "my-custom-code",
13     "description": "name must be foo",
14     "status": "406",
15     "title": "expection example"
16 }
```

1.10.2 MethodException

While defining routes or using the `@allow_methods` decorator you can specify the allowed [HTTP methods](#) to support, the ZunZun instance, internally will verify for the corret method, otherwise will raise an `MethodException`

The `MethodException` behaves similar to the `HTTPException` the only difference is that by default sets the status code to `405 Method Not Allowed`

```
MethodException(status=405, title=None, description=None, headers=None, code=None, display=False)
```

For example the `zun_exception.py` custom module only accepts ‘GET’ methods from this code snippet:

```
@tools.allow_methods('get')
def dispatch(self, request, response):
```

Therefor if you try the following:

```
curl -i -X HEAD http://api.zunzun.io/exception/foo
```

- Note the: **-X HEAD** this will send a **HEAD** request not a **GET**

The answer will be simillar to:

```

1 HTTP/1.1 405 Method Not Allowed
2 Request-ID: 52c6ac4e00ff060346c67c66450001737e7a756e7a756e6369746f2d617069000131000100
3 Content-Type: application/json; charset=UTF-8
4 Date: Fri, 03 Jan 2014 12:25:50 GMT
5 Server: Google Frontend
6 Content-Length: 0
7 Alternate-Protocol: 80:quic,80:quic
```

1.10.3 allow_methods

The `allow_methods` is a function used as [decorator](#)

See also:

@allow_methods decorator

1.10.4 log_json

The `log_json` is a function that given a [dictionary](#), returns a [json](#) structure.

This helps that logs can be parsed and processed by external tools.

The arguments are:

```
log_json(log, indent=False)
```

log a python dictionary

indent returns the json structured indented (more human readable)

Note: If `indent` is a non-negative integer, then JSON array elements and object members will be pretty-printed with that indent level. An indent level of 0, or negative, will only insert newlines. `None` (the default) selects the most compact representation.

See also:

[python json](#)

Example

```
1 from zunzuncito import tools
2
3 class APIResource(object):
4
5     def dispatch(self, request, response):
6
7         request.log.debug(tools.log_json({
8             'API': request.version,
9             'Method': request.method,
10            'URI': request.URI,
11            'vroot': request.vroot
12        }, True))
13
14     """ your code goes here """
```

See also:

[Structured Logging](#)

1.10.5 clean_dict

The `clean_dict` is a small recursive function that given a [dictionary](#) will try to ‘clean it’ converting it to a string:

```
clean_dict(dictionary)
```

It is commonly used in conjunction with the `log_json` function.

Example

snippet taken from `zun_self.py`:

```

1  def dispatch(self, request, response):
2
3      return (
4          json.dumps(
5              tools.clean_dict(request.__dict__),
6              sort_keys=True,
7              indent=4)
8      )
9      # taking advantage of the tools.log_json
10     # return tools.log_json(request.__dict__, 4)

```

1.10.6 CaseInsensitiveDict

A case-insensitive dict-like object.

Note: Class taken from: `requests`

Implements all methods and operations of `collections.MutableMapping` as well as `dict`'s `copy`. Also provides `lower_items`.

All keys are expected to be strings. The structure remembers the case of the last key to be set, and `iter(instance)`, `keys()`, `items()`, `iterkeys()`, and `iteritems()` will contain case-sensitive keys. However, querying and contains testing is case insensitive:

```

cid = CaseInsensitiveDict()
cid['Accept'] = 'application/json'
cid['aCCEPT'] == 'application/json' # True
list(cid) == ['Accept'] # True

```

For example, `headers['content-encoding']` will return the value of a 'Content-Encoding' response header, regardless of how the header name was originally stored.

If the constructor, `.update`, or equality comparison operations are given keys that have equal `.lower()` s, the behavior is undefined.

1.11 HTTP status codes

```

1  # 1xx: Informational - Request received, continuing process
2  HTTP_100 = '100 Continue' # [RFC2616]
3  HTTP_101 = '101 Switching Protocols' # [RFC2616]
4  HTTP_102 = '102 Processing' # [RFC2518]
5
6  # 2xx: Success - The action was successfully received, understood, and accepted
7  HTTP_200 = '200 OK' # [RFC2616]
8  HTTP_201 = '201 Created' # [RFC2616]
9  HTTP_202 = '202 Accepted' # [RFC2616]
10 HTTP_203 = '203 Non-Authoritative Information' # [RFC2616]
11 HTTP_204 = '204 No Content' # [RFC2616]
12 HTTP_205 = '205 Reset Content' # [RFC2616]
13 HTTP_206 = '206 Partial Content' # [RFC2616]
14 HTTP_207 = '207 Multi-Status' # [RFC4918]

```

```
15 HTTP_208 = '208 Already Reported' # [RFC5842]
16 HTTP_226 = '226 IM Used' # [RFC3229]
17
18 # 3xx: Redirection - Further action must be taken in order to complete the request
19 HTTP_300 = '300 Multiple Choices' # [RFC2616]
20 HTTP_301 = '301 Moved Permanently' # [RFC2616]
21 HTTP_302 = '302 Found' # [RFC2616]
22 HTTP_303 = '303 See Other' # [RFC2616]
23 HTTP_304 = '304 Not Modified' # [RFC2616]
24 HTTP_305 = '305 Use Proxy' # [RFC2616]
25 HTTP_306 = '306 Reserved' # [RFC2616]
26 HTTP_307 = '307 Temporary Redirect' # [RFC2616]
27 HTTP_308 = '308 Permanent Redirect' # [RFC-reschke-http-status-308-07]
28
29 # 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
30 HTTP_400 = '400 Bad Request' # [RFC2616]
31 HTTP_401 = '401 Unauthorized' # [RFC2616]
32 HTTP_402 = '402 Payment Required' # [RFC2616]
33 HTTP_403 = '403 Forbidden' # [RFC2616]
34 HTTP_404 = '404 Not Found' # [RFC2616]
35 HTTP_405 = '405 Method Not Allowed' # [RFC2616]
36 HTTP_406 = '406 Not Acceptable' # [RFC2616]
37 HTTP_407 = '407 Proxy Authentication Required' # [RFC2616]
38 HTTP_408 = '408 Request Timeout' # [RFC2616]
39 HTTP_409 = '409 Conflict' # [RFC2616]
40 HTTP_410 = '410 Gone' # [RFC2616]
41 HTTP_411 = '411 Length Required' # [RFC2616]
42 HTTP_412 = '412 Precondition Failed' # [RFC2616]
43 HTTP_413 = '413 Request Entity Too Large' # [RFC2616]
44 HTTP_414 = '414 Request-URI Too Long' # [RFC2616]
45 HTTP_415 = '415 Unsupported Media Type' # [RFC2616]
46 HTTP_416 = '416 Requested Range Not Satisfiable' # [RFC2616]
47 HTTP_417 = '417 Expectation Failed' # [RFC2616]
48 HTTP_422 = '422 Unprocessable Entity' # [RFC4918]
49 HTTP_423 = '423 Locked' # [RFC4918]
50 HTTP_424 = '424 Failed Dependency' # [RFC4918]
51 HTTP_425 = '425 Unassigned'
52 HTTP_426 = '426 Upgrade Required' # [RFC2817]
53 HTTP_427 = '427 Unassigned'
54 HTTP_428 = '428 Precondition Required' # [RFC6585]
55 HTTP_429 = '429 Too Many Requests' # [RFC6585]
56 HTTP_430 = '430 Unassigned'
57 HTTP_431 = '431 Request Header Fields Too Large' # [RFC6585]
58
59 # 5xx: Server Error - The server failed to fulfill an apparently valid request
60 HTTP_500 = '500 Internal Server Error' # [RFC2616]
61 HTTP_501 = '501 Not Implemented' # [RFC2616]
62 HTTP_502 = '502 Bad Gateway' # [RFC2616]
63 HTTP_503 = '503 Service Unavailable' # [RFC2616]
64 HTTP_504 = '504 Gateway Timeout' # [RFC2616]
65 HTTP_505 = '505 HTTP Version Not Supported' # [RFC2616]
66 HTTP_506 = '506 Variant Also Negotiates (Experimental)' # [RFC2295]
67 HTTP_507 = '507 Insufficient Storage' # [RFC4918]
68 HTTP_508 = '508 Loop Detected' # [RFC5842]
69 HTTP_509 = '509 Unassigned'
70 HTTP_510 = '510 Not Extended' # [RFC2774]
71 HTTP_511 = '511 Network Authentication Required' # [RFC6585]
```

See also:

HTTP status codes

1.12 zunzuncito Package

1.12.1 zunzuncito Package

1.12.2 http_status_codes Module

1.12.3 request Module

1.12.4 response Module

1.12.5 tools Module

1.12.6 zunzun Module

1.13 WebOb

1.13.1 What is it?

WebOb is a Python library that provides wrappers around the WSGI request environment, and an object to help create WSGI responses. The objects map much of the specified behavior of HTTP, including header parsing, content negotiation and correct handling of conditional and range requests.

This helps you create rich applications and valid middleware without knowing all the complexities of WSGI and HTTP.

1.13.2 Why ?

The ZunZun instance allows you to handle the request by following defined routes and by calling the dispatch method, the way you process the request is up to you, you can either do all by your self or use tools that can allow you to simplify this process, for this last one, **WebOb** is a library that integrates very easy and that may help you to parse the GET, POST arguments, set/get cookies, etc; with out hassle.

1.13.3 Example

The following code, handles the request for <http://api.zunzun.io/webob>.

```
1 from webob import Request
2 from zunzuncito import tools
3
4
5 class APIResource(object):
6
7
8     def dispatch(self, request, response):
9         req = Request(request.environ)
10
11         data = {}
```

```
12     data['req-GET'] = req.GET
13     data['req-POST'] = req.POST
14     data['req-application_url'] = req.application_url
15     data['req-body'] = req.body
16     data['req-content_type'] = req.content_type
17     data['req-cookies'] = req.cookies
18     data['req-method'] = req.method
19     data['req-params'] = req.params
20     data['req-path'] = req.path
21     data['req-path_info'] = req.path_info
22     data['req-path_qs'] = req.path_qs
23     data['req-path_url'] = req.path_url
24     data['req-query_string'] = req.query_string
25     data['req-script_name'] = req.script_name
26     data['req-url'] = req.url
27
28     return tools.log_json(data, 4)
```

Basically you only need to pass the `environ` argument to the `webob.Request`:

```
def dispatch(self, environ):
    req = Request(environ)
    """ your code goes here """
```

See also:

[WebOb Request](#)

1.13.4 GAE

When using google app engine you need to add this lines to your `app.yaml` file in order to be available to import `webob`:

```
libraries:
- name: webob
  version: latest
```

1.14 Jinja2

`Jinja2` is a modern and designer friendly templating language for Python, modelled after Django's templates. It is fast, widely used and secure with the optional sandboxed template execution environment:

```
1 <title>{% block title %}{% endblock %}</title>
2 <ul>
3     {% for user in users %}
4         <li><a href="{{ user.url }}">{{ user.username }}</a></li>
5     {% endfor %}
6 </ul>
```

1.14.1 Why ?

`Zunzuncito` was made mainly for responding in `json format` not HTML, but also one of the design goals is to give the ability to create almost anything easy, therefor if you need to display HTML, `Jinja2` integrates very easy.

1.14.2 Example

The following code, handles the request for: <http://api.zunuz.io/jinja2>.

```

1 import os
2
3 from jinja2 import Environment, FileSystemLoader
4 from zununcito import tools
5
6 jinja = Environment(autoescape=True, loader=FileSystemLoader(
7     os.path.join(os.path.abspath(os.path.dirname(__file__)), 'templates')))
8
9
10 class APIResource(object):
11
12     def __init__(self):
13         self.headers['Content-Type'] = 'text/html; charset=UTF-8'
14
15     def dispatch(self, request, response):
16
17         request.log.debug(tools.log_json({
18             'API': request.version,
19             'method': request.method,
20             'URI': request.URI,
21             'vroot': request.vroot
22         }, True))
23
24         response.headers.update(self.headers)
25
26         template_values = {
27             'IP': request.environ.get('REMOTE_ADDR', 0)
28         }
29
30         template = jinja.get_template('example.html')
31
32         return template.render(template_values).encode('utf-8')

```

The example.html contains:

```

1 <html>
2   <head>
3     <meta charset="utf-8">
4     <title>{{ IP }}</title>
5   </head>
6
7   <body>
8     <h3>IP: {{ IP }}</h3>
9   </body>
10 </html>

```

1.14.3 Directory structure

```

1 /home/
2   |--zunzun/
3     |--app.py
4     |--my_api
5     |--__init__.py

```

```
6     |--default
7         |--__init__.py
8     |--v0
9         |--__init__.py
10        |--zun_jinja2
11            |--__init__.py
12            |--zun_jinja2.py
13        |--templates
14            |--example.html
```

See also:

[zun_jinja2 API resource & Jinja2 Basics](#)

1.14.4 GAE

When using google app engine you need to add this lines to your `app.yaml` file in order to be available to import `jinja2`:

```
libraries:
- name: jinja2
  version: latest
```

1.15 gevent

`gevent` is a coroutine-based Python networking library that uses `greenlet` to provide a high-level synchronous API on top of the `libev` event loop.

When using `gevent` and `'yield'` you may want to call the `'start_response'` before so that it can send your proper headers, for this you must use something like:

```
response.send()
```

See also:

[The start_response\(\) Callable](#)

1.15.1 Example

The following code will:

- output the result using content-type: `'text/html'`,
- print `'sleep 1 second.'` and call `gevent.sleep(1)`,
- print `'sleep 3 second...'` and call `gevent.sleep(3)`,
- print `'done. getting some ips...'` call `gevent.socket.gethostbyname`

An example of the output can be seen here: <https://www.youtube.com/watch?v=0N6qXkT-t5E>; notice that the prints are secuencial not in one shot.

```
1 import gevent
2 import gevent.socket
3
4 from zunzuncito import tools
5
6
```

```
7 def bg_task():
8     for i in range(1, 10):
9         print "background task", i
10        gevent.sleep(2)
11
12
13 def long_task():
14     for i in range(1, 10):
15         print i
16         gevent.sleep()
17
18
19 class APIResource(object):
20
21     def __init__(self):
22         self.headers = {'Content-Type': 'text/html; charset=UTF-8'}
23
24     def dispatch(self, request, response):
25
26         request.log.debug(tools.lo_g_json({
27             'API': request.version,
28             'Method': request.method,
29             'URI': request.URI,
30             'vroot': request.vroot
31         }, True))
32
33         response.headers.update(self.headers)
34
35         """
36         calls start_response
37         """
38         response.send()
39
40         t = gevent.spawn(long_task)
41         t.join()
42
43         yield "sleep 1 second.<br/>"
44
45         gevent.sleep(1)
46
47         yield "sleep 3 seconds...<br/>"
48
49         gevent.sleep(3)
50
51         yield "done.<br/>getting some ips...<br/>"
52
53         urls = [
54             'www.google.com',
55             'www.example.com',
56             'www.python.org',
57             'zunzun.io']
58
59         jobs = [gevent.spawn(gevent.socket.gethostbyname, url) for url in urls]
60         gevent.joinall(jobs, timeout=2)
61
62         for j in jobs:
63             yield "ip = %s<br/>" % j.value
64
```

1.16 Changelog

1.16.1 0.1.20 (2014-08-12)

- added `is_secure` method to Request class, `request.is_secure()` will return True if using https

1.16.2 0.1.19 (2014-07-15)

- added `host_url` property to Request class, `request.host_url` will return something like `http://0:8080` depending on your url

1.16.3 0.1.18 (2014-04-19)

- fixed bug on `tools.py` to remove `'log=True'` on the response body when raising an exception and using `display=True`.

1.16.4 0.1.17 (2014-04-17)

- Added the `add_header` method to the response class, with the intention to allow multiple headers with the same name, example `'Set-Cookie'`.
- Compiling regex using `'r'` (raw string notation).
- Improved logs order and removed `'get'` to use `'in'` instead.

1.16.5 0.1.16 (2014-03-09)

- fixed bug on `py_mod` to allow sub modules based on the URI to work properly, see `URI_module`
- fixed `__init__` to make custom versions match `allowed_URI_chars ^ [\w-]+$`
- changed UUID4 to UUID1

1.16.6 0.1.15 (2014-02-27)

- log when trying to load the `_catchall`, if no `_catchall` raise Exception about the missing module
- replaced `iteritems` with `items()` to be Python 3 compatible

1.16.7 0.1.14 (2014-02-26)

- replaced `itertools.filter` with `filter`
- improve `py_mod` if a URI ends with an slash for example: `http://api.zunzun.io/v1/add/user/`, the `py_mod` will be: `zun_add/zun_user/zun_user.py`

1.16.8 0.1.13 (2014-02-17)

- Added the log option to the HTTPException, if set to True it will log the exception otherwise not.

1.16.9 0.1.12 (2014-02-13)

- Fixed core to be `thread safe`.
- New classes `request`, `response`, the `dispatch` method require this `dispatch(self, request, response)`.
- lazy load of resources.
- `__catchall` module

1.16.10 0.1.11 (2014-02-04)

- Fixed bug in `http_status_codes.py` when handling generic reasons.

1.16.11 0.1.10 (2014-01-28)

- `dispatch` method requires now only one argument, which is `environ`, the `start_response` is handled by the API it self.
- `http_status_codes` now is a dictionary.

1.16.12 0.1.9 (2014-01-06)

- `self._headers` is created only once at the beginning and per request just copied to `self.headers`.

1.16.13 0.1.8 (2014-01-04)

- Fixed `tools.log_json` function to not indent when no indent value is set.

1.17 Issues

Please report any problem, bug, here: <https://github.com/nbari/zunzuncito/issues>

1.18 FAQ

1.18.1 Why I get a warnings for some requests?

If you get something like this:

```
1 WARNING 2014-03-07 10:01:30,845 zunzun.py:102] {
2   "API": "v0",
3   "HTTPError": "501",
4   "URI": "/",
5   "body": {
6     "code": "None",
7     "description": "No module named myapp_api.default.v0.zun_default.zun_default",
8     "display": "False",
9     "headers": "None",
10    "log": "True",
11    "status": "501",
12    "title": "ImportError: myapp_api.default.v0.zun_default.zun_default, myapp_api.default.v0.zun_cat
13  },
14  "method": "GET",
15  "rid": "f8df3ffc8294c0ec0fcc10bbc7c4bfe0febbcaaf83ff619ab56ca0209225dd0d5f1fd19e42f6b4c1fa2ef0a1f3
16 }
```

Is because you could be missing an `__init__.py`, all the subdirectories of your *API* need need to be treated like python modules.

See also:

This is the directory structure.