ejpiaj Documentation

Release 0.4.0

Marek Wywiał

Mar 06, 2018

Contents

Contents:

CHAPTER 1

ejpiaj

pypi package 0.4.4 build passing

1.1 License

- Free software: BSD license

1.2 Features

- describe your API requests in single file (YAML, JSON and XML at this moment) so you can store you API tests with code in same repository (f.i. as ejpiaj.json file)
- file format is detected from file extension `.yml`, `.json` and `.xml`
- extract variables from responses and store them to use in next requests (f.i. to get and use authorization token)
- write assertions agains responses
- register your own variables extractors and assertions
- run suite using `ejpiaj-cli test -m my_addons -s tests.yml` command

Sample yml file:

```
requests:
  001_search_repos_with_django_in_name:
    method: get
    url: https://api.github.com/search/repositories
    url_params:
      q: django
      sort: stars
      order: desc
    variables:
      json:
        total_count: count
        items.[0].full_name: repo_name
    assertions:
      response:
        - 'status_code equals 200'
      json:
        - 'items.[0].full_name contains ango'

  002_get_commits_from_first_repo:
    method: get
    url: https://api.github.com/repos/{{repo_name}}/commits
    assertions:
      response:
        - 'status_code equals 200'
```

Run it:

```
$ ejpiaj-cli test sample.yml -s

--------------------------------------------------------------------------------
P - passed assertions, F - failed assertions, V - extracted variables
--------------------------------------------------------------------------------
✓ 001_search_repos_with_django_in_name [P2,F0,V2] {'count': 29754, 'repo_name': u
↪'django/django'}
✓ 002_get_commits_from_first_repo [P1,F0,V0] {}
--------------------------------------------------------------------------------
```

1.3 SNI note

In order to support SNI in python 2.6/2.7 you need to install additional packages:

- *pyOpenSSL*, a Python wrapper module around the OpenSSL library.
- *ndg-httpsclient*, enhanced HTTPS support for httplib and urllib2.
- *pyasn1*, ASN.1 types and codecs.

1.4 Documentation

- http://ejpiaj.readthedocs.org/en/latest/

CHAPTER 2

Installation

At the command line:

```
$ easy_install ejpiaj
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv ejpiaj
$ pip install ejpiaj
```

CHAPTER 3

Usage

Simple usage with ejpiaj-cli.

A ejpiaj-cli tool has one command test:

```
$ ejpiaj-cli test --help

Usage: ejpiaj-cli test <yaml_file> [<debug>] [<module>] [<display_variables>]

Run tests using yaml file

Required Arguments:

  yaml_file

Options:

   -d --debug   run with debug mode
   -s --display_variables  display extracted variables
   -m --module  your module with custom extractors and assertions
   -e --env     initial variables in format var1=val1&var2=val2
```

A yaml_file is file with tests. Debug mode (-d) displays logs and returns content from requests.

A --module option allows you to specify own module with custom assertions and variables extractors. F.i.:

```
$ ejpiaj-cli test ./myapi.yml --module my_module -s
```

A -e allows you to pass initial variables f.i.:

```
$ ejpiaj-cli test ./myapi.yml --module my_module -s -e api_url=localhost
```

And use this variable in yml file:

```
requests:
  001_get_token:
    method: post
    url: http://{{api_url}}/api/v10/api-token-auth/
```

I will explain idea using example `example_full.yml` file:

- https://github.com/onjin/ejpiaj/blob/master/examples/example_full.yml

All requests are written under key *requests*. Every request has unique name. It's name is used to sort request while runing, so numeric prefix is very convinient.

Every request is build from elements:

- method - request method like 'get', 'post', 'put', 'options' (under the hood is requests library)
- url - full url to call
- url_params - params added to url after '?' sign
- form_params - params used with POST method and PUT
- body - POST or PUT body, if used then 'form_params' will be skipped
- variables - variables to extract using registered variables extractors
- assertions - assertions to run using also variables extractors and registered assertions

3.1 Simple example

First example:

```
requests:
  001_search_repos_with_django_in_name:
    method: get
    url: https://api.github.com/search/repositories
    url_params:
      q: django
      sort: stars
      order: desc
```

Run it with:

```
ejpiaj-cli test -s examples/example_001.yml
```

The result should be:

```
--------------------------------------------------------------------------------
P - passed assertions, F - failed assertions, V - extracted variables
--------------------------------------------------------------------------------
✓ 001_search_repos_with_django_in_name [P0,F0,V0] {}
--------------------------------------------------------------------------------
```

`P0` means 0 passed assertions, `F0` means 0 failed assertions, `V0` means 0 extracted variables

3.2 Assertions

Now we are going to add first assertions:

```
requests:
  001_search_repos_with_django_in_name:
    method: get
    url: https://api.github.com/search/repositories
    url_params:
      q: django
      sort: stars
      order: desc
    assertions:
      response:
        - 'status_code equals 200'
      json:
        - 'items.[0].full_name contains ango'
```

Run it with:

```
ejpiaj-cli test -s examples/example_002.yml
```

The result should be:

```
--------------------------------------------------------------------------------
P - passed assertions, F - failed assertions, V - extracted variables
--------------------------------------------------------------------------------
✓ 001_search_repos_with_django_in_name [P2,F0,V0] {}
--------------------------------------------------------------------------------
```

Under key *assertions* we put any variables extractor registered name (json, request). Under this key we put list of assertions in format:

```
variable assertions parameter
```

`variables` is variable extractor parameter, `assertion` is assertion keyword and `parameter` is optional parameter for assertion (depends on assertion type)

In this example we used *response* extractor:

```
response:
 - 'status_code equals 200'
```

So we told *response* extractor to get *status_code* attribute from response object and test if it equals to *200*

We used also *json* extractor:

```
json:
  - 'items.[0].full_name contains ango'
```

So we told *json* extractor to get *items.[0].full_name* from response:

```
{
  "total_count": 29532,
  "items": [
    {
      "id": 4164482,
      "name": "Django",
      "full_name": "django/django",
      "owner": {
        ...
      },
    }
}
```

and check if the *full_name* contains word *ango*

3.3 Variables extracting

We can use variables extractors to extract and store variables for further usage in next requests.

Extracting and using variables:

```
requests:
  001_search_repos_with_django_in_name:
    method: get
    url: https://api.github.com/search/repositories
    url_params:
      q: django
      sort: stars
      order: desc
    variables:
      json:
        total_count: count
        items.[0].full_name: repo_name
    assertions:
      response:
        - 'status_code equals 200'
      json:
        - 'items.[0].full_name contains ango'

  002_get_commits_from_first_repo:
    method: get
    url: https://api.github.com/repos/{{repo_name}}/commits
    assertions:
      response:
        - 'status_code equals 200'
```

Run it with:

```
ejpiaj-cli test -s examples/example_003.yml
```

The result should be:

```
--------------------------------------------------------------------------------
P - passed assertions, F - failed assertions, V - extracted variables
--------------------------------------------------------------------------------
✓ 001_search_repos_with_django_in_name [P2,F0,V2] {'count': 29532, 'repo_name': u
↪'django/django'}
✓ 002_get_commits_from_first_repo [P1,F0,V0] {}
--------------------------------------------------------------------------------
```

We simply added `variables` key and used same variable extractor as in *assertions*:

```
variables:
  json:
    total_count: count
    items.[0].full_name: repo_name
```

And now we have variables:

```
count = 29532
repo_name = django/django
```

And we can use those variables in next request:

```
002_get_commits_from_first_repo:
  method: get
  url: https://api.github.com/repos/{{repo_name}}/commits
```

Variables are put between '{{' and '}}' and **can't** contains spaces'. For example:

```
{{repo_name}} - it's good
{{ repo_nama}} - it's wrong
```

3.4 Full example

Now you can could understand full example at file:

- https://github.com/onjin/ejpiaj/blob/master/examples/example_full.yml

CHAPTER 4

Variables extractors

Variables extractors are used to extract variables for assertions or to store them and use in next requests.

4.1 Builtin variables extractors

4.1.1 response

A `response` variables extractor gives you access to attributes of response objects:

- http://requests.readthedocs.org/en/latest/user/advanced/#request-and-response-objects

Usage:

```
variables:
  response:
    status_code: last_code

assertions:
  response:
    - 'status_code equals 200'
```

4.1.2 json

An `json` extractor treats response content as json. You can access json body using python dictionary syntax.

Usage:

```
variables:
  json:
    '[0].sha': sha1
    '[1].sha': sha2
    '[2].sha': sha3
```

```
assertions:
  json:
    - 'items.[0].full_name contains ango'
```

4.2 Custom variables extractors

You can easily create your own extractors by creating python file with code:

```python
import re

import json

from ejpiaj.decorators import variable_extractor


@variable_extractor('json2')
def json2_variables_extractor(response, variables):
    """Extracts variables from json response.content.

    Variables path are written using 'dot' access and index access to lists
    f.i.:
        some.path.to.list.[0]
        [1].dict.access.later
    """
    result = {}
    re_list = re.compile('^\[\d+\]$')

    # use 'dot' access to dictionary
    data = json.loads(response.content)
    for path, name in variables.items():
        try:
            subdata = data
            for attr in path.split('.'):
                # support for list access [0]
                if re_list.match(attr):
                    ind = int(attr[1:-1])
                    subdata = subdata[ind]
                else:
                    subdata = subdata.get(attr)
            result[name] = subdata
        except:
            result[name] = None
    return result
```

From now you can use `json2` variables extractor in your tests:

```
variables:
  json2:
    myvar: varname
```

by running `ejpiaj-cli` with your module:

```
$ ejpiaj-cli test -s --module myfile mytest.yml
```

CHAPTER 5

Assertions

Assertions are used to check extracted variables against your tests.

5.1 Builtin assertions

5.1.1 equals / notequals

Example:

```
assertions:
  response:
    - 'status_code equals 200'
    - 'status_code notequals 500'
```

5.1.2 in / notin

Example:

```
assertions:
  response:
    - 'status_code in 200,301,302'
    - 'status_code notin 404,500'
```

5.1.3 empty / notempty

Example:

```
assertions:
  response:
    - 'contentText empty'
    - 'contentText notempty'
```

5.1.4 contains / notcontains

Example:

```
assertions:
  response:
    - 'contentText contains Hello'
    - 'contentText notcontains World'
```

5.2 Custom assertions

You can easily create your own assertions:

```
from ejpiaj.decorators import assertion

@assertion('false')
def equals_assertion(value):
    return value == False
```

From now you can use `false` assertion in your tests:

```
assertions:
  response:
    - 'status_code false'
```

by running `ejpiaj-cli` with your module:

```
$ ejpiaj-cli test -s --module myfile mytest.yml
```

CHAPTER 6

Examples

6.1 Post body

Example:

```
requests:
  001_token_post:
    method: post
    url: https://example.com/webapi/v1/token
    headers: ~
    url_params:
      lang: pl
    form_params: ~
    body: '{"username":"user", "password":"bestpass"}'

    assertions:
      response:
        - 'status_code in 200'
    variables:
      json:
        token: token
```

6.2 Authenticate by header

Example:

```
requests:
  001_get_token:
    method: post
    url: http://localhost:8000/api/v10/api-token-auth/
    form_params:
      username: apitest
      password: apitest
    variables:
      json:
        token: token
    assertions:
      response:
        - 'status_code equals 200'
      json:
        - 'token notempty'

  002_get_users:
    method: get
    url: http://localhost:8000/api/v10/users/
    headers:
      Authorization: Token {{token}}
    assertions:
      response:
        - 'status_code equals 200'
```

CHAPTER 7

ejpiaj

7.1 ejpiaj package

7.1.1 Submodules

7.1.2 ejpiaj.assertions module

ejpiaj.assertions.**contains_assertion**(*value*, *params*)

ejpiaj.assertions.**empty_assertion**(*value*)

ejpiaj.assertions.**equals_assertion**(*value*, *params*)

ejpiaj.assertions.**in_assertion**(*value*, *params*)

ejpiaj.assertions.**notcontains_assertion**(*value*, *params*)

ejpiaj.assertions.**notempty_assertion**(*value*)

ejpiaj.assertions.**notequals_assertion**(*value*, *params*)

ejpiaj.assertions.**notin_assertion**(*value*, *params*)

7.1.3 ejpiaj.core module

ejpiaj.core.**check_assertion**(*expression*, *value*)

ejpiaj.core.**test_request**(*request*, *variables*)

7.1.4 ejpiaj.decorators module

ejpiaj.decorators.**assertion**(*key*)

ejpiaj.decorators.**variable_extractor**(*key*)

7.1.5 ejpiaj.parsers module

7.1.6 ejpiaj.registry module

exception `ejpiaj.registry.`**`UnregisteredAssertion`**

Bases: `exceptions.Exception`

exception `ejpiaj.registry.`**`UnregisteredVariablesExtractor`**

Bases: `exceptions.Exception`

`ejpiaj.registry.`**`get_assertion`**(*name*)

`ejpiaj.registry.`**`get_assertions`**()

`ejpiaj.registry.`**`get_variables_extractor`**(*name*)

`ejpiaj.registry.`**`get_variables_extractors`**()

`ejpiaj.registry.`**`register_assertion`**(*name*, *assertion*)

`ejpiaj.registry.`**`register_variables_extractor`**(*name*, *extractor*)

`ejpiaj.registry.`**`unregister_assertion`**(*name*)

`ejpiaj.registry.`**`unregister_variables_extractor`**(*name*)

7.1.7 ejpiaj.runner module

7.1.8 ejpiaj.variable_extractor module

`ejpiaj.variable_extractor.`**`json_variables_extractor`**(*response*, *variables*)

Extracts variables from json response.content.

Variables path are written using ‘dot’ access and index access to lists f.i.:

some.path.to.list.[0] [1].dict.access.later

`ejpiaj.variable_extractor.`**`response_variables_extractor`**(*response*, *variables*)

7.1.9 Module contents

CHAPTER 8

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

8.1 Types of Contributions

8.1.1 Report Bugs

Report bugs at https://github.com/onjin/ejpiaj/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

8.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

8.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

8.1.4 Write Documentation

ejpiaj could always use more documentation, whether as part of the official ejpiaj docs, in docstrings, or even on the web in blog posts, articles, and such.

8.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/onjin/ejpiaj/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

8.2 Get Started!

Ready to contribute? Here's how to set up *ejpiaj* for local development.

1. Fork the *ejpiaj* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/ejpiaj.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv ejpiaj
$ cd ejpiaj/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

8.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/onjin/ejpiaj/pull_requests and make sure that the tests pass for all supported Python versions.

CHAPTER 9

Credits

9.1 Development Lead

- Marek Wywiał <onjinx@gmail.com>

9.2 Contributors

None yet. Why not be the first?

CHAPTER 10

History

10.1 0.4.4 (2015-01-21)

- Fixed unicode errors while reading str/int parameters

10.2 0.4.3 (2014-09-10)

- Fixed compatibility with python 3.3

10.3 0.4.2 (2014-09-10)

- Fixed context replacing at variables replacing

10.4 0.4.1 (2014-09-10)

- Fixed non unicode extracted variables

10.5 0.4.0 (2014-03-11)

- Added support for xml and json files

10.6 0.3.3 (2014-03-06)

- Added -e / –env option to `ejpiaj-cli` to pass initial variables

- Added -q / –quiet option to `ejpiaj-cli` to quiet output

10.7 0.3.2 (2014-02-17)

- Added -s option to `ejpiaj-cli` to display extracted variables

10.8 0.3.1 (2014-02-17)

- Fixed loading custom module from current directory

10.9 0.3.0 (2014-02-16)

- Added support to load own module with custom assertions and variable extractors using `ejpiaj-cli` tool

10.10 0.2.3 (2014-02-10)

- Fixed tests order (alphabetical)

10.11 0.2.2 (2014-02-10)

- Fixed variable substiution for multi varaibles
- Added variable substitution in ‘url’

10.12 0.2.1 (2014-02-07)

- Fixed variables substitution if variable is None

10.13 0.2.0 (2014-02-07)

- Added support for form_params and headers

10.14 0.1.0 (2014-02-01)

- First release on PyPI.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index