play requests Documentation

Release 0.0.1

Davide Moro

Jan 10, 2018

Contents

Contents:

CHAPTER 1

play requests

pytest-play plugin driving the famous python requests library for making HTTP calls

More info and examples on:

- pytest-play, documentation
- cookiecutter-qa, see `pytest-play` in action with a working example if you want to start hacking

1.1 Features

This pytest-play command provider let you drive a Python requests HTTP library using a json configuration file containing a set of pytest-play commands.

you can see a pytest-play script powered by a command provided by the play_requests plugin:

```
{
    "steps": [{
        "provider": "play_requests",
        "type": "GET",
        "assert": "'pytest-play' in response.json()",
        "url": "https://www.google.it/complete/search",
        "parameters": {
            "headers": {
                "Host": "www.google.it",
                "User-Agent": "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:57.0)␣
↪Gecko/20100101 Firefox/57.0",
                "Accept": "*/*",
                "Accept-Language": "en-US,en;q=0.5",
                "Referer": "https://www.google.it/",
                "Connection": "keep-alive"
            },
            "params": [
                ["client", "psy-ab"],
                ["hl", "it"],
                ["gs_rn", "64"],
                ["gs_ri", "psy-ab"],
                ["gs_mss", "pytest-"],
                ["cp", "11"],
                ["gs_id", "172"],
                ["q", "pytest-play"],
                ["xhr", "t"]
            ],
            "timeout": 2.5
        }
    }]
}
```

The above example:

- performs a GET call to https://www.google.it/complete/search?client=psy-ab&hl=it&... with the provided headers, a timeout (if it takes more than 2.5 seconds a timeout exception will be raised) and an assertion expression that verifies that the response meets the expected value

play_requests supports all the HTTP verbs supported by the requests library:

- OPTIONS
- HEAD
- GET
- POST
- PUT
- PATCH
- DELETE

NOTES: cookies and auth implementations supported by requests are not yet implemented because this package is still under development.

You'll find other play_requests command examples in the following sections.

1.1.1 Condition

```
{
    "provider": "play_requests",
    "type": "POST",
    "url": "http://something/1",
    "condition": "1 > 0",
    "parameters": {
        "json": {
            "foo": "bar",
        },
        "timeout": 2.5
    }
}
```

the `condition` option let you execute Python expressions thanks to the play_python plugin.

Other `condition` examples:

- `"$myvar" == 'dev'`

- `variables["myvar"] == 'dev'`

1.1.2 Upload files

Post a csv file:

```
{"provider": "play_requests",
 "type": "POST",
 "url": "http://something/1",
 "parameters": {
     "files": {
         "filecsv": [
             "report.csv",
             "some,data"
             ]
         }
     }
 }
```

Post a csv file with custom headers:

```
{"provider": "play_requests",
 "type": "POST",
 "url": "http://something/1",
 "parameters": {
     "files": {
         "filecsv": [
             "report.csv",
             "some,data",
             "application/csv",
             {"Expires": "0"}
         ]}
     }
 }
```

Post a file providing the path:

```
{
     "provider": "play_requests",
     "type": "POST",
     "url": "http://something/1",
     "parameters": {
         "files": {
             "filecsv": [
                 "file.csv",
                 "path:$base_path/file.csv"
             ]
         }
    }
}
```

assuming that you have a `$base_path` variable.

1.1.3 Save the response to a variable

You can save a response elaboration to a pytest-play variable and reuse in the following commands:

```
{
    "provider": "play_requests",
    "type": "POST",
    "url": "http://something/1",
    "variable": "myvar",
    "variable_expression": "response.json()",
    "assertion": "variables["myvar"]["status"] == "ok"",
    "parameters": {
        "json": {
            "foo": "bar",
            },
        "timeout": 2.5
        }
    }
```

It the endpoint returns a non JSON response, use `response.text` instead.

1.1.4 Default payload

If all your requests have a common payload it might be annoying but thanks to play_requests you can avoid repetitions.

You can set variables in many ways programatically using the pytest-play execute command or execute commands. You can also update variables using the play_python `exec` command:

```
{
    "steps": [{
        "provider": "python",
        "type": "store_variable",
        "name": "bearer",
        "expression": "'BEARER'"
    },
    {
        "provider": "python",
        "type": "exec",
        "expression": "variables.update({'play_requests': {'parameters': {'headers': {
↪'Authorization': '$bearer'}}}})"
    },
    {
         "provider": "play_requests",
         "type": "GET",
         "url": "$base_url"
    }
}
```

and all the following HTTP calls will be performed with the authorization bearer provided in the default payload.

We suggest to define variables and update play_requests defaults programmatically, use json only for trivial examples.

Merging rules:

- if a play_requests command provides any other header value, the resulting HTTP call will be performed with merged header values (eg: `Authorization` + `Host`)
- if a play_requests command provides a conflicting header value or any other default option, the `Authorization` header provided by the command will win and it will override just for the current call the default conflicting header value

1.1.5 Assert response status code

```
{
    "provider": "play_requests",
    "type": "POST",
    "url": "http://something/1",
    "variable": "myvar",
    "variable_expression": "response.json()",
    "assertion": "response.status_code == 200",
    "parameters": {
        "json": {
            "foo": "bar",
            }
        }
    }
```

of if you want you can use the expression `response.raise_for_status()` instead of checking the exact match of status code.

The `raise_for_status` call will raise an `HTTPError` if the `HTTP` request returned an unsuccessful status code.

1.1.6 Redirections

By default requests will perform location redirection for all verbs except HEAD:

- http://docs.python-requests.org/en/master/user/quickstart/#redirection-and-history

You can disable or enable redirects playing with the `allow_redirects` option:

```
{
    "provider": "play_requests",
    "type": "POST",
    "url": "http://something/1",
    "variable": "myvar",
    "variable_expression": "response.json()",
    "assertion": "response.status_code == 200",
    "parameters": {
        "allow_redirects": false,
        "json": {
            "foo": "bar",
            }
        }
    }
```

1.2 Twitter

`pytest-play` tweets happens here:

- @davidemoro

1.3 Credits

This package was created with Cookiecutter and the cookiecutter-play-plugin (based on audreyr/cookiecutter-pypackage project template).

CHAPTER 2

Installation

2.1 Stable release

To install play requests, run this command in your terminal:

```
$ pip install play_requests
```

This is the preferred method to install play requests, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for play requests can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/tierratelematics/play_requests
```

Or download the tarball:

```
$ curl  -OL https://github.com/tierratelematics/play_requests/tarball/master
```

Once you have a copy of the source, you can install it with:

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

CHAPTER 3

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:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/tierratelematics/play_requests/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.

3.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

3.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

3.1.4 Write Documentation

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

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/tierratelematics/play_requests/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 :)

3.2 Get Started!

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

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

   ```
   $ git clone git@github.com:your_name_here/play_requests.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 play_requests
   $ cd play_requests/
   $ 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:

   ```
   $ flake8 play_requests tests
   $ python setup.py test or py.test
   $ 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.

3.3 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, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/tierratelematics/play_requests/pull_requests and make sure that the tests pass for all supported Python versions.

3.4 Tips

To run a subset of tests:

```
$ py.test tests.test_play_requests
```

CHAPTER 4

CHANGES

4.1 0.0.1 (unreleased)

- First release

CHAPTER 5

Indices and tables

- genindex
- modindex
- search