assertio

Release 1.4.1

Aldo Vázquez

May 04, 2022

CONTENTS

CHAPTER

ONE

INSTALLATION

To install assertio you will need only pip, nothing else is required

```
$ pip install assertio
```

CHAPTER

TWO

BASIC USAGE

2.1 Assertio Configuration

In order to start writing your test cases, there is some configuration that you must do. `assertio` is flexible enough to get som necessary data from different sources.

2.1.1 Environment Variables

If there is not any settings file under your project's root, `assertio` will look for the following environment variables

```
ASSERTIO_BASE_URL     # Define the API to be tested url
ASSERTIO_LOGFILE      # Defaulted to assertio.log will recieve the execution logs
ASSERTIO_PAYLOADS_DIR # Defaulted to features/payloads will define the local directory
                      # to .json files that can be used while executing requests
```

`ASSERTIO_BASE_URL` **must** exist if there is not a settings file, otherwise the tests won't run.

2.1.2 assertio.json file

`assertio` Will look for this settings file in your poject's root, the structure of this file should be like this

```
{
    "base_url": "your-api-url",
    "logfile": "assertio.log",
    "payloads_dir": "features/payloads"
}
```

If any key is missing, `assertio` will try to replace it with the environment variables mentioned above, using the same default values.

2.1.3 assertio.yaml file

If you are a fan of `yaml` files, `assertio` can be setup using this settings file in your poject's root, the structure of this file should be like this

```
base_url: your-api-url
logfile: assertio.log
payloads_dir: features/payloads
```

If any key is missing, `assertio` will try to replace it with the environment variables mentioned above, using the same default values.

2.2 Runners

`assertio` has a Runner class in order to create child classes to hold all your test cases, to create a new Runner it is important to add a new runner file within `features/runners` folder.

Your new child class name **must** end with *Runner* suffix, otherwise it wont be reachable from `assertio` CLI.

```
# features/runners/example.py
from asserito import Runner

class BookRunner(Runner):

    def test_get_books(self):
        ...

    def test_post_book(self):
        ...

    def test_patch_book(self):
        ...

    def test_delete_book(self):
        ...
```

It's worth to mention that all your test cases **must** start with **test** prefix in order to be executed.

2.2.1 Start a set of tests

Once you have defined your *Runners* run them is as easy as just invoke the `.start()` method.

```
# main.py
from features.runners.book import BookRunner
from features.runners.author import AuthorRunner

if __name__ == "__main__":
    BookRunner().start()
    AuthorRunner().start()
```

2.3 Creating Request() chains

`assertio` also has a Request class with all the necessary methods to

- Setup http request body, headers, method, endpoint
- Perform a http request
- Assert a lot of data of the http response

This can be done using a *chain-like syntax* which will make the test easier to read, understand and modify.

```
# features/runners/example.py
from asserito import Runner

class BookRunner(Runner):

    def test_get_books(self):
        Request()\
            .to("/api/v1/books/")
            .with_method("GET")\
            .perform()\
            .assert_http_ok()
```

This test above will execute the next steps one by one:

- Creates a new request
- Sets the request endpoint to `/api/v1/books/`
- Sets request method to `GET`
- Executes the requets and store the response
- Asserts that response status code equals `HTTP OK 200`

2.3.1 Assertions

But, you can assert many more information about your request, let's take a look to a little more complex example.

```
# features/runners/example.py
from asserito import Runner

class BookRunner(Runner):

    def test_get_books(self):
        Request()\
            .to("/api/v1/books/")
            .with_method("GET")\
            .perform()\
            .assert_http_ok()\
            .assert_response_field("results")\
            .is_not_empty()
```

Now, this Request chain will execute the same steps as above and will assert that the response's json body field `results` is not an empty array.

All the assertions **must** be invoked after `.perform()`.

Otherwise an exception will be raised.

2.3.2 Preconditions

So far, we have been talking about how to assert information from a http response, but your Request object allows you to do more than that.

Let's take a look to a different example using more request preconditions.

```
# features/runners/example.py
from asserito import Runner

DEFAULT_HEADERS = {"Content-Type": "application/json"}
BOOK_PAYLOAD = {
    "id": 144,
    "title": "The Divine Comedy",
    "author": {
        "id": 12,
        "name": "Dante Alighieri",
        "nationality": "Italian"
    },
    "year": 1472
}

class BookRunner(Runner):

    def test_post_book(self):
        Request()\
            .to("/api/v1/books/")
            .with_method("POST")\
            .with_headers(DEFAULT_HEADERS)\
            .with_body(BOOK_PAYLOAD)\
            .perform()\
            .assert_http_created()\
            .assert_response_field("author.name")\
            .equals("Dante Alighieri")
```

This test above will execute the next steps one by one:

- Creates a new request
- Sets the request endpoint to `/api/v1/books/`
- Sets request method to `POST`
- Sets request headers to `DEFAULT_HEADERS` dictionary
- Sets request payload to `BOOK_PAYLOAD` dictionary
- Asserts that response's status code equals `HTTP CREATED 201`
- Asserts that response's json body field `author.name` field equals "Dante Alighieri"

As you might have guessed, all the preconditions **must** be invoked before invoking `.perform()`.

2.3.3 Quick tips

Skipping one line

When defining a new request `method` can be added to `Request()` initial line.

```
# features/runners/example.py
from asserito import Runner

class BookRunner(Runner):

    def test_get_books(self):
        Request("GET")\
            .to("/api/v1/books/")
            .perform()\
            .assert_http_ok()
```

Will work exactly the same as the previous request, this might help you to save one chain member!

Using data from .json files.

Sometimes you might need to add huge payloads to your request, full of static data, therefore, `assertio` allows you to load a json file when using `.with_body()`, just add the name as parameter! As long as your .json file exists within `features/payloads` `assertio` will find it! Let's see an example

Listing 1: features/payloads/api/book.json

```
{
    "id": 144,
    "title": "The Divine Comedy",
    "author": {
        "id": 12,
        "name": "Dante Alighieri",
        "nationality": "Italian"
    },
    "year": 1472
}
```

```
class BookRunner(Runner):

    def test_post_book(self):
        Request("POST")\
            .to("/api/v1/books/")
            .with_headers(DEFAULT_HEADERS)\
            .with_body("api/book.json")\
            .perform()\
            .assert_http_created()\
            .assert_response_field("author.name")\
            .equals("Dante Alighieri")
```

Should work just the same as the previous POST example!

2.4 Assertio CLI (WIP)

`assertio` has a CLI as well!

Once you have defined your runners under `features/runners`, you can run all of them using:

Listing 2: Within your virtualenv

```
$ assertio
```

Runnig this command will execute all the tests for each class with the `Runner` suffix, but you can always choose just one using the `--run` flag

```
$ assertio --run BookRunner
```

Will run only the BookRunner defined.

Using the `assertio` command will execute runners using the default settings mentioned above, looking for either `assertio.json` or *assertio.yaml* files in your project's root or using the defined environment variables.

But you can also specify wich settings file to use with the `--settings` this might be useful when you work with more that one API environment, you can easily switch between testing dev environment API to uat environment API!

```
$ assertio --run BookRunner --settings=dev.json
$ assertio --run BookRunner --settings=uat.json
```

Will execute the same test set just changing the url and other settings already metioned!

Hope this basic usasge was useful for you to start testing your APIs!

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search