Python Ariadne Documentation

Release 0.1.0a0

Tomáš Ehrlich

May 23, 2015

Contents

Danger: This library is early alpha without stable API.

Contents:

CHAPTER 1

Tutorial

1.1 Structure of tests

Going from the top to the bottom, tests are organized into **scenarios**. These scenarios are very similar to the ones from BDD [1]. They describe *how user interacts with an application*.

For example one scenario for eshop could be: "User puts an item into shopping cart." This scenario would proably contain other not directly related actions like signing up into website which brings us to the **stories**.

Stories are group of related actions. In example above it would be: sign up and put item into shopping cart. More complex test would consist of sign up, putting item into cart, sign off and sign up again – to check that content of shopping cart is persistent across logins.

Last at the most atomic part are **actions**. These actions describe single event like visiting a website, filling the form, etc.

Actions, stories and scenarios itself doesn't test anything. Think about scenarios like of *Ariadne's threads* – they describe all possible ways through an application.

The next step is to define **checks** which will be run during execution of actions. These checks can perform anything from functional tests to performance tests. In case of web applications, possible checks are:

- does website work? (Does it returns HTTP 200)
- does it take too long to render?
- are there any duplicate or unwanted SQL queries
- are there all translations?
- is HTML rendered correctly (compared to last run)

1.2 Writing scenarios

Let's start with testing simple story – user wants to log in:

1. User come to login page
2. They fill in the username and password and submit the form

1.2.1 Actions

The most atomic part of stories are actions. In this case, each line is one action:

1. `actions.Visit` – go to page specified by URL
2. `actions.FillForm` – fill in form and submit it

We can use predefined actions (see Reference) and initialize them with custom parameters:

```
from ariadne import actions

visit_login = actions.Visit(url='/login')

credentials = {
  'username': 'admin',
  'password': 'secret',
}
fill_credentials = actions.FillForm(data=credentials, submit='#btn-submit')
```

1.2.2 Stories

Once we have `actions` defined, we can groups them into stories and specify in which order these actions should be executed:

```
from ariadne import stories

login_process = stories.Simple([
  visit_login,
  fill_credentials
])
```

`stories.Simple` is just wrapper around set of actions, which executes them serially one after another.

1.2.3 Scenarios

The last step is to define scenarios. It usually consists of several stories, but sometimes is scenario as simple as story itself. We can use simple shortcut in such case:

```
user_login = login_process.as_scenario()
```

The main difference between scenario and stories is context. Each scenario is executed with isolated context, while stories pass context from one to another. In unittest scenario would be single test case.

1.2.4 Context and config

Now we are ready to run our scenario in browser.

We mentioned context in previous section. Context is basically a dictionary of data which is passed through scenario and contains *persistent* object. Web browser is example of such object – we need to keep state of our browser during whole scenario.

Context are constructed using *context preprocessors* at the beginning of each scenario. *Context preprocessors* are simple functions which modify context, eg. add key `browser` with instantiated web browser.

We use config to define used *context preprocessors*:

```
from ariadne.config import BaseConfig
from ariadne.context import browsers

class ExampleConfig(BaseConfig):
    def context_preprocessors(self):
        return [
            browsers.Splinter('firefox')
        ]
```

1.2.5 Let's run it already!

Alright. Enough talk, let's fight. We have scenarios and basic configuration, the last thing we need is a runner. In this case we can use very simple one:

```
from ariadne.runners import SimpleRunner


runner = SimpleRunner(config=ExampleConfig)
runner.add(user_login)

if __name__ == '__main__':
    runner.run()
```

As simple as it could be, it only run one scenario and output results to stdout.

Try this file execute in Python, you should see following output:

```
$ python example.py
Scenario 1 of 1: login_process
- visit http://localhost/login ... OK
- fill form ... OK
```

That's it. You might have noticed that we haven't tested anything and you're 100% right. We've just opened browser and filled form. If we want to test something, we need to write *checks*.

CHAPTER 2

Reference

2.1 Actions

class ariadne.actions.**Action**
Base class for all actions

class ariadne.actions.**Visit**(*url=None*)
Visit an URL in browser

get_url()
Return target URL. May be predefined in subclass with dynamic behaviour. :return: URL to visit

run(*context*)
Run action in context

Parameters **context** –

Returns (optional) context dictionary

2.2 Context

2.2.1 Browsers

ariadne.context.browsers.**splinter**(*driver_name=u'phantomjs'*, *context=None*)
Add splinter browser instance to context.

Parameters **driver_name** – The name of browser to use: phantomjs (default),

firefox, chrome :param context: (optional) context to fill or manipulate :return: new context

2.3 Scenarios

class ariadne.scenarios.**Scenario**(*stories*)
Base class for all scenarios

run()
Run all stories in scenario. Pass shared context through whole scenario.

Returns context dictionary

2.4 Stories

class ariadne.stories.**Simple**(*actions*)
Run actions in series

class ariadne.stories.**Story**(*actions*)
Base class for all stories

as_scenario()
Convert story into simple scenario with single story only. :return: Scenario

run(*context*)
Run all actions in story and pass context.

Parameters **context** –

Returns context dictionary

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

a

A

G

R

S

V