

---

# **Django SubUI Tests Documentation**

***Release 0.2.2***

**Miroslav Shubernetskiy**

**Jul 28, 2017**



---

## Contents

---

<b>1</b>	<b>subui package</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Advanced Use-Cases . . . . .	3
1.3	Submodules . . . . .	4
<b>2</b>	<b>Django SubUI Tests</b>	<b>13</b>
<b>3</b>	<b>Installing</b>	<b>15</b>
<b>4</b>	<b>Testing</b>	<b>17</b>
	<b>Python Module Index</b>	<b>19</b>



SubUI is a framework to ease the pain of writing and running integration tests. The “SubUI” part means that it is not meant to test any of the UI (like html validation) but instead allows to make complete workflow server integration tests.

## Introduction

The framework consists of 3 main components:

### 1. SubUI test runner

This is the interface layer of the SubUI framework. In other words, test methods will instantiate the runner and use it to interact with the SubUI integration tests. Primary job of a SubUI test runner is to execute test steps (described below) in correct order and maintain state between steps if necessary.

---

**Note:** Even though it is called test runner, it does not replace or even relate to `nosetests` test runner.

---

### 2. Test Steps

In the most part, this framework is meant to make integration tests for complete workflows (e.g. go to page 1 -> submit form and assert redirect to page 2 -> go to page 2). A test step is a self-contained piece of the complete workflow like “go to page 1”. Combinations of multiple steps then make up the workflow. Since steps are independent, they should know how to complete their task (e.g. submit form via `POST`) and validate that they got expected result from the server. To allow flexible validation, they themselves do not validate anything but use validators (described below) to inspect server response in very similar way to how Django Form Field uses validators to verify user-input.

### 3. Validator

Validator’s task is to make assertions about the response from the server. All validators are pretty straight forwards like assert that the response status code is `Redirect - 302` or that redirect header `Location` is returned. More complex assertions can be made by either using multiple validators in each test step or make more complex validator via multiple class inheritance.

## Example

An example should show some of the advantages of using this framework for a hypothetical todo application:

```
1  # define steps
2  class GoToLogin(TestStep):
3      url_name = 'login'
4      request_method = 'get'
5      validators = [StatusCodeOkValidator]
6
7  class SubmitLoginForm(TestStep):
8      url_name = 'form'
9      validators = [RedirectToRouteValidator(expected_route_name='list')]
10     data = {
11         'username': 'user',
12         'password': 'password',
13     }
14
15  class GoToList(TestStep):
16      url_name = 'list'
17      request_method = 'get'
18      validators = [StatusCodeOkValidator]
19
20  class CreateToDo(TestStep):
21      url_name = 'create'
22      validators = [RedirectToRouteValidator(expected_route_name='list')]
23      data = {
24          'notes': 'need to finish something',
25          'due date': '2015-01-01',
26      }
27
28  # integration tests
29  class TestWorkflow(TestCase):
30      def test_login_and_create_todo(self):
31          runner = SubUITestRunner(
32              OrderedDict((
33                  ('login', GoToLogin),
34                  ('login_submit', SubmitLoginForm),
35                  ('list1', GoToList),
36                  ('create', CreateToDo),
37                  ('list2', GoToList),
38              )),
39              client=self.client
40          ).run()
41
42          self.assertNotContains(runner.steps['list1'].response,
43                                'need to finish something')
44          self.assertContains(runner.steps['list2'].response,
45                              'need to finish something')
46
47      def test_just_create(self):
48          data = {
49              'notes': 'other task here to complete',
50              'due date': '2015-01-01',
51          }
52          runner = SubUITestRunner(
53              OrderedDict((
54                  ('list1', GoToList),
```

```

55         ('create', CreateToDo(data=data)),
56         ('list2', GoToList),
57     )),
58     client=self.client
59 ).run()
60
61 self.assertNotContains(runner.steps['list1'].response,
62                        'other task here to complete')
63 self.assertContains(runner.steps['list2'].response,
64                    'other task here to complete')

```

some useful things to note about what happened above:

- Reuse of test steps. Since each step is self-contained, they can be combined in different ways to make different integration tests. They can even be reused multiple times within the same integration test.
- Step attributes can easily be overwritten if need to like in `test_just_create` test method - `CreateToDo`'s data is overwritten to post different values.
- Assertions on steps can be performed outside of the test runner. After steps are executed, all steps can be accessed via `runner.steps` attribute which will be an instance of `collections.OrderedDict`.
- `subui.validators.RedirectToRouteValidator` is combined validator via multiple inheritance which verifies that the response status code is 302 - Redirect; Location response header is present; and that the page redirects to a particular route as determined by Django's `resolve`.

## Advanced Use-Cases

More advanced things can be accomplished with the framework. In the previous example, all steps had a fixed url without any parameters. This example will use state to pass information between steps:

```

1  # login returns redirect to user profile with user id
2  class LoginStep(Step):
3      url_name = 'login'
4      validators = [RedirectToRouteValidator(expected_route_name='profile')]
5      data = {
6          'username': 'username',
7          'password': 'password',
8      }
9
10     def post_test_response(self):
11         # extract user kwargs from redirect location
12         resolved = resolve(self.response['Location'])
13         self.state.push({
14             'url_kwargs': resolved.kwargs,
15         })
16
17     class ProfileStep(StatefulUrlParamsTestStep):
18         url_name = 'profile' # requires url kwargs of username
19         request_method = 'get'
20         validators = [StatusCodeOkValidator]
21
22     class EditProfileStep(StatefulUrlParamsTestStep):
23         url_name = 'edit_profile' # requires url kwarg of username
24         validators = [StatusCodeOkValidator]
25
26     class TestWorkflow(TestCase):

```

```
27 def test_login_and_edit(self):
28     data = {
29         'username': 'otheruser',
30         'password': 'otherpassword',
31     }
32     runner = SubUITestRunner(
33         [
34             LoginStep,                # 0
35             ProfileStep,              # 1
36             EditProfileStep(data=data), # 2
37             ProfileStep,              # 3
38         ],
39         client=self.client
40     ).run()
41
42     self.assertContains(runner.steps[3].response,
43                         'otheruser')
```

some notes about what happened:

- LoginStep uses a hook `subui.step.TestStep.post_test_response()` to add data to a state. Since state is global for all steps within test runner, other steps can access it.
- ProfileStep and EditProfileStep subclass `subui.step.StatefulUrlParamsTestStep` which uses state to get url args and kwargs.
- When using `resolve` in `post_test_response`, there is no need to do `try: except Resolver404` since that will be executed after validator verifications hence it is guaranteed that the url will resolve without issues.
- Steps are provided as `list()` instead of `collections.OrderedDict`. Test runner automatically converts the steps into `collections.OrderedDict` with keys as indexes which allows to type test runner a bit faster ;-)

## Submodules

### subui.step module

**class** `subui.step.StatefulUrlParamsTestStep` (*\*\*kwargs*)

Bases: `subui.step.TestStep`

Test step same as `TestStep` except it references `url_args` and `url_kwargs` from the state.

Having url computed from the state, allows for a particular step to change `url_args` or `url_kwargs` hence future steps will fetch different resources.

**get\_url\_args()**

Get URL args for Django's reverse

Similar to `TestStep.get_url_args()` except url args are retrieved by default from state and if not available get args from class attribute.

**Returns** tuple of url\_args

**get\_url\_kwargs()**

Get URL kwargs for Django's reverse



Similar to `TestStep.get_url_kwargs()` except url args are retrieved by default from state and if not available get kwargs from class attribute.

**Returns** dict of url\_kwargs

```
class subui.step.TestStep(*kwargs)
    Bases: object
```

Test step for `subui.test_runner.SubUITestRunner`.

The step is responsible for executing a self-contained task such as submitting a form to a particular URL and then make assertions regarding the server response.

#### Variables

- **`TestStep.test`** (`unittest.TestCase`) – Test class instance with which validators are going to run all their assertions with. By default it is an instance of `unittest.TestCase` however can be changed to any other class to add additional assertion methods.
- **`url_name`** (`str`) – Name of the url as defined in `urls.py` by which the URL is going to be calculated.
- **`url_args`** (`tuple`) – URL args to be used while calculating the URL using Django's reverse.
- **`url_kwargs`** (`dict`) – URL kwargs to be used while calculating the URL using Django's reverse.
- **`request_method`** (`str`) – HTTP method to use for the request. Default is "post"
- **`urlconf`** (`str`) – Django URL Configuration.
- **`content_type`** (`str`) – Content-Type of the request.
- **`overridden_settings`** (`dict`) – Dictionary of settings to be overridden for the test request.
- **`data`** (`dict`) – Data to be sent to the server in the request
- **`state`** (`pycontext.context.Context`) – Reference to a global state from the test runner.
- **`TestStep.validators`** (`list`) – List of response validators
- **`response`** – Server response for the made request. This attribute is only available after `request()` is called.

**Parameters** **`kwargs`** (`dict`) – A dictionary of values which will overwrite any instance attributes. This allows to pass additional data to the test step without necessarily subclassing and manually instantiating step instance.

**`content_type`** = None

**`data`** = None

**`get_content_type()`**

Get `content_type` which will be used when making the test request.

By default this returns `content_type`, if defined, else empty string.

**Return type** str

**`get_override_settings()`**

Get `overridden_settings` which will be used to decorate the request with the defined settings to be overridden.

By default this returns *overriden\_settings*, if defined, else empty dict

**Return type** dict

**get\_request\_data** (*data=None*)

Get data dict to be sent to the server.

**Parameters** *data* – Data to be used while sending server request. If not defined, *data* is returned.

**Return type** dict

**get\_request\_kwargs** ()

Get kwargs to be passed to the *client*.

By default this returns dict of format:

```
{
    'path': ...,
    'data': ...
}
```

Can be overwritten in case additional parameters need to be passed to the client to make the request.

**Return type** dict

**get\_url** ()

Compute the URL to request using Django's *reverse*.

*Reverse* is called using *url\_name*, *get\_url\_args* () and *get\_url\_kwargs* () .

**Return type** str

**get\_url\_args** ()

Get *url\_args* which will be used to compute the URL using *reverse*.

By default this returns *url\_args*, if defined, else empty tuple.

**Return type** tuple

**get\_url\_kwargs** ()

Get *url\_kwargs* which will be used to compute the URL using *reverse*.

By default this returns *url\_kwargs*, if defined, else empty dict.

**Return type** dict

**get\_urlconf** ()

Get *urlconf* which will be used to compute the URL using *reverse*

By default this returns *urlconf*, if defined, else *None*

**Return type** str

**get\_validators** ()

Get all validators.

By default returns *validators* however can be used as a hook to returns additional validators dynamically.

**Return type** list

**init** (*client, steps, step\_index, step\_key, state*)

Initialize the step with necessary values from the test runner.

**Parameters**

- **client** (*django.test.client.Client*) – Django test client to use to make server requests
- **steps** (*collections.OrderedDict*) – All steps from the test runner. This and step index allows to get previous and/or next steps.
- **step\_index** (*int*) – Index of the step within all steps test runner will execute.
- **step\_key** (*str*) – Key of the state of how it was provided to the test runner in case test step needs to reference other steps within the runner by their.
- **state** (*platform\_utils.utils.dt\_context.BaseContext*) – Global state reference from the test runner.

**next\_step**

Get previous step instance, if any.

**Return type** *TestStep*

**next\_steps**

Get *collections.OrderedDict* of next steps excluding itself, if any.

**Return type** *collections.OrderedDict*

**overridden\_settings = None****post\_request\_hook ()**

Hook which is executed after server request is sent.

**post\_test\_response ()**

Hook which is executed after validating the response.

**pre\_request\_hook ()**

Hook which is executed before server request is sent.

**pre\_test\_response ()**

Hook which is executed before validating the response.

**prev\_step**

Get previous step instance, if any.

**Return type** *TestStep*

**prev\_steps**

Get *collections.OrderedDict* of previous steps excluding itself, if any.

---

**Note:** The steps are returned in order of adjacency from the current step. For example (using list instead of *OrderedDict* in example):

```
> step = TestStep()
> step.steps = [0, 1, 2, 3, 4]
> step.step_index = 3
> step.prev_steps
[2, 1, 0]
```

---

**Return type** *collections.OrderedDict*

**request ()**

Make the server request. Server response is then saved in *request*.

Before making the request, `pre_request_hook()` is called and `post_request_hook()` is called after the request.

**Returns** server response

`request_method = u'post'`

`state = None`

`test = <unittest.case.TestCase testMethod=__init__>`

`test_response()`

Test the server response by looping over all validators as returned by `get_validators()`.

Before assertions, `pre_test_response()` is called and `post_test_response()` is called after assertions.

`url_args = None`

`url_kwargs = None`

`url_name = None`

`urlconf = None`

`validators = []`

## subui.test\_runner module

`class subui.test_runner.SubUITestRunner(steps, client, state=None, **kwargs)`

Bases: object

SubUI Test Runner.

This is the interface class of the SubUI framework. It runs all of the provided steps in the provided order. Since some steps might need state from previous executed steps, the runner maintains reference to a global state which it then passes to each step during execution.

### Variables

- **steps** (`collections.OrderedDict`) – Test steps to be executed
- **client** – Django test client to run tests
- **kwargs** (`dict`) – Additional kwargs given to class. These kwargs will be provided to each step when executed.
- **state** (`pycontext.context.Context`) – State to be shared between test step executions.

### Parameters

- **steps** (`list`, `tuple`, `collections.OrderedDict`) – Steps to be executed. Executed in the provided order hence need to be provided in order-maintaining data-structure.
- **client** (`django.test.client.Client`) – Django test Client to query server with
- **state** (`dict`) – State to be shared between step executions. Optional and by default is empty dict.
- **kwargs** – Additional kwargs to be passed to each step during initialization if it is not already provided as initialized object.

**run()**

Run the test runner.

This executes all steps in the order there are defined in `SubUITestRunner.steps`. Before each step is executed `subui.step.TestStep.init()` is called providing all the necessary attributes to execute the step like test client, steps, and state).

**Returns** Reference to the test runner

## subui.validators module

**class** `subui.validators.BaseValidator` (*test\_step=None, \*\*kwargs*)

Bases: `object`

Base validator which should be sub-classed to create custom validators.

**Variables** `BaseValidator.expected_attrs` (*tuple*) – Required attributes for the validator. `__check_improper_configuration()` will verify that all of these attributes are defined.

---

**Note:** Each validator should only define required attributes for itself. `__get_expected_attrs()` will automatically return required attributes from all current validator and base classes.

---

**Parameters** `test_step` – `subui.step.TestStep` instance which will be used to make assertions on.

---

**Note:** This parameter is not really required in the `__init__` because the same step will be passed in `test()` however is useful in `__init__` in case subclass validator needs to apply custom login according to values from the step.

---

**expected\_attrs = None**

**test** (*test\_step*)

Test the step's server response by making all the necessary assertions. This method by default saves the `test_step` parameter into `step` and validates the validator by using `__check_improper_configuration()`. All subclass validators should actually implement assertions in this method.

**Parameters** `test_step` – Test step

**class** `subui.validators.FormInitialDataValidator` (*test\_step=None, \*\*kwargs*)

Bases: `subui.validators.BaseValidator`

Validator checks that form in response has expected data in initial data.

**Variables**

- `initial_data_key` (*str*) – Expected initial data key to be present in form initial data
- `expected_initial_data_value` – Expected value initial value should be set to
- `context_data_form_name` (*str*) – Template context data key for form data
- `test_initial_value` (*bool*) – Test if the initial value matched expected value
- `test_initial_value_present` (*bool*) – Test if the initial value key is present in initial data

- `test_initial_value_not_none` (*bool*) – Test if the initial value is not None

```
context_data_form_name = u'form'
expected_attrs = (u'initial_data_key',)
expected_initial_data_value = None
initial_data_key = None
test (test_step)
test_initial_value = False
test_initial_value_not_none = True
test_initial_value_present = True
```

```
class subui.validators.HeaderContentTypeValidator (test_step=None, **kwargs)
    Bases: subui.validators.HeaderValidator
```

Validator to check that the expected “Content-Type” header is returned.

```
header_name = u'Content-Type'
```

```
class subui.validators.HeaderLocationValidator (test_step=None, **kwargs)
    Bases: subui.validators.HeaderValidator
```

Validator to check that the redirect “Location” header is returned

```
header_name = u'Location'
```

```
class subui.validators.HeaderValidator (test_step=None, **kwargs)
    Bases: subui.validators.BaseValidator
```

Validator which can check that a particular header is returned and that it is of particular value.

#### Variables

- `HeaderValidator.header_name` (*str*) – Name of the header which must be returned
- `expected_header` (*str*) – Expected header value to be returned
- `HeaderValidator.test_header_value` (*bool*) – Whether to test the header value or simply check its existence
- `test_contains_value` – Whether to test if the header value contains another value, or simply check equality to that value

```
expected_attrs = (u'header_name', u'expected_header')
```

```
expected_header = None
```

```
header_name = None
```

```
test (test_step)
```

Test the response returned with `:py:attr:header_name` header and that its value is equal to `:py:attr:expected_header` if `:py:attr:test_header_value` is True. If `:py:attr:test_contains_value` is True, header value will be tested to contain expected value.

```
test_contains_value = False
```

```
test_header_value = True
```

```
class subui.validators.RedirectToRouteValidator (test_step=None, **kwargs)
    Bases: subui.validators.StatusCodeRedirectValidator
```

Validator which also checks that the server returns a redirect to an expected Django route.

**Variables** `expected_route_name` (*str*) – Route name to which the server should redirect to

`expected_attrs = (u'expected_route_name',)`

`expected_route_name = None`

**test** (*test\_step*)

Test the response by additionally testing that the response redirects to an expected route as defined by `expected_route_name`.

**class** `subui.validators.ResponseContentContainsValidator` (*test\_step=None, \*\*kwargs*)

Bases: `subui.validators.StatusCodeOkValidator`

Validator which also checks that returned response content contains expect string.

**Variables** `expected_content` (*str*) – Expected string in the server response

`expected_attrs = (u'expected_content',)`

`expected_content = None`

**test** (*test\_step*)

Test the response by additionally testing that the response context contains expected string as defined by `expected_content`.

**class** `subui.validators.ResponseContentNotContainsValidator` (*test\_step=None, \*\*kwargs*)

Bases: `subui.validators.StatusCodeOkValidator`

Validator checks that returned response content does not contain unexpected string.

**Variables** `unexpected_content` (*str*) – Unexpected string in the server response

`expected_attrs = (u'unexpected_content',)`

**test** (*test\_step*)

Test the response by additionally testing that the response context does not contain the unexpected string as defined by `unexpected_content`.

`unexpected_content = None`

**class** `subui.validators.SessionDataValidator` (*test\_step=None, \*\*kwargs*)

Bases: `subui.validators.BaseValidator`

Validator which allows to verify the data in the session based on session key.

**Variables**

- `expected_session_key` (*str*) – Expected session key to be present in session
- `expected_session_secondary_keys` (*list*) – List of Expected session key to be present in session

`expected_attrs = (u'expected_session_key',)`

`expected_session_key = None`

`expected_session_secondary_keys = []`

**test** (*test\_step*)

Test that expected session key is present. If expected session data provided, ensure the expected session key data matches what is there currently.

**class** `subui.validators.StatusCodeCreatedValidator` (*test\_step=None*, *\*\*kwargs*)

Bases: `subui.validators.StatusCodeValidator`

Validator to check that the returned status code is created - 201

**expected\_status\_code** = 201

**class** `subui.validators.StatusCodeOkValidator` (*test\_step=None*, *\*\*kwargs*)

Bases: `subui.validators.StatusCodeValidator`

Validator to check that the returned status code is OK - 200

**expected\_status\_code** = 200

**class** `subui.validators.StatusCodeRedirectValidator` (*test\_step=None*, *\*\*kwargs*)

Bases: `subui.validators.HeaderLocationValidator`, `subui.validators.StatusCodeValidator`

Validator to check that the server returns a redirect with the “Location” header defined.

**expected\_status\_code** = 302

**test\_header\_value** = False

**class** `subui.validators.StatusCodeValidator` (*test\_step=None*, *\*\*kwargs*)

Bases: `subui.validators.BaseValidator`

Validator which allows to verify the returned server status code such as “OK-200” or “Redirect-302”, etc.

**Variables** `StatusCodeValidator.expected_status_code` (*int*) – Expected status code to be returned by the server

**expected\_attrs** = (u'expected\_status\_code',)

**expected\_status\_code** = None

**test** (*test\_step*)

Test that the response status code matched expected status code.



## CHAPTER 2

---

### Django SubUI Tests

---

Framework to make workflow server integration test suites

- Free software: MIT license
- GitHub: <https://github.com/dealertrack/django-subui-tests>
- Documentation: <http://django-subui-tests.readthedocs.io/>



## CHAPTER 3

---

### Installing

---

You can install `django-subui-tests` using `pip`:

```
$ pip install django-subui-tests
```



## CHAPTER 4

---

### Testing

---

To run the tests you need to install testing requirements first:

```
$ make install
```

Then to run tests, you can use `nosetests` or simply use Makefile command:

```
$ nosetests -sv  
# or  
$ make test
```

- `genindex`
- `modindex`
- `search`



### S

- `subui`, [1](#)
- `subui.step`, [4](#)
- `subui.test_runner`, [8](#)
- `subui.validators`, [9](#)





## B

BaseValidator (class in subui.validators), 9

## C

content\_type (subui.step.TestStep attribute), 5

context\_data\_form\_name  
(subui.validators.FormInitialDataValidator  
attribute), 10

## D

data (subui.step.TestStep attribute), 5

## E

expected\_attrs (subui.validators.BaseValidator attribute),  
9

expected\_attrs (subui.validators.FormInitialDataValidator  
attribute), 10

expected\_attrs (subui.validators.HeaderValidator at-  
tribute), 10

expected\_attrs (subui.validators.RedirectToRouteValidator  
attribute), 11

expected\_attrs (subui.validators.ResponseContentContainsValidator  
attribute), 11

expected\_attrs (subui.validators.ResponseContentNotContainsValidator  
attribute), 11

expected\_attrs (subui.validators.SessionDataValidator at-  
tribute), 11

expected\_attrs (subui.validators.StatusCodeValidator at-  
tribute), 12

expected\_content (subui.validators.ResponseContentContainsValidator  
attribute), 11

expected\_header (subui.validators.HeaderValidator at-  
tribute), 10

expected\_initial\_data\_value  
(subui.validators.FormInitialDataValidator  
attribute), 10

expected\_route\_name (subui.validators.RedirectToRouteValidator  
attribute), 11

expected\_session\_key (subui.validators.SessionDataValidator  
attribute), 11

expected\_session\_secondary\_keys  
(subui.validators.SessionDataValidator at-  
tribute), 11

expected\_status\_code (subui.validators.StatusCodeCreatedValidator  
attribute), 12

expected\_status\_code (subui.validators.StatusCodeOkValidator  
attribute), 12

expected\_status\_code (subui.validators.StatusCodeRedirectValidator  
attribute), 12

expected\_status\_code (subui.validators.StatusCodeValidator  
attribute), 12

## F

FormInitialDataValidator (class in subui.validators), 9

## G

get\_content\_type() (subui.step.TestStep method), 5

get\_override\_settings() (subui.step.TestStep method), 5

get\_request\_data() (subui.step.TestStep method), 6

get\_request\_kwargs() (subui.step.TestStep method), 6

get\_url() (subui.step.TestStep method), 6

get\_url\_args() (subui.step.StatefulUrlParamsTestStep  
method), 4

get\_url\_args() (subui.step.TestStep method), 6

get\_url\_kwargs() (subui.step.StatefulUrlParamsTestStep  
method), 4

get\_url\_kwargs() (subui.step.TestStep method), 6

get\_urlconf() (subui.step.TestStep method), 6

get\_validators() (subui.step.TestStep method), 6

## H

header\_name (subui.validators.HeaderContentTypeValidator  
attribute), 10

header\_name (subui.validators.HeaderLocationValidator  
attribute), 10

header\_name (subui.validators.HeaderValidator at-  
tribute), 10

HeaderContentTypeValidator (class in subui.validators), 10

HeaderLocationValidator (class in subui.validators), 10

HeaderValidator (class in subui.validators), 10

## I

init() (subui.step.TestStep method), 6

initial\_data\_key (subui.validators.FormInitialDataValidator attribute), 10

## N

next\_step (subui.step.TestStep attribute), 7

next\_steps (subui.step.TestStep attribute), 7

## O

overridden\_settings (subui.step.TestStep attribute), 7

## P

post\_request\_hook() (subui.step.TestStep method), 7

post\_test\_response() (subui.step.TestStep method), 7

pre\_request\_hook() (subui.step.TestStep method), 7

pre\_test\_response() (subui.step.TestStep method), 7

prev\_step (subui.step.TestStep attribute), 7

prev\_steps (subui.step.TestStep attribute), 7

## R

RedirectToRouteValidator (class in subui.validators), 10

request() (subui.step.TestStep method), 7

request\_method (subui.step.TestStep attribute), 8

ResponseContentContainsValidator (class in subui.validators), 11

ResponseContentNotContainsValidator (class in subui.validators), 11

run() (subui.test\_runner.SubUITestRunner method), 8

## S

SessionDataValidator (class in subui.validators), 11

state (subui.step.TestStep attribute), 8

StatefulUriParamsTestStep (class in subui.step), 4

StatusCodeCreatedValidator (class in subui.validators), 11

StatusCodeOkValidator (class in subui.validators), 12

StatusCodeRedirectValidator (class in subui.validators), 12

StatusCodeValidator (class in subui.validators), 12

subui (module), 1

subui.step (module), 4

subui.test\_runner (module), 8

subui.validators (module), 9

SubUITestRunner (class in subui.test\_runner), 8

## T

test (subui.step.TestStep attribute), 8

test() (subui.validators.BaseValidator method), 9

test() (subui.validators.FormInitialDataValidator method), 10

test() (subui.validators.HeaderValidator method), 10

test() (subui.validators.RedirectToRouteValidator method), 11

test() (subui.validators.ResponseContentContainsValidator method), 11

test() (subui.validators.ResponseContentNotContainsValidator method), 11

test() (subui.validators.SessionDataValidator method), 11

test() (subui.validators.StatusCodeValidator method), 12

test\_contains\_value (subui.validators.HeaderValidator attribute), 10

test\_header\_value (subui.validators.HeaderValidator attribute), 10

test\_header\_value (subui.validators.StatusCodeRedirectValidator attribute), 12

test\_initial\_value (subui.validators.FormInitialDataValidator attribute), 10

test\_initial\_value\_not\_none (subui.validators.FormInitialDataValidator attribute), 10

test\_initial\_value\_present (subui.validators.FormInitialDataValidator attribute), 10

test\_response() (subui.step.TestStep method), 8

TestStep (class in subui.step), 5

## U

unexpected\_content (subui.validators.ResponseContentNotContainsValidator attribute), 11

url\_args (subui.step.TestStep attribute), 8

url\_kwargs (subui.step.TestStep attribute), 8

url\_name (subui.step.TestStep attribute), 8

urlconf (subui.step.TestStep attribute), 8

## V

validators (subui.step.TestStep attribute), 8