DjBurger Documentation

Release 0.8.1

@orsinium

Sep 07, 2018

Main Info

1 Installation 3

2 Philosophy 5

3 Usage 9

4 Examples 13

5 Interfaces 15

6 Short names 17

7 External libraries support 19

8 How to choose validation library 21

9 Changelog 23

10 Views 25

11 Parsers 29

12 Validators 31

13 Controllers 37

14 Renderers 41

Python Module Index 43

DjBurger – framework for big Django projects.

What DjBurger do?

- Split Django views into steps for secure and clean code.
- Provide built-in objects for all steps.
- Integrates this many side libraries like Django REST Framework and Marshmallow.

DjBurger doesn't depend on Django. You can use it in any projects if you want.

CHAPTER 1

Installation

1.1 STABLE

```
pip install djburger
```

1.2 DEV

Using pip:

```
sudo pip install -e git+https://github.com/orsinium/djburger.git#egg=djburger
```

In `requirements.txt`:

```
-e git+https://github.com/orsinium/djburger.git#egg=djburger
```

1.3 Dependencies

DjBurger only dependency is six. DjBurger doesn't depend on Django.

But DjBurger doesn't support `Django<1.7` and `DjangoRESTFramework<3.5`. You can pass constraints from DjBurger in pip:

```
pip install -c constraints.txt ...
```

CHAPTER 2

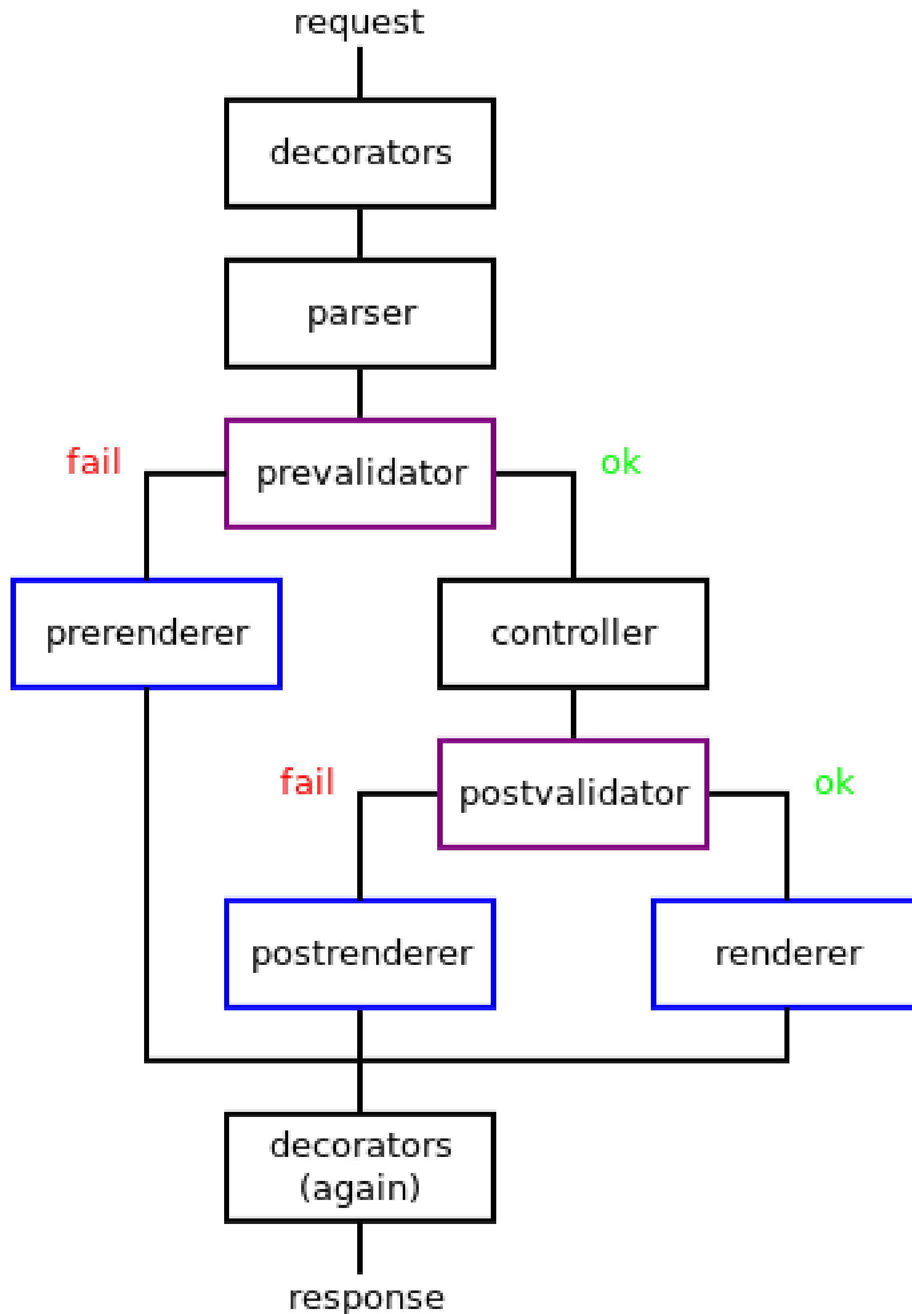
Philosophy

2.1 Key principles

1. Validation logic is separate from the main logic.
2. Reusable logic for many views.
3. Reusable input and output data formats.
4. More clean views.

2.2 Dataflow

1. **Decorators**. Feel free to use any side Django decorators like `csrf_exempt`.
2. **Parser**. Parse request body.
3. **PreValidator**. Validate and clear request.
4. **PreRenderer**. Render and return PreValidation errors response.
5. **Controller**. Main logic: do some things.
6. **PostValidator**. Validate and clear response.
7. **PostRenderer**. Render and return PostValidation errors response.
8. **Renderer**. Render successful response.



Required only controller and renderer.

2.3 Related conceptions

- Design by contract. DjBurger use pre-validation and post-validation as contracts for web interfaces and between controllers. For non-web projects you can use deal powered pure design by contract with DjBurger validators. More info into deal documentation.
- SOLID. DjBurger support this conceptions for projects:
  - SRP. This is main DjBurger idea. If you have many API for one business logic you must made only one

controller and many views with different parsers, validators and renderers.

  - OCP. All DjBurger components are extendable and do not use any global variables such as Django settings.
  - LSP. You can use any callable object as parser, validator, controller or renderer if it is match to interface.
  - ISP. DjBurger made simpler interface creation and main logic reusing.
  - DIP. Parsers, validators, controllers and renderers is are mutually independent.

- Cohesion & Coupling. DjBurger created for simplifying modules structure, high cohesion and low coupling. This is main idea how DjBurger help you make good, readable and extendable code for big projects.
- REST by support state codes, any external decorators and Django REST Framework. You can effective use DjBurger for building REST API. But DjBurger doesn't force this conception, and you can use it for any views as user interface rendering.
- MVC. Django use MTV conception, and views manipulates models and render templates. This is simple but dirty. DjBurger split view to parser, validator and renderer (this is all view from MVC) and controller for low coupling and clean code. And DjBurger render template only from renderer.
- The Clean Architecture. DjBurger hard isolate UI from Presenters. This is main clean architecture conception.

2.4 Solved problems

1. Mixins chaos and wrong usage. Mixins is good but dangerous conception. Use it carefully.
2. You can decorate Django view in so many places: urls, view, dispatch method, get/post method, form_valid method etc. DjBurger has special place for all decorators.
3. Validation and main logic mixing. In bad code you can validate and apply some data in one loop, and if you found and return validation error then operation will be applied partially. Example:

```
for user in users:
    if user.is_anonymous():
        raise ValueError("Can't send SMS to anonymous user!")
    send_sms(user)
```

This is bad code! We will send some SMS before error raising. DjBurger force input validation only before main logic.

1. Big and implicit views. This code is unsupported and non-expandable. DjBurger split view by steps.
2. Logic duplication in views require more time to made changes and and increase probability of mistake. With DjBurger you can use one controller into many views.

CHAPTER 3

Usage

3.1 Components structure

DjBurger modules:

- `djburger.parsers`. Parsers. Can be used as `parser`.
- `djburger.validators`. Can be used as `prevalidator` and `postvalidator`.
  - `djburger.validators.bases`. Validators that can be used as base class for your own validators.
  - `djburger.validators.constructors`. Validators that can be used as constructors for simple validators.
  - `djburger.validators.wrappers`. Validators that can be used as wrappers for external validators.
- `djburger.controllers`. Can be used as `controller`.
- `djburger.renderers`. Can be used as `prerenderer`, `postrenderer` and `renderer`.
- `djburger.exceptions`.

Keyword arguments for `djburger.rule`:

1. `decorators` – decorators list. Optional.
2. `parser` – parser. `djburger.parsers.Default` by default.
3. `prevalidator` – pre-validator. Optional.
4. `prerenderer` – renderer for pre-validator. If missed then `r` will be used for pre-validation errors rendering.
5. `controller` – controller. Required.
6. `postvalidator` – post-validator. Optional.
7. `postrenderer` – renderer for post-validator. If missed then `r` will be used for post-validation errors rendering.
8. `renderer` – renderer for success response. Required.

3.2 View

1. Extend `djburger.ViewBase`
2. Set `rules` with HTTP-request names as keys and `djburger.rule` as values.
3. Set up your `djburger.rule` by passing components as kwargs.

Example:

```
import djburger

class ExampleView(djburger.ViewBase):
    rules = {
        'get': djburger.rule(
            controller=lambda request, data, **kwargs: 'Hello, World!',
            postvalidator=djburger.validators.constructors.IsStr,
            postrenderer=djburger.renderers.Exception(),
            renderer=djburger.renderers.Template(template_name='index.html'),
        ),
    }
```

More info:

1. Dataflow
2. View usage examples
3. Example project
4. heck views API, but `rules` attribute are sufficient and redefinition of methods is not required.

3.3 Decorators

You can use any Django decorators like `csrf_exempt`. `djburger.ViewBase` wraps into decorators view's `validate_request` method that get `request` object, `**kwargs` from URL resolver and returns renderer's result (usually Django HttpResponse).

```
decorators=[csrf_exempt]
```

3.4 Parsers

Parser get `request` and return `data` which will be passed as is into pre-validator. Usually `data` has `dict` or QueryDict interface. DjBurger use `djburger.parsers.Default` as default parser. See list of built-in parsers into parsers API.

```
parser=djburger.parsers.JSON()
```

3.5 Validators

Validators get data and validate it. They have Django Forms-like interface. See interfaces and interface API for details.

Base validators - base class for your schemes:

```
from django import forms

class Validator(djburger.validators.bases.Form):
    name = forms.CharField(max_length=20)

...
prevalidator=Validator
...
```

Wrappers wrap external validators for DjBurger usage:

```
from django import forms

class DjangoValidator(forms.Form):
    name = forms.CharField(max_length=20)

...
prevalidator=djburger.validators.wrappers.Form(DjangoValidator)
...
```

And constructors for quick constructing simple validators. You can validate anything with constructors, but recommended case - one-line validators.

```
prevalidator=djburger.validators.constructors.IsDict
```

How to choose validator type:

1. `djburger.validators.constructors` – for one-line simple validation.
2. `djburger.validators.wrappers` – for using validators which also used into non-DjBurger components.
3. `djburger.validators.bases` – for any other cases.

3.6 Controllers

Controller – any callable object which get `request` object, `data` and `**kwargs` from URL resolver and return any `data`. Usually you will use your own controllers.

```
def echo_controller(request, data, **kwargs):
    return data

...
controller=echo_controller
...
```

Additionally DjBurger have built-in controllers for simple cases.

```
controller=djburger.controllers.Info(model=User)
```

3.7 Renderers

Renderer get errors or cleaned data from validator and return HttpResponse or any other view result.

```
postrenderer=djburger.renderers.JSON()
```

3.8 Exceptions

Raise `djburger.exceptions.StatusCodeError` from validator if you want stop validation and return `errors`.

```
from django import forms

class Validator(djburger.validators.bases.Form):
    name = forms.CharField(max_length=20)

    def clean_name(self):
        name = self.cleaned_data['name']
        if name == 'admin':
            self.errors = {'__all__': ['User not found']}
            raise djburger.exceptions.StatusCodeError(404)
        return name

...
prevalidator=Validator
...
```

3.9 SubControllers

If you need to validate data in controller, better use `djburger.controllers.subcontroller`:

```
def get_name_controller(request, data, **kwargs):
    return data['name']

def echo_controller(request, data, **kwargs):
    subc = djburger.controllers.subcontroller(
        prevalidator=djburger.controllers.IsDict,
        controller=get_name_controller,
        postvalidator=djburger.controllers.IsStr,
    )
    return subc(request, data, **kwargs)

...
controller=echo_controller
...
```

If data passed to subcontroller is invalid then `djburger.exceptions.SubValidationError` will be immediately raised. View catch error and pass error to `postrenderer`.

CHAPTER 4

Examples

4.1 Views

```
import djburger

class ExampleView(djburger.ViewBase):
    rules = {
        'get': djburger.rule(
            controller=lambda request, data, **kwargs: 'Hello, World!',
            postvalidator=djburger.validators.constructors.IsStr,
            postrenderer=djburger.renderers.Exception(),
            renderer=djburger.renderers.Template(template_name='index.html'),
        ),
    }
```

Minimum info:

```
class ExampleView(djburger.ViewBase):
    default_rule = djburger.rule(
        controller=lambda request, data, **kwargs: 'Hello, World!',
        renderer=djburger.renderers.Template(template_name='index.html'),
    ),
```

All requests without the method defined in the `rules` will use the rule from `default_rule`.

Example:

```
class UsersView(djburger.ViewBase):
    rules = {
        'get': djburger.rule(
            decorators=[login_required, csrf_exempt],
            prevalidator=SomeValidator,
            controller=djburger.controllers.List(model=User),
            postvalidator=djburger.validators.constructors.QuerySet,
```

(continues on next page)

(continued from previous page)

```
            postrenderer=djburger.renderers.Exception(),
            renderer=djburger.renderers.JSON(),
        ),
        'put': djburger.rule(
            decorators=[csrf_exempt],
            parser=djburger.parsers.JSON(),
            prevalidator=SomeOtherValidator,
            controller=djburger.controllers.Add(model=User),
            renderer=djburger.renderers.JSON(),
        ),
    }
```

4.2 Validators

Simple base validator:

```
class GroupInputValidator(djburger.validators.bases.Form):
    name = djburger.forms.CharField(label='Name', max_length=80)
```

`djburger.forms` is useful alias for `django.forms`.

Simple wrapper:

```
import djburger
from django import forms

class GroupInputForm(forms.Form):
    name = forms.CharField(label='Name', max_length=80)

Validator = djburger.validators.wrappers.Form(GroupInputForm)
```

See usage for more examples and explore example project.

CHAPTER 5

Interfaces

1. **Decorator**. Any decorator which can wrap Django view.
2. **Parser**. Any callable object which get request object and return parsed data.
3. **Validator**. Have same interfaces as Django Forms, but get `request` by initialization:
   (a) `.__init__()`
      - `request` – Request object.
      - `data` – data from user (`prevalidator`) or controller (`postvalidator`).
      - `**kwargs` – any keyword arguments for validator.

   (b) `.is_valid()` – return True if data is valid False otherwise.
   (c) `.errors` – errors if data is invalid.
   (d) `.cleaned_data` – cleaned data if input data is valid.
4. **Controller**. Any callable object. Kwargs:
   (a) `request` – Request object.
   (b) `data` – validated request data. 3 `**kwargs` – kwargs from url.
5. **Renderer**. Any callable object. Kwargs:
   (a) `request` – Request object.
   (b) `data` – validated controller data (only for `r`).
   (c) `validator` – validator which not be passed (only for `prerenderer` and `postrenderer`).
   (d) `status_code` – HTTP status code if validator raise `djburger.exceptions.StatusCodeError`, None otherwise.

CHAPTER 6

Short names

DjBurger provide some short aliases. Please, do not use it into big and Open Source packages. Use it only for personal projects.

6.1 Rule

You can use short kwargs for rule function:

Full name	Short name
decorators	d
parser	p
prevalidator	prev
prerenderer	prer
controller	c
postvalidator	postv
postrenderer	postr
renderer	r

6.2 Modules

Also DjBurger modules has short names:

Full name	Short name
djburger.controllers	djburger.c
djburger.exceptions	djburger.e
djburger.forms	djburger.f
djburger.parsers	djburger.p
djburger.renderers	djburger.r
djburger.validators.bases	djburger.v.b
djburger.validators.constructors	djburger.v.c
djburger.validators.wrappers	djburger.v.w

CHAPTER 7

External libraries support

- BSON
  - `djburger.parsers.BSON`
  - `djburger.renderers.BSON`
- Django REST Framework
  - `djburger.validators.bases.RESTFramework`
  - `djburger.validators.wrappers.RESTFramework`
  - `djburger.renderers.RESTFramework`
- Marshmallow
  - `djburger.validators.bases.Marshmallow`
  - `djburger.validators.wrappers.Marshmallow`
- PySchemes
  - `djburger.validators.constructors.PySchemes`
  - `djburger.validators.wrappers.PySchemes`
- PyYAML
  - `djburger.renderers.YAML`
- Tablib
  - `djburger.renderers.Tablib`

CHAPTER 8

How to choose validation library

8.1 Validators by features

1. Type casting
   (a) Cerberus
   (b) DjBurger
   (c) WTForms
2. Optional fields
3. Required fields
   (a) Cerberus
   (b) DjangoRESTFramework
   (c) DjBurger
   (d) Marshmallow
   (e) PySchemes
   (f) WTForms
4. Drop unknown fields
   (a) Cerberus
   (b) DjBurger
   (c) WTForms
5. Django querysets support
6. Django models support
   (a) DjangoRESTFramework
   (b) Marshmallow

8.2 Validators by recommended validation type

1. Pre-validation
   (a) Cerberus
   (b) DjBurger
   (c) WTForms
2. Post-validation
   (a) DjangoRESTFramework
   (b) Marshmallow
   (c) PySchemes (doesn't support Django models)

CHAPTER 9

Changelog

See projects tab on Github for roadmap.

9.1 v. 0.11.0

- New default parser
- WTForms support
- Validators comparing and testing.
- Some new internal data structures for Django independence.

9.2 v. 0.10.1

- Long kwargs for rule function and subcontroller.
- Replaced aliases to long names into documentation, tests and examples.
- Added documentation tests
- Added info about aliases into documentation.

9.3 v. 0.9.0

- Good documentation.
- Short readme.
- First anounce on Reddit.

9.4 v. 0.8.0

- Tested and improved.
- Many custom renderers.

CHAPTER 10

Views

`djburger.views.rule(**kwargs)`

Factory for _Rule objects

- Any kwarg can contain str which point where function can get value for kwarg.
- Some kwargs can contain None.

Example::

```
>>> rule(
...     decorators=[login_required, csrf_exempt],
...     prevalidator=SomeDjangoForm,
...     prerenderer='postrenderer',
...     # ^ here `prerenderer` point to `postrenderer`
...     controller=some_controller,
...     postvalidator=None,
...     # ^ here post-validator is missed
...     postrenderer='renderer',
...     renderer=djburger.renderers.JSON(),
... )
```

Parameters

- **decorators** (*list*) – list of decorators.
- **parser** (*callable*) – parse request body. *djburger.parsers.Default* by default.
- **prevalidator** (djburger.validators.bases.IValidator) – validate and clean user params.
- **prerenderer** (*callable*) – renderer for pre-validation errors.
- **controller** (*callable*) –
- **postvalidator** (djburger.validators.bases.IValidator) – validate and clean response.

- **postrenderer** (*callable*) – renderer for post-validation errors.
- **renderer** (*callable*) – renderer for successfull response.

Returns rule.

Return type djburger._Rule

Raises **TypeError** – if missed *c* or *r*.

class djburger.views.**ViewBase**
Base views for DjBurger usage.

Parameters

- **request** (*django.http.request.HttpRequest*) – user request object.
- ****kwargs** – kwargs from urls.py.

Returns django response.

Return type django.http.HttpResponse

dispatch(*request*, ***kwargs*)
Entrypoint for view

1. Select rule from rules.
2. Decorate view
3. Call *validate* method.

Parameters

- **request** (*django.http.request.HttpRequest*) – user request object.
- ****kwargs** – kwargs from urls.py.

Returns django response.

Return type django.http.HttpResponse

get_data(*request*)
Extract data from request by parser.

Parameters **request** (*django.http.request.HttpRequest*) – user request object.

Returns parsed data.

get_validator_kwargs(*data*)
Get kwargs for validators

Parameters **data** – data which will be validated.

Returns kwargs for (post)validator.

Return type dict

make_response(*data*)
Make response by renderer

Parameters **data** – cleaned and validated data from controller.

Returns django response.

Return type django.http.HttpResponse

request_invalid(*validator*, *status_code*)
Return result of prer (renderer for pre-validator errors)

Parameters

- **validator** (djburger.validators.bases.IValidator) – validator object with *errors* attr.
- **status_code** (*int*) – status code for HTTP-response.

Returns django response.

Return type django.http.HttpResponse

request_valid(*data*, ***kwargs*)
Call controller.

Get response from controller and return result of validate_response method.

Parameters

- **data** – cleaned and validated data from user.
- ****kwargs** – kwargs from urls.py.

Returns django response.

Return type django.http.HttpResponse

response_invalid(*validator*, *status_code*)
Return result of postr (renderer for post-validation errors).

Parameters

- **validator** (djburger.validators.bases.IValidator) – validator object with *errors* attr.
- **status_code** (*int*) – status code for HTTP-response.

Returns django response.

Return type django.http.HttpResponse

response_valid(*validator*)
Return result of make_response. This method calls only if postv is not None.

Parameters **validator** (djburger.validators.bases.IValidator) – validator object with *cleaned_data* attr.

Returns django response.

Return type django.http.HttpResponse

subvalidation_invalid(*validator*, *status_code=200*)
Return result of postr (renderer for post-validation errors).

Parameters

- **validator** (djburger.validators.bases.IValidator) – validator object with *errors* attr.
- **status_code** (*int*) – status code for HTTP-response.

Returns django response.

Return type django.http.HttpResponse

validate_request(*request*, ***kwargs*)

1. Call *request_valid* method if validation is successfull or missed.
2. Call *request_invalid* method otherwise.

Parameters

- **request** (*django.http.request.HttpRequest*) – user request object.
- ****kwargs** – kwargs from urls.py.

Returns django response.

Return type django.http.HttpResponse

validate_response(*response*)

Validate response by postv (post-validator)

1. Return make_response method result if post-validator is missed.
2. **Validate data by post-validator otherwise and call…**
   - response_valid if validation is passed
   - or response_invalid otherwise.

Parameters **response** – unvalidated data from controller.

Returns django response.

Return type django.http.HttpResponse

CHAPTER 11

Parsers

class djburger.parsers.**MultiDict**(*method=None*)
Parse standart GET/POST query to MultiDict

Parameters **method** (*str*) – optional method which will be forced for request

Returns parsed data.

Return type django.http.request.QueryDict

class djburger.parsers.**DictList**(*method=None*)
Parse standart GET/POST query to dict of lists

Parameters **method** (*str*) – optional method which will be forced for request

Returns parsed data.

Return type *Dict*[list]

class djburger.parsers.**DictMixed**(*method=None*)
Parse standart GET/POST query to dict of lists or values

Parameters **method** (*str*) – optional method which will be forced for request

Returns parsed data.

Return type dict

class djburger.parsers.**Dict**(*method=None*)
Parse standart GET/POST query to dict

Parameters **method** (*str*) – optional method which will be forced for request

Returns parsed data.

Return type dict

class djburger.parsers.**Base**(*parser*, *encoding='utf-8'*, ***kwargs*)
Allow use any callable object as parser

Parameters

- **parser** (*callable*) – callable object for parsing request body.
- **encoding** (*str*) – if not None body will be decoded from byte to str.
- ****kwargs** – kwargs for parser.

Returns parsed data.

djburger.parsers.**JSON = <functools.partial object>**
Parse JSON body.

Parameters

- **encoding** (*str*) – body encoding. UTF-8 by default.
- ****kwargs** – kwargs for *json.loads*.

Returns parsed data.

djburger.parsers.**BSON = <functools.partial object>**
Parse BSON body.

Parameters ****kwargs** – kwargs for *bson.loads*.

Returns parsed data.

Return type dict

Raises **ImportError** – if *bson* module not installed yet.

djburger.parsers.**Default**
alias of *djburger.parsers.MultiDict*

CHAPTER 12

Validators

12.1 Bases

Base classes for validators

Use this classes as base class for your own validators.

class djburger.validators.bases.**IValidator**

Abstract base class for validators.

cleaned_data

Cleaned data dict (or other type). Set by *is_valid* method.

errors

Errors dict. Set by *is_valid* method.

key: name of invalid field or *__all__*. value: list of errors strings.

is_valid()

Validate and clean data.

1. Set *cleaned_data* and return True if data is valid
2. Set *errors* and return False otherwise.

Returns: True: data is valid False: data is invalid

class djburger.validators.bases.**Form**(*data*, *request=None*, ***kwargs*)

Validator based on Django Forms.

class djburger.validators.bases.**ModelForm**(*data*, *request=None*, ***kwargs*)

Validator based on Django Model Forms.

save(**args*, ***kwargs*)

All operations into validators must be idempotency.

class djburger.validators.bases.**Marshmallow**(*data*, *request=None*, ***kwargs*)

Validator based on marshmallow schema.

class djburger.validators.bases.**WTForms**(*data*, *request=None*, ***kwargs*)
Validator based on WTForms form.

class djburger.validators.bases.**RESTFramework**(*data*, *request=None*, ***kwargs*)
Validator based on Django REST Framework serializers.

12.2 Wrappers

Wrappers for validators

Use this classes as wrappers for non-djburger validators

class djburger.validators.wrappers.**Form**(*validator*)
Wrapper for use Django Form (or ModelForm) as validator.

djburger.validators.wrappers.**ModelForm**
alias of *djburger.validators.wrappers.Form*

class djburger.validators.wrappers.**Marshmallow**(*validator*)
Wrapper for use marshmallow scheme as validator.

class djburger.validators.wrappers.**PySchemes**(*validator*)
Wrapper for use PySchemes as validator.

class djburger.validators.wrappers.**Cerberus**(*validator*)
Wrapper for use Cerberus as validator.

class djburger.validators.wrappers.**RESTFramework**(*validator*)
Wrapper for use Django REST Framework serializer as validator.

class djburger.validators.wrappers.**WTForms**(*validator*)
Wrapper for use WTForms form as validator.

12.3 Constructors

Constructors for validators

Use this classes for constructing your own validators.

djburger.validators.constructors.**Any**
alias of *djburger.validators.constructors.Or*

djburger.validators.constructors.**All**
alias of *djburger.validators.constructors.Chain*

class djburger.validators.constructors.**Cerberus**(***kwargs*)
Validate data by Cerberus.

Parameters

- **schema** (*dict*) – validation scheme for Cerberus.
- **allow_unknown** (*bool*) –

class djburger.validators.constructors.**Chain**(**validators*)
Validate data by validators chain (like *reduce* function).

Calls the validators in order, passing in each subsequent cleaned data from the previous one.

Parameters **validators** (*list*) – list of validators.

is_valid()

Validate and clean data.

1. Set *cleaned_data* and return True if data is valid
2. Set *errors* and return False otherwise.

Returns: True: data is valid False: data is invalid

djburger.validators.constructors.**Dict**(*validator*)

Validate data dict

Parameters **validator** – validator which be applyed to all values of dict.

djburger.validators.constructors.**DictForm**(*form*)

Validate dict values by Django Forms

djburger.validators.constructors.**DictMixed**(*validators*, *policy='error'*, *required=False*)

Validate dict keys by multiple validators

Parameters

- **validators** (*dict*) – validator which be applyed to all values of dict.
- **policy** (*str*) – policy if validator for data not found: "error" - add error into *errors* attr and return False. "except" - raise KeyError exception. "ignore" - add source value into cleaned_data. "drop" - drop this value and continue.

djburger.validators.constructors.**DictModelForm**(*form*)

Validate dict values by Django Model Forms

djburger.validators.constructors.**IsBool = <djburger.validators.constructors.Type object>**

Data type is bool

djburger.validators.constructors.**IsDict = <djburger.validators.constructors.Type object>**

Data type is dict

djburger.validators.constructors.**IsFloat = <djburger.validators.constructors.Type object>**

Data type is float

djburger.validators.constructors.**IsInt = <djburger.validators.constructors.Type object>**

Data type is int

djburger.validators.constructors.**IsIter = <djburger.validators.constructors.Type object>**

Data type is iterable

djburger.validators.constructors.**IsList = <djburger.validators.constructors.Type object>**

Data type is list

djburger.validators.constructors.**IsStr = <djburger.validators.constructors.Type object>**

Data type is str

class djburger.validators.constructors.**Lambda**(*key*, *error_msg='Custom validation is failed'*)

Validate data by lambda expression.

Parameters **key** (*callable*) – lambda, function or other callable object which get data and return bool result (True if valid).

is_valid()

Validate and clean data.

1. Set *cleaned_data* and return True if data is valid
2. Set *errors* and return False otherwise.

Returns: True: data is valid False: data is invalid

`djburger.validators.constructors.`**`List`**(*validator*)

Validate data list.

Parameters **`validators`** – if passed only one validator it's be applied to each list element. One validator will be applyed to one element sequentionaly otherwise.

`djburger.validators.constructors.`**`ListForm`**(*form*)

Validate list elements by Django Forms

`djburger.validators.constructors.`**`ListModelForm`**(*form*)

Validate list elements by Django Model Forms

`djburger.validators.constructors.`**`ModelInstance = <djburger.validators.constructors.Chain ol`**

Validate model instance and convert it to dict.

Doesn't require initialization.

`class` `djburger.validators.constructors.`**`Or`**(**validators*)

Validate data by validators (like *any* function).

Calls the validators in order, return cleaned_data from first successfull validation or errors from last validator

Parameters **`validators`** (*`list`*) – list of validators.

`is_valid`()

Validate and clean data.

1. Set *cleaned_data* and return True if data is valid
2. Set *errors* and return False otherwise.

Returns: True: data is valid False: data is invalid

`djburger.validators.constructors.`**`OR`**

alias of *`djburger.validators.constructors.Or`*

`class` `djburger.validators.constructors.`**`PySchemes`**(***kwargs*)

Validate data by PySchemes.

Parameters **`scheme`** – validation scheme for pyschemes.

`class` `djburger.validators.constructors.`**`Type`**(*data_type*, *error_msg='Invalid data type: {}. Required {}.'*)

Validate data type

Parameters

- **`data_type`** (*`type`*) – required type of data.
- **`error_msg`** (*`str`*) – template for error message.

`is_valid`()

Validate and clean data.

1. Set *cleaned_data* and return True if data is valid
2. Set *errors* and return False otherwise.

Returns: True: data is valid False: data is invalid

djburger.validators.constructors.**QuerySet = <djburger.validators.constructors.Chain object>**

Validate queryset and convert each object in it to dict.

Doesn't require initialization.

CHAPTER 13

Controllers

class djburger.controllers.**List**(*only_data=True*, ***kwargs*)
Controller based on Django ListView

1. Get list of objects
2. Filter list by validated data from user
3. Optional pagination

Parameters

- **only_data** (*bool*) – return only filtered queryset if True, all context data otherwise. Use *only_data=False* with TemplateSerializerFactory.
- ****kwargs** – all arguments of ListView.

Returns filtered queryset.

Return type django.db.models.query.QuerySet

class djburger.controllers.**Info**(*queryset=None*, *model=None*)
Return one object from queryset

1. **Return object filtered by params from URL kwargs (like *pk* or *slug*)** if url-pattern have kwargs.
2. **Return object from validated data if data have key *object*** or data have only one key.
3. Raise exception otherwise.

Parameters

- **queryset** (*django.db.models.query.QuerySet*) – QuerySet for retrieving object.
- **model** (*django.db.models.Model*) – Model for retrieving object.

Returns one object from queryset or model.

Return type django.db.models.Model

Raises

- **ValueError** – if can't select params for queryset filtering.
- **django.http.Http404** – if object does not exist or multiple objects returned

class djburger.controllers.**Add**(*model*)

Controller for adding object with validated data.

Parameters **model** (*django.db.models.Model*) – Model for adding object.

Returns created object.

Return type django.db.models.Model

class djburger.controllers.**Edit**(*queryset=None*, *model=None*)

Controller for editing objects.

1. Get object of initialized model by URL's kwargs.
2. Set params from validated data.
3. Update tuple into database.

Parameters

- **queryset** (*django.db.models.query.QuerySet*) – QuerySet for editing object.
- **model** (*django.db.models.Model*) – Model for editing object.

Returns edited object.

Return type django.db.models.Model

Raises **django.http.Http404** – if object does not exist or multiple objects returned

class djburger.controllers.**Delete**(*queryset=None*, *model=None*)

Controller for deleting objects.

Delete object filtered by URL's kwargs.

Parameters

- **queryset** (*django.db.models.query.QuerySet*) – QuerySet for deleting object.
- **model** (*django.db.models.Model*) – Model for deleting object.

Returns count of deleted objects.

Return type int

Raises **django.http.Http404** – if object does not exist or multiple objects returned

class djburger.controllers.**ViewAsController**(*view*, *method=None*)

Allow use any django view as controller.

1. For CBV with render_to_response method return context.
2. Return rendered response otherwise.

Parameters

- **view** (*callable*) – view.
- **method** (*str*) – optional method which will be forced for request

Returns if possible response context or content, response object otherwise.

static patch_view(*view*)
Patch view for getting context instead of rendered response.

class djburger.controllers.**pre**(*validator*, ***kwargs*)
Decorator for input data validation before subcontroller calling

Parameters

- **validator** (djburger.validators.bases.IValidator) – validator for pre-validation.
- ****kwargs** – kwargs for validator.

Raises **djburger.exceptions.SubValidationError** – if pre-validation not passed

class djburger.controllers.**post**(*validator*, ***kwargs*)
Decorator for output data validation before subcontroller calling

Parameters

- **validator** (djburger.validators.bases.IValidator) – validator for post-validation.
- ****kwargs** – kwargs for validator.

Raises **djburger.exceptions.SubValidationError** – if post-validation not passed

djburger.controllers.**subcontroller**(*controller*, *prevalidator=None*, *postvalidator=None*)
Constructor for subcontrollers If any validation failed, immediately raise SubValidationError.

Parameters

- **prevalidator** (djburger.validators.bases.IValidator) –
- **controller** (*callable*) –
- **postvalidator** (djburger.validators.bases.IValidator) –

Raises **djburger.exceptions.SubValidationError** – if any validation not passed

CHAPTER 14

Renderers

class djburger.renderers.**BSON**(*flat=True*, ***kwargs*)
Render into BSON format by BSON package

Returns rendered response.

Return type django.http.HttpResponse

class djburger.renderers.**Base**(*renderer*, *content_name*, *request_name=None*, *names=None*, *content=None*, *flat=False*, ***kwargs*)
Wrapper for using any function as renderer.

Parameters

- **renderer** (*callable*) – function for rendering.
- **content_name** (*str*) – keyword of data argument for renderer.
- **request_name** (*str*) – keyword of Request argument for renderer.
- **names** (*dict*) – dict of names. Keys: * data (default: "data"): name for data into content. * errors (default: "errors"): name for errors into content. * validator (default: None): name for validator into content. If not setted, validator will not be passed into content.
- **content** (*dict*) – default params for content.
- **flat** (*bool*) – if True content contains only data or errors.
- ****kwargs** – some kwargs which will be passed to renderer.

Returns rendered response.

Return type django.http.HttpResponse

class djburger.renderers.**BaseWithHTTP**(*renderer*, *content_name*, *request_name=None*, *names=None*, *content=None*, *flat=False*, ***kwargs*)
Base class wrapped by HttpResponse

Parameters ****kwargs** – all kwargs of djburger.renderers.Base.

set_http_kwargs(***kwargs*)
Set kwargs for HttpResponse

Parameters ****kwargs** – all kwargs of HttpResponse.

Returns self. Allow use chain.

Return type *BaseWithHTTP*

class djburger.renderers.**Exception**(*exception=<class 'djburger.mocks.ValidationError'>*)
Raise Exception

We are recommend use this renderer as *postrenderer*. Raised exception can be handled by decorators or loggers.

Parameters **exception** – exception for raising.

class djburger.renderers.**HTTP**(*status_code=200*, ***kwargs*)
Render data by HttpResponse.

data can be only *str* or *bytes* type.

class djburger.renderers.**JSON**(*flat=True*, *safe=False*, ***kwargs*)
Serialize data into JSON

class djburger.renderers.**RESTFramework**(*renderer*, *flat=True*, ***kwargs*)
Wrapper for renderers from Django REST Framework

class djburger.renderers.**Redirect**(*url=None*)
Redirect to URL

URL can be passed by initialization or as data (str).

Parameters **url** – url for redirect.

Returns rendered redirect response.

Return type django.http.HttpResponseRedirect

class djburger.renderers.**Tablib**(*ext*, *headers=None*, ***kwargs*)
Render into multiple formats by tablib

Parameters

- **ext** – extension for rendering by Tablib.
- **headers** – table headers.

Returns rendered response.

Return type django.http.HttpResponse

djburger.renderers.**Template = <functools.partial object>**
Serializer based on Django Templates

class djburger.renderers.**YAML**(*flat=True*, ***kwargs*)
Render into YAML format by PyYAML

Returns rendered response.

Return type django.http.HttpResponse

Python Module Index

d

djburger.controllers, 37
djburger.parsers, 29
djburger.renderers, 41
djburger.validators.bases, 31
djburger.validators.constructors, 32
djburger.validators.wrappers, 32
djburger.views, 25

Index

A

Add (class in djburger.controllers), 38
All (in module djburger.validators.constructors), 32
Any (in module djburger.validators.constructors), 32

B

Base (class in djburger.parsers), 29
Base (class in djburger.renderers), 41
BaseWithHTTP (class in djburger.renderers), 41
BSON (class in djburger.renderers), 41
BSON (in module djburger.parsers), 30

C

Cerberus (class in djburger.validators.constructors), 32
Cerberus (class in djburger.validators.wrappers), 32
Chain (class in djburger.validators.constructors), 32
cleaned_data (djburger.validators.bases.IValidator attribute), 31

D

Default (in module djburger.parsers), 30
Delete (class in djburger.controllers), 38
Dict (class in djburger.parsers), 29
Dict() (in module djburger.validators.constructors), 33
DictForm() (in module djburger.validators.constructors), 33
DictList (class in djburger.parsers), 29
DictMixed (class in djburger.parsers), 29
DictMixed() (in module djburger.validators.constructors), 33
DictModelForm() (in module djburger.validators.constructors), 33
dispatch() (djburger.views.ViewBase method), 26
djburger.controllers (module), 37
djburger.parsers (module), 29
djburger.renderers (module), 41
djburger.validators.bases (module), 31
djburger.validators.constructors (module), 32
djburger.validators.wrappers (module), 32
djburger.views (module), 25

E

Edit (class in djburger.controllers), 38
errors (djburger.validators.bases.IValidator attribute), 31
Exception (class in djburger.renderers), 42

F

Form (class in djburger.validators.bases), 31
Form (class in djburger.validators.wrappers), 32

G

get_data() (djburger.views.ViewBase method), 26
get_validator_kwargs() (djburger.views.ViewBase method), 26

H

HTTP (class in djburger.renderers), 42

I

Info (class in djburger.controllers), 37
is_valid() (djburger.validators.bases.IValidator method), 31
is_valid() (djburger.validators.constructors.Chain method), 32
is_valid() (djburger.validators.constructors.Lambda method), 33
is_valid() (djburger.validators.constructors.Or method), 34
is_valid() (djburger.validators.constructors.Type method), 34
IsBool (in module djburger.validators.constructors), 33
IsDict (in module djburger.validators.constructors), 33
IsFloat (in module djburger.validators.constructors), 33
IsInt (in module djburger.validators.constructors), 33
IsIter (in module djburger.validators.constructors), 33
IsList (in module djburger.validators.constructors), 33
IsStr (in module djburger.validators.constructors), 33
IValidator (class in djburger.validators.bases), 31

J

JSON (class in djburger.renderers), 42
JSON (in module djburger.parsers), 30

L

Lambda (class in djburger.validators.constructors), 33
List (class in djburger.controllers), 37
List() (in module djburger.validators.constructors), 34
ListForm() (in module djburger.validators.constructors), 34
ListModelForm() (in module djburger.validators.constructors), 34

M

make_response() (djburger.views.ViewBase method), 26
Marshmallow (class in djburger.validators.bases), 31
Marshmallow (class in djburger.validators.wrappers), 32
ModelForm (class in djburger.validators.bases), 31
ModelForm (in module djburger.validators.wrappers), 32
ModelInstance (in module djburger.validators.constructors), 34
MultiDict (class in djburger.parsers), 29

O

Or (class in djburger.validators.constructors), 34
OR (in module djburger.validators.constructors), 34

P

patch_view() (djburger.controllers.ViewAsController static method), 39
post (class in djburger.controllers), 39
pre (class in djburger.controllers), 39
PySchemes (class in djburger.validators.constructors), 34
PySchemes (class in djburger.validators.wrappers), 32

Q

QuerySet (in module djburger.validators.constructors), 34

R

Redirect (class in djburger.renderers), 42
request_invalid() (djburger.views.ViewBase method), 26
request_valid() (djburger.views.ViewBase method), 27
response_invalid() (djburger.views.ViewBase method), 27
response_valid() (djburger.views.ViewBase method), 27
RESTFramework (class in djburger.renderers), 42
RESTFramework (class in djburger.validators.bases), 32
RESTFramework (class in djburger.validators.wrappers), 32
rule() (in module djburger.views), 25

S

save() (djburger.validators.bases.ModelForm method), 31
set_http_kwargs() (djburger.renderers.BaseWithHTTP method), 41
subcontroller() (in module djburger.controllers), 39
subvalidation_invalid() (djburger.views.ViewBase method), 27

T

Tablib (class in djburger.renderers), 42
Template (in module djburger.renderers), 42
Type (class in djburger.validators.constructors), 34

V

validate_request() (djburger.views.ViewBase method), 27
validate_response() (djburger.views.ViewBase method), 28
ViewAsController (class in djburger.controllers), 38
ViewBase (class in djburger.views), 26

W

WTForms (class in djburger.validators.bases), 31
WTForms (class in djburger.validators.wrappers), 32

Y

YAML (class in djburger.renderers), 42