
sanic-validation Documentation

Release 0.4.4

Piotr Bakalarski

Apr 27, 2019

Contents

| | | |
|----------|---------------------------|----------|
| 1 | Contents | 3 |
| 1.1 | Getting started | 3 |
| 1.2 | Usage | 4 |
| 1.3 | API | 6 |

sanic-validation is an extension to sanic that simplifies validating request data. For an overview of the library see *Getting started*. More detailed information can be found in the *Usage* section. For a full reference navigate to *API*.

1.1 Getting started

1.1.1 Installation

Installation from PyPi:

```
pip install sanic-validation
```

1.1.2 Simple example

Code of the *hello service*:

```
from sanic import Sanic
from sanic.response import json
from sanic_validation import validate_json

app = Sanic()

schema = {'name': {'type': 'string', 'required': True}}

@app.route('/')
@validate_json(schema)
async def hello(request):
    return json({'message': 'Hello ' + request.json['name']})

app.run('0.0.0.0')
```

An example of a bad request:

```
GET / HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: localhost:8000
User-Agent: HTTPie/0.9.9
```

And the response:

```
HTTP/1.1 400 Bad Request
Connection: keep-alive
Content-Length: 168
Content-Type: application/json
Keep-Alive: 5

{
  "error": {
    "invalid": [
      {
        "constraint": true,
        "entry": "name",
        "entry_type": "json_data_property",
        "rule": "required"
      }
    ],
    "message": "Validation failed.",
    "type": "validation_failed"
  }
}
```

1.2 Usage

1.2.1 Validating JSON

To validate body JSON, use the `validate_json()` decorator:

```
@app.route('/')
@validate_json(schema)
async def hello(request):
    return text("OK")
```

If all fields in schema are optional, then an empty JSON object `{ }` will be accepted, but an empty request body will be rejected.

If you set the `clean` argument to `True`, validated and normalized data will be passed to the handler method as `valid_json`:

```
app.route('/')
@validate_json(schema, clean=True)
async def my_age(request, valid_json):
    return text(valid_json['age'])
```


1.2.2 Validating querystring arguments

To validate querystring arguments, use the `validate_args()` decorator:

```
@app.route('/')
@validate_args(schema)
async def hello(request):
    return text("OK")
```

Note: All querystring argument values are strings. To use validation rules for other types use coercion rules (see *Normalization*).

If you set the `clean` argument to `True`, validated and normalized data will be passed to the handler method as `valid_args`:

```
app.route('/')
@validate_args(schema, clean=True)
async def my_age(request, valid_args):
    return text(valid_args['age'])
```

1.2.3 Error response format

In case of an error the request returns with status of 400. Example error response:

```
{
  "error": {
    "invalid": [
      {
        "constraint": true,
        "entry": "name",
        "entry_type": "json_data_property",
        "rule": "required"
      }
    ],
    "message": "Validation failed.",
    "type": "validation_failed"
  }
}
```

Fields definitions:

type: machine readable description of the problem

message: user readable description of the problem

invalid: list containing all validation errors

entry_type: type of the incorrect entry (json data, querystring parameter, etc)

entry: path to the incorrect entry

rule: rule that failed validation

constraint: expected value for the rule

1.2.4 Schema

sanic-validation uses Cerberus as the validation library. For the list of available rules see [Cerberus' schema documentation](#).

1.2.5 Normalization

Normalization during validation works by default. To access normalized data in handler methods set the *clean* flag on the decorator, and create the correct argument in the handler method. See [Validating JSON](#) and [Validating querystring arguments](#) for more details.

See [Cerberus' normalization documentation](#) for the list of normalization rules.

1.2.6 Extending

Custom rules, data types and coercers can be easily added. Consult [Cerberus' customization documentation](#) for details.

1.3 API

1.3.1 JSON validation

`sanic_validation.validate_json` (*schema*, *clean=False*, *status_code=400*)
Decorator. Validates request body json.

When *clean* is true, normalized data is passed to the decorated method as *valid_json*.

Parameters

- **schema** (*dict*) – Cerberus-compatible schema description
- **clean** (*bool*) – should cleaned json be passed to the decorated method
- **status_code** (*number*) – status code to return when data is incorrect

1.3.2 Querystring validation

`sanic_validation.validate_args` (*schema*, *clean=False*, *status_code=400*)
Decorator. Validates querystring arguments.

When *clean* is True, normalized data is passed to the decorated method as *valid_args*.

Parameters

- **schema** (*dict*) – Cerberus-compatible schema description
- **clean** (*bool*) – should cleaned args be passed to the decorated method
- **status_code** (*number*) – status code to return when data is incorrect

V

`validate_args()` (*in module `sanic_validation`*), 6
`validate_json()` (*in module `sanic_validation`*), 6