filteralchemy

Release 0.1.0

October 25, 2015

Contents

1 Guide **3**
1.1 Installation 3
1.2 Quickstart 3
1.3 API Reference 5

2 Project info **7**
2.1 Authors 7
2.2 License 7

Python Module Index **9**

Release v0.1.0. (*Changelog*)

filteralchemy is a declarative query builder for SQLAlchemy. **filteralchemy** uses **marshmallow-sqlalchemy_** to auto-generate filter fields and **webargs_** to parse field parameters from the request.

CHAPTER 1

Guide

1.1 Installation

1.1.1 From the PyPI

```
$ pip install -U filteralchemy
```

1.1.2 From Source

filteralchemy is actively developed on Github.

You can clone the public repo:

```
git clone https://github.com/jmcarp/filteralchemy.git
```

Once you have the source, you can install it into your site-packages with

```
$ python setup.py install
```

1.2 Quickstart

Use **filteralchemy** to auto-generate filters based on a SQLAlchemy model:

```
import flask
from models import Album, session
from webargs.flaskparser import parser
from filteralchemy import FilterSet

class AlbumFilterSet(FilterSet):
    class Meta:
        model = Album
        query = session.query(Album)
        parser = parser

app = flask.Flask(__name__)

@app.route('/albums')
def get_albums():
```

```
    query = AlbumFilterSet().filter()
    return flask.jsonify(query.all())
```

1.2.1 Customizing operators

By default, **filteralchemy** generates filters using the `Equal` operator. To generate filters with different operators, set the `operators` and `column_overrides` options in `Meta`:

```
from filteralchemy.operators import Greater, Less, Like

class AlbumFilterSet(FilterSet):
    class Meta:
        model = Album
        query = session.query(Album)
        operators = (Equal, Greater, Less)
        column_overrides = {
            'name': {'operators': (Equal, Like)}
        }
```

1.2.2 Restricting filtered columns

By default, all columns in the specified model are used to generate filters. To restrict filtered columns, set the `fields` or `exclude` options in Meta:

```
class AlbumFilterSet(FilterSet):
    class Meta:
        model = Album
        query = session.query(Album)
        fields = ('name', 'genre')
```

Both `fields` and `exclude` can either be a sequence of column names or a callable that receives the `FilterSet` subclass and returns a sequence of names. **filteralchemy** provides a helper, `index_columns`, that restricts fields to indexed columns:

```
from filteralchemy.utils import index_columns

class AlbumFilterSet(FilterSet):
    class Meta:
        model = Album
        query = session.query(Album)
        fields = index_columns(engine)
```

1.2.3 Declaring fields manually

Individual filters can also be declared manually as class variables:

```
from webargs import fields
from filteralchemy import Filter
from filteralchemy.operators import In, ILike

class AlbumFilterSet(FilterSet):
    class Meta:
        model = Album
        query = session.query(Album)
```

```
        parser = parser
    name = Filter(fields.Str(), operator=ILike)
    genre = Filter(fields.List(fields.Str), operator=In)
```

1.2.4 Customizing query format

By default, **filteralchemy** uses two underscores to separate field names and operator names for non-default operators (e.g., "value__gt=5"). Multiple values for the same field are passed using repeated query parameters (e.g., "foo=&foo=2"). To override these defaults, set the `formatter` and `list_class` options in `Meta`:

```
from webargs.fields import DelimitedList
from filteralchemy.formatters import JsonApiFormatter

class AlbumFilterSet(FilterSet):
    class Meta:
        model = Album
        query = session.query(Album)
        formatter = JsonApiFormatter()
        list_class = DelimitedList
```

This example implements the JSON API standards for filtering, using parameters like "filter[value][in]=1,2,3".

1.3 API Reference

1.3.1 Filters

class `filteralchemy.filters.`**`Filter`**(*field=None*, *attr=None*, *label=None*, *operator=<class 'filteralchemy.operators.Equal'>*)

Base filter.

Parameters

- **`field`** (*Field*) – Field to deserialize filter parameter
- **`attr`** (*str*) – Model attribute name
- **`label`** (*str*) – Lookup key on input dictionary
- **`operator`** – Operator or filter callable

1.3.2 Filterset

class `filteralchemy.filterset.`**`FilterSet`**(*query=None*)

Example usage:

```
    from models import Album, session
    from webargs.flaskparser import parser
    from filteralchemy import FilterSet

    class AlbumFilterSet(FilterSet):
        class Meta:
            model = Album
            query = session.query(Album)
            parser = parser
```

```
query = AlbumFilterSet().filter()
```

Parameters **query** – Optional SQLAlchemy query; if not provided, use query defined on options class

class Meta

Available options:

- `model`: SQLAlchemy model class
- `query`: Query on `model`
- `fields`: Sequence of model field names to include, or a callable that

accepts a *`FilterSet`* subclass and returns a sequence of fields - `exclude`: Tuple or list of model field names to exclude, or a callable that accepts a *`FilterSet`* subclass and returns a sequence of fields - `list_class`: List field class; defaults to `List` - `converter`: `ModelConverter` instance; defaults to `ModelConverter()` - `operators`: Tuple or list of `Operator` classes - `default_operator`: Default operator; non-default operators will include

operator labels in auto-generated filter names

- `formatter`: Callable for building names of auto-generated filters
- `column_overrides`: Dictionary mapping column names to operator and field overrides
- `parser`: Webargs request parser

`FilterSet.`**`filter`**`()`

Generate a filtered query from request parameters.

Returns Filtered SQLALchemy query

CHAPTER 2

Project info

2.1 Authors

- Joshua Carp @jmcarp

2.2 License


```
Copyright 2015 Joshua Carp

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

Python Module Index

f

filteralchemy.filters, 5
filteralchemy.filterset, 5

F

Filter (class in filteralchemy.filters), 5
filter() (filteralchemy.filterset.FilterSet method), 6
filteralchemy.filters (module), 5
filteralchemy.filterset (module), 5
FilterSet (class in filteralchemy.filterset), 5
FilterSet.Meta (class in filteralchemy.filterset), 6