Eve Documentation

Release 2.1.0

Nicola Iarocci

Mar 14, 2023

CONTENTS

Version 2.1.0.

Eve is an *open source* Python REST API framework designed for human beings. It allows to effortlessly build and deploy highly customizable, fully featured RESTful Web Services.

Eve is powered by Flask and Cerberus and it offers native support for MongoDB data stores. Support for SQL, Elasticsearch and Neo4js backends is provided by community extensions.

The codebase is thoroughly tested under Python 3.7+, and PyPy.

CHAPTER

ONE

EVE IS SIMPLE

```
from eve import Eve

settings = {'DOMAIN': {'people': {}}}

app = Eve(settings=settings)
app.run()
```

The API is now live, ready to be consumed:

```
$ curl -i http://example.com/people
HTTP/1.1 200 OK
```

All you need to bring your API online is a database, a configuration file (defaults to `settings.py`) or dictionary, and a launch script. Overall, you will find that configuring and fine-tuning your API is a very simple process.

CHAPTER

TWO

FUNDING EVE

Eve REST framework is a *collaboratively funded project*. If you run a business and are using Eve in a revenue-generating product, it would make business sense to sponsor Eve development: it ensures the project that your product relies on stays healthy and actively maintained. Individual users are also welcome to make either a recurring pledge or a one time donation if Eve has helped you in your work or personal projects. Every single sign-up makes a significant impact towards making Eve possible.

You can support Eve development by pledging on GitHub, Patreon, or PayPal.

- Become a Backer on GitHub
- Become a Backer on Patreon
- Donate via PayPal (one time)

2.1 Foreword

Read this before you get started with Eve. This hopefully answers some questions about the purpose and goals of the project, and when you should or should not be using it.

2.1.1 Philosophy

You have data stored somewhere and you want to expose it to your users through a RESTful Web API. Eve is the tool that allows you to do so.

Eve provides a robust, feature rich, REST-centered API implementation, and you just need to configure your API settings and behavior, plug in your datasource, and you're good to go. See *Features* for a list of features available to Eve-powered APIs. You might want to check the *REST API for Humans* slide deck too.

API settings are stored in a standard Python module (defaults to `settings.py`), which makes customization quite a trivial task. It is also possible to extend some key features, namely *Authentication and Authorization*, *Data Validation* and Data Access, by providing the Eve engine with custom objects.

2.1.2 A little context

At Gestionale Amica we had been working hard on a full featured, Python powered, RESTful Web API. We learned quite a few things on REST best patterns, and we had a chance to put Python's renowned web capabilities to the test. Then, at EuroPython 2012, I had the opportunity to share what we learned. My talk sparked quite a bit of interest, and even after a few months had passed, the slides were still receiving a lot of hits every day. I kept receiving emails asking for source code examples and whatnot. After all, a REST API lies in the future of every web-oriented developer, and who isn't one these days?

So, I thought, perhaps I could take the proprietary, closed code (codenamed 'Adam') and refactor it "just a little bit", so that it could fit a much wider number of use cases. I could then release it as an open source project. Well it turned out to be slightly more complex than that but finally here it is, and of course it's called Eve.

2.1.3 REST, Flask and MongoDB

The slides from my EuroPython talk, *Developing RESTful Web APIs with Flask and MongoDB*, are available online. You might want to check them out to understand why and how certain design decisions were made, especially with regards to REST implementation.

2.1.4 BSD License

A large number of open source projects you find today are GPL Licensed. While the GPL has its time and place, it should most certainly not be your go-to license for your next open source project.

A project that is released as GPL cannot be used in any commercial product without the product itself also being offered as open source.

The MIT, BSD, ISC, and Apache2 licenses are great alternatives to the GPL that allow your open-source software to be used freely in proprietary, closed-source software.

Eve is released under terms of the BSD License. See *Licensing*.

2.2 REST API for Humans

I have been introducing Eve at several conferences and meetups. A few people suggested that I post the slides on the Eve website, so here it is: a quick rundown on Eve features, along with a few code snippets and examples. Hopefully it will do a good job in letting you decide whether Eve is valid solution for your use case.

- REST API for Humans @ SpeakerDeck

2.2.1 Conferences

Eve REST API for Humans™ has been presented at the following events so far:

- PyConWeb 2018, Munich
- PyCon Belarus 2018, Kiev
- Codemotion 2017, Rome
- PiterPy 2016, St. Petersburg
- Percona Live 2015, Amsterdam
- EuroPython 2014, Berlin

- Python Meetup, Helsinki
- PyCon Italy 2014, Florence
- PyCon Sweden 2014, Stockholm
- FOSDEM 2014, Brussels

Want this talk delivered at your conference? Get in touch!

2.3 Installation

This part of the documentation covers the installation of Eve. The first step to using any software package is getting it properly installed.

Installing Eve is simple with pip:

```
$ pip install eve
```

2.3.1 Development Version

Eve is actively developed on GitHub, where the code is always available. If you want to work with the development version of Eve, there are two ways: you can either let *pip* pull in the development version, or you can tell it to operate on a git checkout. Either way, virtualenv is recommended.

Get the git checkout in a new virtualenv and run in development mode.

```
$ git clone https://github.com/pyeve/eve.git
Cloning into 'eve'...
...

$ cd eve
$ virtualenv venv
...
Installing setuptools, pip, wheel...
done.

$ . venv/bin/activate
$ pip install .
...
Successfully installed ...
```

This will pull in the dependencies and activate the git head as the current version inside the virtualenv. Then all you have to do is run `git pull origin` to update to the latest version.

To just get the development version without git, do this instead:

```
$ mkdir eve
$ cd eve
$ virtualenv venv
$ . venv/bin/activate
$ pip install git+https://github.com/pyeve/eve.git
...
Successfully installed ...
```

And you're done!

2.4 Quickstart

Eager to get started? This page gives a first introduction to Eve.

2.4.1 Prerequisites

- You already have Eve installed. If you do not, head over to the *Installation* section.
- MongoDB is installed.
- An instance of MongoDB is running.

2.4.2 A Minimal Application

A minimal Eve application looks something like this:

```
from eve import Eve
app = Eve()

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

Just save it as run.py. Next, create a new text file with the following content:

```
DOMAIN = {'people': {}}
```

Save it as settings.py in the same directory where run.py is stored. This is the Eve configuration file, a standard Python module, and it is telling Eve that your API is comprised of just one accessible resource, `people`.

Now your are ready to launch your API.

```
$ python run.py
 * Running on http://127.0.0.1:5000/
```

Now you can consume the API:

```
$ curl -i http://127.0.0.1:5000
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 82
Server: Eve/0.0.5-dev Werkzeug/0.8.3 Python/2.7.3
Date: Wed, 27 Mar 2013 16:06:44 GMT
```

Congratulations, your GET request got a nice response back. Let's look at the payload:

```
{
  "_links": {
    "child": [
      {
        "href": "people",
```

(continues on next page)

(continued from previous page)

```
        "title": "people"
      }
    ]
  }
}
```

API entry points adhere to the *HATEOAS* principle and provide information about the resources accessible through the API. In our case there's only one child resource available, that being `people`.

Try requesting `people` now:

```
$ curl http://127.0.0.1:5000/people
```

```
{
  "_items": [],
  "_links": {
    "self": {
      "href": "people",
      "title": "people"
    },
    "parent": {
      "href": "/",
      "title": "home"
    }
  },
  "_meta": {
    "max_results": 25,
    "page": 1,
    "total": 0
  }
}
```

This time we also got an `_items` list. The `_links` are relative to the resource being accessed, so you get a link to the parent resource (the home page) and to the resource itself. If you got a timeout error from pymongo, make sure the prerequisites are met. Chances are that the `mongod` server process is not running.

By default Eve APIs are read-only:

```
$ curl -X DELETE http://127.0.0.1:5000/people
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>405 Method Not Allowed</title>
<h1>Method Not Allowed</h1>
<p>The method DELETE is not allowed for the requested URL.</p>
```

Since we didn't provide any database detail in settings.py, Eve has no clue about the real content of the `people` collection (it might even be non-existent) and seamlessly serves an empty resource, as we don't want to let API users down.

2.4.3 Database Interlude

Let's connect to a database by adding the following lines to settings.py:

```
# Let's just use the local mongod instance. Edit as needed.

# Please note that MONGO_HOST and MONGO_PORT could very well be left
# out as they already default to a bare bones local 'mongod' instance.
MONGO_HOST = 'localhost'
MONGO_PORT = 27017

# Skip this block if your db has no auth. But it really should.
MONGO_USERNAME = '<your username>'
MONGO_PASSWORD = '<your password>'
# Name of the database on which the user can be authenticated,
# needed if --auth mode is enabled.
MONGO_AUTH_SOURCE = '<dbname>'

MONGO_DBNAME = 'apitest'
```

Due to MongoDB *laziness*, we don't really need to create the database collections. Actually we don't even need to create the database: GET requests on an empty/non-existent DB will be served correctly (`200 OK` with an empty collection); DELETE/PATCH/PUT will receive appropriate responses (`404 Not Found`), and POST requests will create database and collections as needed. However, such an auto-managed database will perform very poorly since it lacks indexes and any sort of optimization.

2.4.4 A More Complex Application

So far our API has been read-only. Let's enable the full spectrum of CRUD operations:

```
# Enable reads (GET), inserts (POST) and DELETE for resources/collections
# (if you omit this line, the API will default to ['GET'] and provide
# read-only access to the endpoint).
RESOURCE_METHODS = ['GET', 'POST', 'DELETE']

# Enable reads (GET), edits (PATCH), replacements (PUT) and deletes of
# individual items  (defaults to read-only item access).
ITEM_METHODS = ['GET', 'PATCH', 'PUT', 'DELETE']
```

`RESOURCE_METHODS` lists methods allowed at resource endpoints (`/people`) while `ITEM_METHODS` lists the methods enabled at item endpoints (`/people/<ObjectId>`). Both settings have a global scope and will apply to all endpoints. You can then enable or disable HTTP methods at individual endpoint level, as we will soon see.

Since we are enabling editing we also want to enable proper data validation. Let's define a schema for our `people` resource.

```
schema = {
    # Schema definition, based on Cerberus grammar. Check the Cerberus project
    # (https://github.com/pyeve/cerberus) for details.
    'firstname': {
        'type': 'string',
        'minlength': 1,
        'maxlength': 10,
```

(continues on next page)

(continued from previous page)

```
    },
    'lastname': {
        'type': 'string',
        'minlength': 1,
        'maxlength': 15,
        'required': True,
        # talk about hard constraints! For the purpose of the demo
        # 'lastname' is an API entry-point, so we need it to be unique.
        'unique': True,
    },
    # 'role' is a list, and can only contain values from 'allowed'.
    'role': {
        'type': 'list',
        'allowed': ["author", "contributor", "copy"],
    },
    # An embedded 'strongly-typed' dictionary.
    'location': {
        'type': 'dict',
        'schema': {
            'address': {'type': 'string'},
            'city': {'type': 'string'}
        },
    },
    'born': {
        'type': 'datetime',
    },
}
```

For more information on validation see *Data Validation*.

Now let's say that we want to further customize the `people` endpoint. We want to:

- set the item title to `person`
- add an extra *custom item endpoint* at `/people/<lastname>`
- override the default *cache control directives*
- disable DELETE for the `/people` endpoint (we enabled it globally)

Here is how the complete `people` definition looks in our updated settings.py file:

```
people = {
    # 'title' tag used in item links. Defaults to the resource title minus
    # the final, plural 's' (works fine in most cases but not for 'people')
    'item_title': 'person',

    # by default the standard item entry point is defined as
    # '/people/<ObjectId>'. We leave it untouched, and we also enable an
    # additional read-only entry point. This way consumers can also perform
    # GET requests at '/people/<lastname>'.
    'additional_lookup': {
        'url': 'regex("[\w]+")',
        'field': 'lastname'
    },
```

(continues on next page)

(continued from previous page)

```
    # We choose to override global cache-control directives for this resource.
    'cache_control': 'max-age=10,must-revalidate',
    'cache_expires': 10,

    # most global settings can be overridden at resource level
    'resource_methods': ['GET', 'POST'],

    'schema': schema
}
```

Finally we update our domain definition:

```
DOMAIN = {
    'people': people,
}
```

Save settings.py and launch run.py. We can now insert documents at the `people` endpoint:

```
$ curl -d '[{"firstname": "barack", "lastname": "obama"}, {"firstname": "mitt", "lastname
↪": "romney"}]' -H 'Content-Type: application/json' http://127.0.0.1:5000/people
HTTP/1.0 201 OK
```

We can also update and delete items (but not the whole resource since we disabled that). We can also perform GET requests against the new `lastname` endpoint:

```
$ curl -i http://127.0.0.1:5000/people/obama
HTTP/1.0 200 OK
Etag: 28995829ee85d69c4c18d597a0f68ae606a266cc
Last-Modified: Wed, 21 Nov 2012 16:04:56 GMT
Cache-Control: 'max-age=10,must-revalidate'
Expires: 10
...
```

```
{
    "firstname": "barack",
    "lastname": "obama",
    "_id": "50acfba938345b0978fccad7"
    "updated": "Wed, 21 Nov 2012 16:04:56 GMT",
    "created": "Wed, 21 Nov 2012 16:04:56 GMT",
    "_links": {
        "self": {"href": "people/50acfba938345b0978fccad7", "title": "person"},
        "parent": {"href": "/", "title": "home"},
        "collection": {"href": "people", "title": "people"}
    }
}
```

Cache directives and item title match our new settings. See *Features* for a complete list of features available and more usage examples.

2.5 Features

Below is a list of main features that any EVE-powered APIs can expose.

2.5.1 Emphasis on REST

The Eve project aims to provide the best possible REST-compliant API implementation. Fundamental REST principles like *separation of concerns*, *stateless and layered system*, *cacheability*, *uniform interface* have been kept into consideration while designing the core API.

2.5.2 Full range of CRUD operations

APIs can support the full range of CRUD operations. Within the same API, you can have a read-only resource accessible at one endpoint, along with a fully editable resource at another endpoint. The following table shows Eve's implementation of CRUD via REST:

Action	HTTP Verb	Context
Create	POST	Collection
Create	PUT	Document
Replace	PUT	Document
Read	GET, HEAD	Collection/Document
Update	PATCH	Document
Delete	DELETE	Collection/Document

Overriding HTTP Methods

As a fallback for the odd client not supporting any of these methods, the API will gladly honor `X-HTTP-Method-Override` requests. For example a client not supporting the `PATCH` method could send a `POST` request with a `X-HTTP-Method-Override: PATCH` header. The API would then perform a `PATCH`, overriding the original request method.

2.5.3 Customizable resource endpoints

By default, Eve will make known database collections available as resource endpoints (persistent identifiers in REST idiom). So a database `people` collection will be available at the `example.com/people` API endpoint. You can customize the URIs though, so the API endpoint could become, say, `example.com/customers/overseas`. Consider the following request:

```
$ curl -i http://myapi.com/people
HTTP/1.1 200 OK
```

The response payload will look something like this:

```
{
    "_items": [
        {
            "firstname": "Mark",
            "lastname": "Green",
            "born": "Sat, 23 Feb 1985 12:00:00 GMT",
```

(continues on next page)

(continued from previous page)

```
            "role": ["copy", "author"],
            "location": {"city": "New York", "address": "4925 Lacross Road"},
            "_id": "50bf198338345b1c604faf31",
            "_updated": "Wed, 05 Dec 2012 09:53:07 GMT",
            "_created": "Wed, 05 Dec 2012 09:53:07 GMT",
            "_etag": "ec5e8200b8fa0596afe9ca71a87f23e71ca30e2d",
            "_links": {
                "self": {"href": "people/50bf198338345b1c604faf31", "title": "person"},
            },
        },
        ...
    ],
    "_meta": {
        "max_results": 25,
        "total": 70,
        "page": 1
    },
    "_links": {
        "self": {"href": "people", "title": "people"},
        "parent": {"href": "/", "title": "home"}
    }
}
```

The `_items` list contains the requested data. Along with its own fields, each item provides some important, additional fields:

Field	Description
`_created`	item creation date.
`_updated`	item last updated on.
`_etag`	ETag, to be used for concurrency control and conditional requests.
`_id`	unique item key, also needed to access the individual item endpoint.

These additional fields are automatically handled by the API (clients don't need to provide them when adding/editing resources).

The `_meta` field provides pagination data and will only be there if *Pagination* has been enabled (it is by default). The `_links` list provides HATEOAS directives.

Sub Resources

Endpoints support sub-resources so you could have something like: `people/<contact_id>/invoices`. When setting the `url` rule for such an endpoint you would use a regex and assign a field name to it:

```
invoices = {
    'url': 'people/<regex("[a-f0-9]{24}"):contact_id>/invoices'
    ...
```

Then, a GET to the following endpoint:

```
people/51f63e0838345b6dcd7eabff/invoices
```

would cause the underlying database to be queried like this:

```
{'contact_id': '51f63e0838345b6dcd7eabff'}
```

And this one:

```
people/51f63e0838345b6dcd7eabff/invoices?where={"number": 10}
```

would be queried like:

```
{'contact_id': '51f63e0838345b6dcd7eabff', "number": 10}
```

Please note that when designing your API, most of the time you can get away without resorting to sub-resources. In the example above the same result would be achieved by simply exposing an `invoices` endpoint that clients could query this way:

```
invoices?where={"contact_id": 51f63e0838345b6dcd7eabff}
```

or

```
invoices?where={"contact_id": 51f63e0838345b6dcd7eabff, "number": 10}
```

It's mostly a design choice, but keep in mind that when it comes to enabling individual document endpoints you might incur performance hits. This otherwise legit GET request:

```
people/<contact_id>/invoices/<invoice_id>
```

would cause a two fields lookup on the database. This is not ideal and also not really needed, as `<invoice_id>` is a unique field. By contrast, if you had a simple resource endpoint the document lookup would happen on a single field:

```
invoices/<invoice_id>
```

Endpoints that supports sub-resources will have a specific behavior on `DELETE` operations. A `DELETE` to the following endpoint:

```
people/51f63e0838345b6dcd7eabff/invoices
```

would cause the deletion of all the documents that match follow query:

```
{'contact_id': '51f63e0838345b6dcd7eabff'}
```

Therefore, for sub-resource endpoints, only the documents satisfying the endpoint semantic will be deleted. This differs from the standard behavior, whereas a delete operation on a collection enpoint will cause the deletion of all the documents in the collection.

Another example. A `DELETE` to the following item endpoint:

```
people/51f63e0838345b6dcd7eabff/invoices/1
```

would cause the deletion all the documents matched by the follow query:

```
{'contact_id': '51f63e0838345b6dcd7eabff', "<invoice_id>": 1}
```

This behaviour enables support for typical tree structures, where the id of the resource alone is not necessarily a primary key by itself.

2.5.4 Customizable, multiple item endpoints

Resources can or cannot expose individual item endpoints. API consumers could get access to `people`, `people/<ObjectId>` and `people/Doe`, but only to `/works`. When you do grant access to item endpoints, you can define up to two lookups, both defined with regexes. The first will be the primary endpoint and will match your database primary key structure (i.e., an `ObjectId` in a MongoDB database).

```
$ curl -i http://myapi.com/people/521d6840c437dc0002d1203c
HTTP/1.1 200 OK
Etag: 28995829ee85d69c4c18d597a0f68ae606a266cc
Last-Modified: Wed, 21 Nov 2012 16:04:56 GMT
...
```

The second, which is optional and read-only, will match a field with unique values since Eve will retrieve only the first match anyway.

```
$ curl -i http://myapi.com/people/Doe
HTTP/1.1 200 OK
Etag: 28995829ee85d69c4c18d597a0f68ae606a266cc
Last-Modified: Wed, 21 Nov 2012 16:04:56 GMT
...
```

Since we are accessing the same item, in both cases the response payload will look something like this:

```
{
    "firstname": "John",
    "lastname": "Doe",
    "born": "Thu, 27 Aug 1970 14:37:13 GMT",
    "role": ["author"],
    "location": {"city": "Auburn", "address": "422 South Gay Street"},
    "_id": "50acfba938345b0978fccad7"
    "_updated": "Wed, 21 Nov 2012 16:04:56 GMT",
    "_created": "Wed, 21 Nov 2012 16:04:56 GMT",
    "_etag": "28995829ee85d69c4c18d597a0f68ae606a266cc",
    "_links": {
        "self": {"href": "people/50acfba938345b0978fccad7", "title": "person"},
        "parent": {"href": "/", "title": "home"},
        "collection": {"href": "people", "title": "people"}
    }
}
```

As you can see, item endpoints provide their own HATEOAS directives.

Please Note

According to REST principles resource items should only have one unique identifier. Eve abides by providing one default endpoint per item. Adding a secondary endpoint is a decision that should be pondered carefully.

Consider our example above. Even without the `people/<lastname>` endpoint, a client could always retrieve a person by querying the resource endpoint by last name: `people/?where={"lastname": "Doe"}`. Actually the whole example is fubar, as there could be multiple people sharing the same last name, but you get the idea.

2.5.5 Filtering

Resource endpoints allow consumers to retrieve multiple documents. Query strings are supported, allowing for filtering and sorting. Both native Mongo queries and Python conditional expressions are supported.

Here we are asking for all documents where `lastname` value is `Doe`:

```
http://myapi.com/people?where={"lastname": "Doe"}
```

With `curl` you would go like this:

```
$ curl -i -g http://myapi.com/people?where={%22lastname%22:%20%22Doe%22}
HTTP/1.1 200 OK
```

Filtering on embedded document fields is possible:

```
http://myapi.com/people?where={"location.city": "San Francisco"}
```

Date fields are also easy to query on:

```
http://myapi.com/people?where={"born": {"$gte":"Wed, 25 Feb 1987 17:00:00 GMT"}}
```

Date values should conform to RFC1123. Should you need a different format, you can change the `DATE_FORMAT` setting.

In general you will find that most MongoDB queries "just work". Should you need it, `MONGO_QUERY_BLACKLIST` allows you to blacklist unwanted operators.

Native Python syntax works like this:

```
$ curl -i http://myapi.com/people?where=lastname=="Doe"
HTTP/1.1 200 OK
```

Both syntaxes allow for conditional and logical And/Or operators, however nested and combined.

Filters are enabled by default on all document fields. However, the API maintainer can choose to disable them all and/or whitelist allowed ones (see `ALLOWED_FILTERS` in *Global Configuration*). If scraping, or fear of DB DoS attacks by querying on non-indexed fields is a concern, then whitelisting allowed filters is the way to go.

You also have the option to validate the incoming filters against the resource's schema and refuse to apply the filtering if any filters are invalid, by using the `VALIDATE_FILTERING` system setting (see *Global Configuration*)

2.5.6 Pretty Printing

You can pretty print the response by specifying a query parameter named `pretty`:

```
$ curl -i http://myapi.com/people?pretty
HTTP/1.1 200 OK

{
    "_items": [
        {
            "_updated": "Tue, 19 Apr 2016 08:19:00 GMT",
            "firstname": "John",
            "lastname": "Doe",
            "born": "Thu, 27 Aug 1970 14:37:13 GMT",
```

(continues on next page)

(continued from previous page)

```
            "role": [
                "author"
            ],
            "location": {
                "city": "Auburn",
                "address": "422 South Gay Street"
            },
            "_links": {
                "self": {
                    "href": "people/5715e9f438345b3510d27eb8",
                    "title": "person"
                }
            },
            "_created": "Tue, 19 Apr 2016 08:19:00 GMT",
            "_id": "5715e9f438345b3510d27eb8",
            "_etag": "86dc6b45fe7e2f41f1ca53a0e8fda81224229799"
        },
        ...
    ]
}
```

2.5.7 Sorting

Sorting is supported as well:

```
$ curl -i http://myapi.com/people?sort=city,-lastname
HTTP/1.1 200 OK
```

Would return documents sorted by city and then by lastname (descending). As you can see you simply prepend a minus to the field name if you need the sort order to be reversed for a field.

The MongoDB data layer also supports native MongoDB syntax:

```
http://myapi.com/people?sort=[("lastname", -1)]
```

which translates to the following `curl` request:

```
$ curl -i http://myapi.com/people?sort=[(%22lastname%22,%20-1)]
HTTP/1.1 200 OK
```

Would return documents sorted by lastname in descending order.

Sorting is enabled by default and can be disabled both globally and/or at resource level (see `SORTING` in *Global Configuration* and `sorting` in *Domain Configuration*). It is also possible to set the default sort at every API endpoints (see `default_sort` in *Domain Configuration*).

Please note

Always use double quotes to wrap field names and values. Using single quotes will result in `400 Bad Request` responses.

2.5.8 Pagination

Resource pagination is enabled by default in order to improve performance and preserve bandwidth. When a consumer requests a resource, the first N items matching the query are served, and links to subsequent/previous pages are provided with the response. Default and maximum page size is customizable, and consumers can request specific pages via the query string:

```
$ curl -i http://myapi.com/people?max_results=20&page=2
HTTP/1.1 200 OK
```

Of course you can mix all the available query parameters:

```
$ curl -i http://myapi.com/people?where={"lastname": "Doe"}&sort=[("firstname", 1)]&
↪page=5
HTTP/1.1 200 OK
```

Pagination can be disabled. Please note that, for clarity, the above example is not properly escaped. If using `curl`, refer to the examples provided in *Filtering*.

2.5.9 HATEOAS

Hypermedia as the Engine of Application State (HATEOAS) is enabled by default. Each GET response includes a `_links` section. Links provide details on their `relation` relative to the resource being accessed, and a `title`. Relations and titles can then be used by clients to dynamically updated their UI, or to navigate the API without knowing its structure beforehand. An example:

```
{
    "_links": {
        "self": {
            "href": "people",
            "title": "people"
        },
        "parent": {
            "href": "/",
            "title": "home"
        },
        "next": {
            "href": "people?page=2",
            "title": "next page"
        },
        "last": {
            "href": "people?page=10",
            "title": "last page"
        }
    }
}
```

A GET request to the API home page (the API entry point) will be served with a list of links to accessible resources. From there, any client could navigate the API just by following the links provided with every response.

HATEOAS links are always relative to the API entry point, so if your API home is at `examples.com/api/v1`, the `self` link in the above example would mean that the *people* endpoint is located at `examples.com/api/v1/people`.

Please note that `next`, `previous`, `last` and `related` items will only be included when appropriate.

Disabling HATEOAS

HATEOAS can be disabled both at the API and/or resource level. Why would you want to turn HATEOAS off? Well, if you know that your client application is not going to use the feature, then you might want to save on both bandwidth and performance.

2.5.10 Rendering

Eve responses are automatically rendered as JSON (the default) or XML, depending on the request `Accept` header. Inbound documents (for inserts and edits) are in JSON format.

```
$ curl -H "Accept: application/xml" -i http://myapi.com
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
...
```

```
<resource>
    <link rel="child" href="people" title="people" />
    <link rel="child" href="works" title="works" />
</resource>
```

Default renderers might be changed by editing `RENDERERS` value in the settings file.

```
RENDERERS = [
    'eve.render.JSONRenderer',
    'eve.render.XMLRenderer'
]
```

You can create your own renderer by subclassing `eve.render.Renderer`. Each renderer should set valid `mime` attr and have `.render()` method implemented. Please note that at least one renderer must always be enabled.

2.5.11 Conditional Requests

Each resource representation provides information on the last time it was updated (`Last-Modified`), along with an hash value computed on the representation itself (`ETag`). These headers allow clients to perform conditional requests by using the `If-Modified-Since` header:

```
$ curl -H "If-Modified-Since: Wed, 05 Dec 2012 09:53:07 GMT" -i http://myapi.com/people/
↪521d6840c437dc0002d1203c
HTTP/1.1 200 OK
```

or the `If-None-Match` header:

```
$ curl -H "If-None-Match: 1234567890123456789012345678901234567890" -i http://myapi.com/
↪people/521d6840c437dc0002d1203c
HTTP/1.1 200 OK
```

2.5.12 Data Integrity and Concurrency Control

API responses include a `ETag` header which also allows for proper concurrency control. An `ETag` is a hash value representing the current state of the resource on the server. Consumers are not allowed to edit (`PATCH` or `PUT`) or delete (`DELETE`) a resource unless they provide an up-to-date `ETag` for the resource they are attempting to edit. This prevents overwriting items with obsolete versions.

Consider the following workflow:

```
$ curl -H "Content-Type: application/json" -X PATCH -i http://myapi.com/people/
↪521d6840c437dc0002d1203c -d '{"firstname": "ronald"}'
HTTP/1.1 428 PRECONDITION REQUIRED
```

We attempted an edit (`PATCH`), but we did not provide an `ETag` for the item so we got a `428 PRECONDITION REQUIRED` back. Let's try again:

```
$ curl -H "If-Match: 1234567890123456789012345678901234567890" -H "Content-Type:␣
↪application/json" -X PATCH -i http://myapi.com/people/521d6840c437dc0002d1203c -d '{
↪"firstname": "ronald"}'
HTTP/1.1 412 PRECONDITION FAILED
```

What went wrong this time? We provided the mandatory `If-Match` header, but it's value did not match the `ETag` computed on the representation of the item currently stored on the server, so we got a `412 PRECONDITION FAILED`. Again!

```
$ curl -H "If-Match: 80b81f314712932a4d4ea75ab0b76a4eea613012" -H "Content-Type:␣
↪application/json" -X PATCH -i http://myapi.com/people/50adfa4038345b1049c88a37 -d '{
↪"firstname": "ronald"}'
HTTP/1.1 200 OK
```

Finally! And the response payload looks something like this:

```
{
    "_status": "OK",
    "_updated": "Fri, 23 Nov 2012 08:11:19 GMT",
    "_id": "50adfa4038345b1049c88a37",
    "_etag": "372fbbebf54dfe61742556f17a8461ca9a6f5a11"
    "_links": {"self": "..."}
}
```

This time we got our patch in, and the server returned the new `ETag`. We also get the new `_updated` value, which eventually will allow us to perform subsequent *conditional requests*.

Concurrency control applies to all edition methods: `PATCH` (edit), `PUT` (replace), `DELETE` (delete).

Disabling concurrency control

If your use case requires, you can opt to completely disable concurrency control. ETag match checks can be disabled by setting the `IF_MATCH` configuration variable to `False` (see *Global Configuration*). When concurrency control is disabled no ETag is provided with responses. You should be careful about disabling this feature, as you would effectively open your API to the risk of older versions replacing your documents. Alternatively, ETag match checks can be made optional by the client if `ENFORCE_IF_MATCH` is disabled. When concurrency check enforcement is disabled, requests with the `If-Match` header will be processed as conditional requests, and requests made without the `If-Match` header will not be processed as conditional.

2.5.13 Bulk Inserts

A client may submit a single document for insertion:

```
$ curl -d '{"firstname": "barack", "lastname": "obama"}' -H 'Content-Type: application/
↪json' http://myapi.com/people
HTTP/1.1 201 OK
```

In this case the response payload will just contain the relevant document metadata:

```
{
    "_status": "OK",
    "_updated": "Thu, 22 Nov 2012 15:22:27 GMT",
    "_id": "50ae43339fa12500024def5b",
    "_etag": "749093d334ebd05cf7f2b7dbfb7868605578db2c"
    "_links": {"self": {"href": "people/50ae43339fa12500024def5b", "title": "person"}}
}
```

When a `201 Created` is returned following a POST request, the `Location` header is also included with the response. Its value is the URI to the new document.

In order to reduce the number of loopbacks, a client might also submit multiple documents with a single request. All it needs to do is enclose the documents in a JSON list:

```
$ curl -d '[{"firstname": "barack", "lastname": "obama"}, {"firstname": "mitt", "lastname
↪": "romney"}]' -H 'Content-Type: application/json' http://myapi.com/people
HTTP/1.1 201 OK
```

The response will be a list itself, with the state of each document:

```
{
    "_status": "OK",
    "_items": [
        {
            "_status": "OK",
            "_updated": "Thu, 22 Nov 2012 15:22:27 GMT",
            "_id": "50ae43339fa12500024def5b",
            "_etag": "749093d334ebd05cf7f2b7dbfb7868605578db2c"
            "_links": {"self": {"href": "people/50ae43339fa12500024def5b", "title":
↪"person"}}
        },
        {
            "_status": "OK",
            "_updated": "Thu, 22 Nov 2012 15:22:27 GMT",
            "_id": "50ae43339fa12500024def5c",
            "_etag": "62d356f623c7d9dc864ffa5facc47dced4ba6907"
            "_links": {"self": {"href": "people/50ae43339fa12500024def5c", "title":
↪"person"}}
        }
    ]
}
```

When multiple documents are submitted the API takes advantage of MongoDB *bulk insert* capabilities which means that not only there's just one request traveling from the client to the remote API, but also that a single loopback is performed between the API server and the database.

In case of successful multiple inserts, keep in mind that the `Location` header only returns the URI of the first created document.

2.5.14 Data Validation

Data validation is provided out-of-the-box. Your configuration includes a schema definition for every resource managed by the API. Data sent to the API to be inserted/updated will be validated against the schema, and a resource will only be updated if validation passes.

```
$ curl -d '[{"firstname": "bill", "lastname": "clinton"}, {"firstname": "mitt", "lastname
↪": "romney"}]' -H 'Content-Type: application/json' http://myapi.com/people
HTTP/1.1 201 OK
```

The response will contain a success/error state for each item provided in the request:

```
{
    "_status": "ERR",
    "_error": "Some documents contains errors",
    "_items": [
        {
            "_status": "ERR",
            "_issues": {"lastname": "value 'clinton' not unique"}
        },
        {
            "_status": "OK",
        }
    ]
]
```

In the example above, the first document did not validate so the whole request has been rejected.

When all documents pass validation and are inserted correctly the response status is `201 Created`. If any document fails validation the response status is `422 Unprocessable Entity`, or any other error code defined by `VALIDATION_ERROR_STATUS` configuration.

For more information see *Data Validation*.

2.5.15 Extensible Data Validation

Data validation is based on the Cerberus validation system and therefore it is extensible, so you can adapt it to your specific use case. Say that your API can only accept odd numbers for a certain field value; you can extend the validation class to validate that. Or say you want to make sure that a VAT field actually matches your own country VAT algorithm; you can do that too. As a matter of fact, Eve's MongoDB data-layer itself extends Cerberus validation by implementing the `unique` schema field constraint. For more information see *Data Validation*.

2.5.16 Editing a Document (PATCH)

Clients can edit a document with the `PATCH` method, while `PUT` will replace it. `PATCH` cannot remove a field, but only update its value.

Consider the following schema:

```
'entity': {
    'name': {
        'type': 'string',
        'required': True
    },
    'contact': {
        'type': 'dict',
        'required': True,
        'schema': {
            'phone': {
                'type': 'string',
                'required': False,
                'default': '1234567890'
            },
            'email': {
                'type': 'string',
                'required': False,
                'default': 'abc@efg.com'
            },
        }
    }
}
```

Two notations: `{contact: {email: 'an email'}}` and `{contact.email: 'an email'}` can be used to update the `email` field in the `contact` subdocument.

Keep in mind that `PATCH` cannot remove a field, but only update existing values. Also, by default `PATCH` will normalize missing body fields that have default values defined in the schema. Consider the schema above. If your `PATCH` has a body like this:

```
{'contact.email': 'xyz@gmail.com'}
```

and targets this document:

```
{
  'name': 'test account',
  'contact': {'email': '123@yahoo.com', 'phone': '9876543210'}
}
```

Then the updated document will look like this:

```
{
  'name': 'test account',
  'contact': {
    'email': 'xyz@gmail.com',
    'phone': '1234567890'
  }
}
```

That is, `contact.phone` has been reset to its default value. This might not been the desired behavior. To change it, you can set `normalize_on_patch` (or `NORMALIZE_ON_PATCH` globally) to `False`. Now the updated document will look like this:

```
{
  'name': 'test account',
  'contact': {
    'email': '123@yahoo.com',
    'phone': '9876543210'
  }
}
```

2.5.17 Resource-level Cache Control

You can set global and individual cache-control directives for each resource.

```
$ curl -i http://myapi
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 131
Cache-Control: max-age=20
Expires: Tue, 22 Jan 2013 09:34:34 GMT
Server: Eve/0.0.3 Werkzeug/0.8.3 Python/2.7.3
Date: Tue, 22 Jan 2013 09:34:14 GMT
```

The response above includes both `Cache-Control` and `Expires` headers. These will minimize load on the server since cache-enabled consumers will perform resource-intensive request only when really needed.

2.5.18 API Versioning

I'm not too fond of API versioning. I believe that clients should be intelligent enough to deal with API updates transparently, especially since Eve-powered API support HATEOAS. When versioning is a necessity, different API versions should be isolated instances since so many things could be different between versions: caching, URIs, schemas, validation, and so on. URI versioning (http://api.example.com/v1/...) is supported.

2.5.19 Document Versioning

Eve supports automatic version control of documents. By default, this setting is turned off, but it can be turned globally or configured individually for each resource. When enabled, Eve begins automatically tracking changes to documents and adds the fields `_version` and `_latest_version` when retrieving documents.

Behind the scenes, Eve stores document versions in shadow collections that parallels the collection of each primary resource that Eve defines. New document versions are automatically added to this collection during normal POST, PUT, and PATCH operations. A special new query parameter is available when GETing an item that provides access to the document versions. Access a specific version with `?version=VERSION`, access all versions with `?version=all`, and access diffs of all versions with `?version=diffs`. Collection query features like projections, pagination, and sorting work with `all` and `diff` except for sorting which does not work on `diff`.

It is important to note that there are a few non-standard scenarios which could produce unexpected results when versioning is turned on. In particular, document history will not be saved when modifying collections outside of the Eve generated API. Also, if at anytime the `VERSION` field gets removed from the primary document (which cannot happen through the API when versioning is turned on), a subsequent write will re-initialize the `VERSION` number with

`VERSION` = 1. At this time there will be multiple versions of the document with the same version number. In normal practice, `VERSIONING` can be enable without worry for any new collection or even an existing collection which has not previously had versioning enabled.

Additionally, there are caching corner cases unique to document versions. A specific document version includes the `_latest_version` field, the value of which will change when a new document version is created. To account for this, Eve determines the time `_latest_version` changed (the timestamp of the last update to the primary document) and uses that value to populate the `Last-Modified` header and check the `If-Modified-Since` conditional cache validator of specific document version queries. Note that this will be different from the timestamp in the version's last updated field. The etag for a document version does not change when `_latest_version` changes, however. This results in two corner cases. First, because Eve cannot determine if the client's `_latest_version` is up to date from an ETag alone, a query using only `If-None-Match` for cache validation of old document versions will always have its cache invalidated. Second, a version fetched and cached in the same second that multiple new versions are created can receive incorrect `Not Modified` responses on ensuing `GET` queries due to `Last-Modified` values having a resolution of one second and the static etag values not providing indication of the changes. These are both highly unlikely scenarios, but an application expecting multiple edits per second should account for the possibility of holing stale `_latest_version` data.

For more information see and *Global Configuration* and *Domain Configuration*.

2.5.20 Authentication

Customizable Basic Authentication (RFC-2617), Token-based authentication and HMAC-based Authentication are supported. OAuth2 can be easily integrated. You can lockdown the whole API, or just some endpoints. You can also restrict CRUD commands, like allowing open read-only access while restricting edits, inserts and deletes to authorized users. Role-based access control is supported as well. For more information see *Authentication and Authorization*.

2.5.21 CORS Cross-Origin Resource Sharing

Eve-powered APIs can be accessed by the JavaScript contained in web pages. Disabled by default, CORS allows web pages to work with REST APIs, something that is usually restricted by most browsers 'same domain' security policy. The `X_DOMAINS` setting allows to specify which domains are allowed to perform CORS requests. A list of regular expressions may be defined in `X_DOMAINS_RE`, which is useful for websites with dynamic ranges of subdomains. Make sure to anchor and escape the regexes properly, for example `X_DOMAINS_RE = ['^http://sub-\d{3}\.example\.com$']`.

2.5.22 JSONP Support

In general you don't really want to add JSONP when you can enable CORS instead:

> There have been some criticisms raised about JSONP. Cross-origin resource sharing (CORS) is a more recent method of getting data from a server in a different domain, which addresses some of those criticisms. All modern browsers now support CORS making it a viable cross-browser alternative (source.)

There are circumstances however when you do need JSONP, like when you have to support legacy software (IE6 anyone?)

To enable JSONP in Eve you just set `JSONP_ARGUMENT`. Then, any valid request with `JSONP_ARGUMENT` will get back a response wrapped with said argument value. For example if you set `JSON_ARGUMENT = 'callback'`:

```
$ curl -i http://localhost:5000/?callback=hello
hello(<JSON here>)
```

Requests with no `callback` argument will be served with no JSONP.

2.5.23 Read-only by default

If all you need is a read-only API, then you can have it up and running in a matter of minutes.

2.5.24 Default and Nullable Values

Fields can have default values and nullable types. When serving POST (create) requests, missing fields will be assigned the configured default values. See `default` and `nullable` keywords in *Schema Definition* for more information.

2.5.25 Predefined Database Filters

Resource endpoints will only expose (and update) documents that match a predefined filter. This allows for multiple resource endpoints to seamlessly target the same database collection. A typical use-case would be a hypothetical `people` collection on the database being used by both the `/admins` and `/users` API endpoints.

See also

- *Advanced Datasource Patterns*
- *Predefined Database Filters*

2.5.26 Projections

This feature allows you to create dynamic views of collections and documents, or more precisely, to decide what fields should or should not be returned, using a 'projection'. Put another way, Projections are conditional queries where the client dictates which fields should be returned by the API.

```
$ curl -i -G http://myapi.com/people --data-urlencode 'projection={"lastname": 1, "born
↪": 1}'
HTTP/1.1 200 OK
```

The query above will only return *lastname* and *born* out of all the fields available in the 'people' resource. You can also exclude fields:

```
$ curl -i -G http://myapi.com/people --data-urlencode 'projection={"born": 0}'
HTTP/1.1 200 OK
```

The above will return all fields but *born*. Please note that key fields such as ID_FIELD, DATE_CREATED, DATE_UPDATED etc. will still be included with the payload. Also keep in mind that some database engines, Mongo included, do not allow for mixing of inclusive and exclusive selections.

See also

- *Limiting the Fieldset Exposed by the API Endpoint*
- *Leveraging Projections to optimize the handling of media files*

2.5.27 Embedded Resource Serialization

If a document field is referencing a document in another resource, clients can request the referenced document to be embedded within the requested document.

Clients have the power to activate document embedding on per-request basis by means of a query parameter. Suppose you have a `emails` resource configured like this:

```
DOMAIN = {
    'emails': {
        'schema': {
            'author': {
                'type': 'objectid',
                'data_relation': {
                    'resource': 'users',
                    'field': '_id',
                    'embeddable': True
                },
            },
            'subject': {'type': 'string'},
            'body': {'type': 'string'},
        }
    }
```

A GET like this: `/emails?embedded={"author":1}` would return a fully embedded users document, whereas the same request without the `embedded` argument would just return the user `ObjectId`. Embedded resource serialization is available at both resource and item (`/emails/<id>/?embedded={"author":1}`) endpoints.

Embedding can be enabled or disabled both at global level (by setting `EMBEDDING` to either `True` or `False`) and at resource level (by toggling the `embedding` value). Furthermore, only fields with the `embeddable` value explicitly set to `True` will allow the embedding of referenced documents.

Embedding also works with a data_relation to a specific version of a document, but the schema looks a little bit different. To enable the data_relation to a specific version, add `'version': True` to the data_relation block. You'll also want to change the `type` to `dict` and add the `schema` definition shown below.

```
DOMAIN = {
    'emails': {
        'schema': {
            'author': {
                'type': 'dict',
                'schema': {
                    '_id': {'type': 'objectid'},
                    '_version': {'type': 'integer'}
                },
                'data_relation': {
                    'resource': 'users',
                    'field': '_id',
                    'embeddable': True,
                    'version': True,
                },
            },
            'subject': {'type': 'string'},
            'body': {'type': 'string'},
        }
```

(continues on next page)

(continued from previous page)

```
    }
```

As you can see, `'version': True` changes the expected value of a data_relation field to a dictionary with fields names `data_relation['field']` and `VERSION`. With `'field': '_id'` in the data_relation definition above and `VERSION = '_version'` in the Eve config, the value of the data_relation in this scenario would be a dictionary with fields `_id` and `_version`.

Predefined Resource Serialization

It is also possible to elect some fields for predefined resource serialization. If the listed fields are embeddable and they are actually referencing documents in other resources (and embedding is enabled for the resource), then the referenced documents will be embedded by default. Clients can still opt out from field that are embedded by default:

```
$ curl -i http://example.com/people/?embedded={"author": 0}
HTTP/1.1 200 OK
```

Limitations

Currently we support embedding of documents by references located in any subdocuments (nested dicts and lists). For example, a query `/invoices/?embedded={"user.friends":1}` will return a document with `user` and all his `friends` embedded, but only if `user` is a subdocument and `friends` is a list of reference (it could be a list of dicts, nested dict, etc.). This feature is about serialization on GET requests. There's no support for POST, PUT or PATCH of embedded documents.

Document embedding is enabled by default.

Please note

When it comes to MongoDB, what embedded resource serialization deals with is *document references* (linked documents), something different from *embedded documents*, also supported by Eve (see MongoDB Data Model Design). Embedded resource serialization is a nice feature that can really help with normalizing your data model for the client. However, when deciding whether to enable it or not, especially by default, keep in mind that each embedded resource being looked up will require a database lookup, which can easily lead to performance issues.

2.5.28 Soft Delete

Eve provides an optional "soft delete" mode in which deleted documents continue to be stored in the database and are able to be restored, but still act as removed items in response to API requests. Soft delete is disabled by default, but can be enabled globally using the `SOFT_DELETE` configuration setting, or individually configured at the resource level using the domain configuration `soft_delete` setting. See *Global Configuration* and *Domain Configuration* for more information on enabling and configuring soft delete.

When soft deletion is enabled, callbacks attached to `on_delete_resource_originals` and `on_delete_resource_originals_<resource_name>` events will receive both deleted and not deleted documents via the `originals` argument (see *Event Hooks*).

Behavior

With soft delete enabled, DELETE requests to individual items and resources respond just as they do for a traditional "hard" delete. Behind the scenes, however, Eve does not remove deleted items from the database, but instead patches the document with a `_deleted` meta field set to `true`. (The name of the `_deleted` field is configurable. See *Global Configuration*.) All requests made when soft delete is enabled filter against or otherwise account for the `_deleted` field.

The `_deleted` field is automatically added and initialized to `false` for all documents created while soft delete is enabled. Documents created prior to soft delete being enabled and which therefore do not define the `_deleted` field in the database will still include `_deleted: false` in API response data, added by Eve during response construction. PUTs or PATCHes to these documents will add the `_deleted` field to the stored documents, set to `false`.

Responses to GET requests for soft deleted documents vary slightly from responses to missing or "hard" deleted documents. GET requests for soft deleted documents will still respond with `404 Not Found` status codes, but the response body will contain the soft deleted document with `_deleted: true`. Documents embedded in the deleted document will not be expanded in the response, regardless of any default settings or the contents of the request's `embedded` query param. This is to ensure that soft deleted documents included in `404` responses reflect the state of a document when it was deleted, and do not to change if embedded documents are updated.

By default, resource level GET requests will not include soft deleted items in their response. This behavior matches that of requests after a "hard" delete. If including deleted items in the response is desired, the `show_deleted` query param can be added to the request. (the `show_deleted` param name is configurable. See *Global Configuration*) Eve will respond with all documents, deleted or not, and it is up to the client to parse returned documents' `_deleted` field. The `_deleted` field can also be explicitly filtered against in a request, allowing only deleted documents to be returned using a `?where={"_deleted": true}` query.

Soft delete is enforced in the data layer, meaning queries made by application code using the `app.data.find_one` and `app.data.find` methods will automatically filter out soft deleted items. Passing a request object with `req.show_deleted == True` or a lookup dictionary that explicitly filters on the `_deleted` field will override the default filtering.

Restoring Soft Deleted Items

PUT or PATCH requests made to a soft deleted document will restore it, automatically setting `_deleted` to `false` in the database. Modifying the `_deleted` field directly is not necessary (or allowed). For example, using PATCH requests, only the fields to be changed in the restored version would be specified, or an empty request would be made to restore the document as is. The request must be made with proper authorization for write permission to the soft deleted document or it will be refused.

Be aware that, should a previously soft deleted document be restored, there is a chance that an eventual unique field might end up being now duplicated in two different documents: the restored one, and another which might have been stored with the same field value while the original (now restored) was in 'deleted' state. This is because soft deleted documents are ignored when validating the *unique* rule for new or updated documents.

Versioning

Soft deleting a versioned document creates a new version of that document with `_deleted` set to `true`. A GET request to the deleted version will receive a `404 Not Found` response as described above, while previous versions will continue to respond with `200 OK`. Responses to `?version=diff` or `?version=all` will include the deleted version as if it were any other.

Data Relations

The Eve `data_relation` validator will not allow references to documents that have been soft deleted. Attempting to create or update a document with a reference to a soft deleted document will fail just as if that document had been hard deleted. Existing data relations to documents that are soft deleted remain in the database, but requests requiring embedded document serialization of those relations will resolve to a null value. Again, this matches the behavior of relations to hard deleted documents.

Versioned data relations to a deleted document version will also fail to validate, but relations to versions prior to deletion or after restoration of the document are allowed and will continue to resolve successfully.

Considerations

Disabling soft delete after use in an application requires database maintenance to ensure your API remains consistent. With soft delete disabled, requests will no longer filter against or handle the `_deleted` field, and documents that were soft deleted will now be live again on your API. It is therefore necessary when disabling soft delete to perform a data migration to remove all documents with `_deleted == True`, and recommended to remove the `_deleted` field from documents where `_deleted == False`. Enabling soft delete in an existing application is safe, and will maintain documents deleted from that point on.

2.5.29 Event Hooks

Pre-Request Event Hooks

When a GET/HEAD, POST, PATCH, PUT, DELETE request is received, both a `on_pre_<method>` and a `on_pre_<method>_<resource>` event is raised. You can subscribe to these events with multiple callback functions.

```
>>> def pre_get_callback(resource, request, lookup):
...  print('A GET request on the "%s" endpoint has just been received!' % resource)

>>> def pre_contacts_get_callback(request, lookup):
...  print('A GET request on the contacts endpoint has just been received!')

>>> app = Eve()

>>> app.on_pre_GET += pre_get_callback
>>> app.on_pre_GET_contacts += pre_contacts_get_callback

>>> app.run()
```

Callbacks will receive the resource being requested, the original `flask.request` object and the current lookup dictionary as arguments (only exception being the `on_pre_POST` hook which does not provide a `lookup` argument).

Dynamic Lookup Filters

Since the `lookup` dictionary will be used by the data layer to retrieve resource documents, developers may choose to alter it in order to add custom logic to the lookup query.

```
def pre_GET(resource, request, lookup):
    # only return documents that have a 'username' field.
    lookup["username"] = {'$exists': True}

app = Eve()

app.on_pre_GET += pre_GET
app.run()
```

Altering the lookup dictionary at runtime would have similar effects to applying *Predefined Database Filters* via configuration. However, you can only set static filters via configuration whereas by hooking to the `on_pre_<METHOD>` events you are allowed to set dynamic filters instead, which allows for additional flexibility.

Post-Request Event Hooks

When a GET, POST, PATCH, PUT, DELETE method has been executed, both a `on_post_<method>` and `on_post_<method>_<resource>` event is raised. You can subscribe to these events with multiple callback functions. Callbacks will receive the resource accessed, original *flask.request* object and the response payload.

```
>>> def post_get_callback(resource, request, payload):
...  print('A GET on the "%s" endpoint was just performed!' % resource)

>>> def post_contacts_get_callback(request, payload):
...  print('A get on "contacts" was just performed!')

>>> app = Eve()

>>> app.on_post_GET += post_get_callback
>>> app.on_post_GET_contacts += post_contacts_get_callback

>>> app.run()
```

Database event hooks

Database event hooks work like request event hooks. These events are fired before and after a database action. Here is an example of how events are configured:

```
>>> def add_signature(resource, response):
...     response['SIGNATURE'] = "A %s from eve" % resource

>>> app = Eve()
>>> app.on_fetched_item += add_signature
```

You may use flask's `abort()` to interrupt the database operation:

```
>>> from flask import abort

```

(continues on next page)

(continued from previous page)

```
>>> def check_update_access(resource, updates, original):
...     abort(403)

>>> app = Eve()
>>> app.on_insert_item += check_update_access
```

The events are fired for resources and items if the action is available for both. And for each action two events will be fired:

- Generic: `on_<action_name>`
- With the name of the resource: `on_<action_name>_<resource_name>`

Let's see an overview of what events are available:

Action	What	When	Event name / method signature
Fetch	Resource	After	on_fetched_resource def event(resource_name, response)
			on_fetched_resource_<resource_name> def event(response)
	Item	After	on_fetched_item def event(resource_name, response)
			on_fetched_item_<resource_name> def event(response)
	Diffs	After	on_fetched_diffs def event(resource_name, response)
			on_fetched_diffs_<resource_name> def event(response)
Insert	Items	Before	on_insert def event(resource_name, items)
			on_insert_<resource_name> def event(items)
		After	on_inserted def event(resource_name,

Fetch Events

These are the fetch events with their method signature:

- `on_fetched_resource(resource_name, response)`
- `on_fetched_resource_<resource_name>(response)`
- `on_fetched_item(resource_name, response)`
- `on_fetched_item_<resource_name>(response)`
- `on_fetched_diffs(resource_name, response)`
- `on_fetched_diffs_<resource_name>(response)`

They are raised when items have just been read from the database and are about to be sent to the client. Registered callback functions can manipulate the items as needed before they are returned to the client.

```
>>> def before_returning_items(resource_name, response):
...  print('About to return items from "%s" ' % resource_name)

>>> def before_returning_contacts(response):
...  print('About to return contacts')

>>> def before_returning_item(resource_name, response):
...  print('About to return an item from "%s" ' % resource_name)

>>> def before_returning_contact(response):
...  print('About to return a contact')

>>> app = Eve()
>>> app.on_fetched_resource += before_returning_items
>>> app.on_fetched_resource_contacts += before_returning_contacts
>>> app.on_fetched_item += before_returning_item
>>> app.on_fetched_item_contacts += before_returning_contact
```

It is important to note that item fetch events will work with *Document Versioning* for specific document versions like `?version=5` and all document versions with `?version=all`. Accessing diffs of all versions with `?version=diffs` will only work with the diffs fetch events. Note that diffs returns partial documents which should be handled in the callback.

Insert Events

These are the insert events with their method signature:

- `on_insert(resource_name, items)`
- `on_insert_<resource_name>(items)`
- `on_inserted(resource_name, items)`
- `on_inserted_<resource_name>(items)`

When a POST requests hits the API and new items are about to be stored in the database, these events are fired:

- `on_insert` for every resource endpoint.
- `on_insert_<resource_name>` for the specific <*resource_name*> resource endpoint.

Callback functions could hook into these events to arbitrarily add new fields or edit existing ones.

After the items have been inserted, these two events are fired:

- on_inserted for every resource endpoint.
- on_inserted_<resource_name> for the specific <*resource_name*> resource endpoint.

Validation errors

Items passed to these events as arguments come in a list. And only those items that passed validation are sent.

Example:

```
>>> def before_insert(resource_name, items):
...  print('About to store items to "%s" ' % resource_name)

>>> def after_insert_contacts(items):
...  print('About to store contacts')

>>> app = Eve()
>>> app.on_insert += before_insert
>>> app.on_inserted_contacts += after_insert_contacts
```

Replace Events

These are the replace events with their method signature:

- on_replace(resource_name, item, original)
- on_replace_<resource_name>(item, original)
- on_replaced(resource_name, item, original)
- on_replaced_<resource_name>(item, original)

When a PUT request hits the API and an item is about to be replaced after passing validation, these events are fired:

- on_replace for any resource item endpoint.
- on_replace_<resource_name> for the specific resource endpoint.

item is the new item which is about to be stored. *original* is the item in the database that is being replaced. Callback functions could hook into these events to arbitrarily add or update *item* fields, or to perform other accessory action.

After the item has been replaced, these other two events are fired:

- on_replaced for any resource item endpoint.
- on_replaced_<resource_name> for the specific resource endpoint.

Update Events

These are the update events with their method signature:

- `on_update(resource_name, updates, original)`
- `on_update_<resource_name>(updates, original)`
- `on_updated(resource_name, updates, original)`
- `on_updated_<resource_name>(updates, original)`

When a PATCH request hits the API and an item is about to be updated after passing validation, these events are fired *before* the item is updated:

- `on_update` for any resource endpoint.
- `on_update_<resource_name>` is fired only when the <*resource_name*> endpoint is hit.

Here *updates* stands for updates being applied to the item and *original* is the item in the database that is about to be updated. Callback functions could hook into these events to arbitrarily add or update fields in *updates*, or to perform other accessory action.

After the item has been updated:

- `on_updated` is fired for any resource endpoint.
- `on_updated_<resource_name>` is fired only when the <*resource_name*> endpoint is hit.

Please note

Please be aware that `last_modified` and `etag` headers will always be consistent with the state of the items on the database (they won't be updated to reflect changes eventually applied by the callback functions).

Delete Events

These are the delete events with their method signature:

- `on_delete_item(resource_name, item)`
- `on_delete_item_<resource_name>(item)`
- `on_deleted_item(resource_name, item)`
- `on_deleted_item_<resource_name>(item)`
- `on_delete_resource(resource_name)`
- `on_delete_resource_<resource_name>()`
- `on_delete_resource_originals(originals, lookup)`
- `on_delete_resource_originals_<resource_name>(originals, lookup)`
- `on_deleted_resource(resource_name)`
- `on_deleted_resource_<resource_name>()`

Items

When a DELETE request hits an item endpoint and *before* the item is deleted, these events are fired:

- `on_delete_item` for any resource hit by the request.
- `on_delete_item_<resource_name>` for the specific <*resource_name*> item endpoint hit by the DELETE.

After the item has been deleted the `on_deleted_item(resource_name, item)` and `on_deleted_item_<resource_name>(item)` are raised.

item is the item being deleted. Callback functions could hook into these events to perform accessory actions. And no you can't arbitrarily abort the delete operation at this point (you should probably look at *Data Validation*, or eventually disable the delete command altogether).

Resources

If you were brave enough to enable the DELETE command on resource endpoints (allowing for wipeout of the entire collection in one go), then you can be notified of such a disastrous occurrence by hooking a callback function to the `on_delete_resource(resource_name)` or `on_delete_resource_<resource_name>()` hooks.

- `on_delete_resource_originals` for any resource hit by the request after having retrieved the originals documents.
- `on_delete_resource_originals_<resource_name>` for the specific <*resource_name*> resource endpoint hit by the DELETE after having retrieved the original document.

NOTE: those two event are useful in order to perform some business logic before the actual remove operation given the look up and the list of originals

Aggregation event hooks

You can also attach one or more callbacks to your aggregation endpoints. The `before_aggregation` event is fired when an aggregation is about to be performed. Any attached callback function will receive both the endpoint name and the aggregation pipeline as arguments. The pipeline can then be altered if needed.

```
>>> def on_aggregate(endpoint, pipeline):
...   pipeline.append({"$unwind": "$tags"})

>>> app = Eve()
>>> app.before_aggregation += on_aggregate
```

The `after_aggregation` event is fired when the aggregation has been performed. An attached callback function could leverage this event to modify the documents before they are returned to the client.

```
>>> def alter_documents(endpoint, documents):
...   for document in documents:
...     document['hello'] = 'well, hello!'

>>> app = Eve()
>>> app.after_aggregation += alter_documents
```

For more information on aggregation support, see *MongoDB Aggregation Framework*

Please note

To provide seamless event handling features Eve relies on the Events package.

2.5.30 Rate Limiting

API rate limiting is supported on a per-user/method basis. You can set the number of requests and the time window for each HTTP method. If the requests limit is hit within the time window, the API will respond with `429 Request limit exceeded` until the timer resets. Users are identified by the Authentication header or (when missing) by the client IP. When rate limiting is enabled, appropriate `X-RateLimit-` headers are provided with every API response. Suppose that the rate limit has been set to 300 requests every 15 minutes, this is what a user would get after hitting a endpoint with a single request:

```
X-RateLimit-Remaining: 299
X-RateLimit-Limit: 300
X-RateLimit-Reset: 1370940300
```

You can set different limits for each one of the supported methods (GET, POST, PATCH, DELETE).

Please Note

Rate Limiting is disabled by default, and needs a Redis server running when enabled. A tutorial on Rate Limiting is forthcoming.

2.5.31 Custom ID Fields

Eve allows to extend its standard data type support. In the *Handling custom ID fields* tutorial we see how it is possible to use UUID values instead of MongoDB default ObjectIds as unique document identifiers.

2.5.32 File Storage

Media files (images, pdf, etc.) can be uploaded as `media` document fields. Upload is done via `POST`, `PUT` and `PATCH` as usual, but using the `multipart/form-data` content-type.

Let us assume that the `accounts` endpoint has a schema like this:

```
accounts = {
    'name': {'type': 'string'},
    'pic': {'type': 'media'},
    ...
}
```

With curl we would `POST` like this:

```
$ curl -F "name=john" -F "pic=@profile.jpg" http://example.com/accounts
```

For optimized performance files are stored in GridFS by default. Custom `MediaStorage` classes can be implemented and passed to the application to support alternative storage systems. A `FileSystemMediaStorage` class is in the works, and will soon be included with the Eve package.

As a proper developer guide is not available yet, you can peek at the MediaStorage source if you are interested in developing custom storage classes.

Serving media files as Base64 strings

When a document is requested media files will be returned as Base64 strings,

```
{
    '_items': [
        {
            '_updated':'Sat, 05 Apr 2014 15:52:53 GMT',
            'pic':'iVBORw0KGgoAAAANSUhEUgAAA4AAAAOACA...',
        }
    ]
    ...
}
```

However, if the `EXTENDED_MEDIA_INFO` list is populated (it isn't by default) the payload format will be different. This flag allows passthrough from the driver of additional meta fields. For example, using the MongoDB driver, fields like `content_type`, `name` and `length` can be added to this list and will be passed-through from the underlying driver.

When `EXTENDED_MEDIA_INFO` is used the field will be a dictionary whereas the file itself is stored under the `file` key and other keys are the meta fields. Suppose that the flag is set like this:

```
EXTENDED_MEDIA_INFO = ['content_type', 'name', 'length']
```

Then the output will be something like

```
{
    '_items': [
        {
            '_updated':'Sat, 05 Apr 2014 15:52:53 GMT',
            'pic': {
                'file': 'iVBORw0KGgoAAAANSUhEUgAAA4AAAAOACA...',
                'content_type': 'text/plain',
                'name': 'test.txt',
                'length': 8129
            }
        }
    ]
    ...
}
```

For MongoDB, further fields can be found in the driver documentation.

If you have other means to retrieve the media files (custom Flask endpoint for example) then the media files can be excluded from the payload by setting to `False` the `RETURN_MEDIA_AS_BASE64_STRING` flag. This takes into account if `EXTENDED_MEDIA_INFO` is used.

Serving media files at a dedicated endpoint

While returning files embedded as Base64 fields is the default behaviour, you can opt for serving them at a dedicated media endpoint. You achieve that by setting `RETURN_MEDIA_AS_URL` to `True`. When this feature is enabled document fields contain urls to the correspondent files, which are served at the media endpoint.

You can change the default media endpoint (`media`) by updating the `MEDIA_BASE_URL` and `MEDIA_ENDPOINT` setting. Suppose you are storing your images on Amazon S3 via a custom `MediaStorage` subclass. You would probably set your media endpoint like so:

```
# disable default behaviour
RETURN_MEDIA_AS_BASE64_STRING = False

# return media as URL instead
RETURN_MEDIA_AS_URL = True

# set up the desired media endpoint
MEDIA_BASE_URL = 'https://s3-us-west-2.amazonaws.com'
MEDIA_ENDPOINT = 'media'
```

Setting `MEDIA_BASE_URL` is optional. If no value is set, then the API base address will be used when building the URL for `MEDIA_ENDPOINT`.

Partial media downloads

When files are served at a dedicated endpoint, clients can request partial downloads. This allows them to provide features such as optimized pause/resume (with no need to restart the download). To perform a partial download, make sure the `Range` header is added the the client request.

```
$ curl http://localhost/media/yourfile -i -H "Range: bytes=0-10"
HTTP/1.1 206 PARTIAL CONTENT
Date: Sun, 20 Aug 2017 14:26:42 GMT
Content-Type: audio/mp4
Content-Length: 11
Connection: keep-alive
Content-Range: bytes 0-10/23671
Last-Modified: Sat, 19 Aug 2017 03:25:36 GMT
Accept-Ranges: bytes

abcdefghilm
```

In the snippet above, we see curl requesting the first chunk of a file.

Leveraging Projections to optimize the handling of media files

Clients and API maintainers can exploit the *Projections* feature to include/exclude media fields from response payloads.

Suppose that a client stored a document with an image. The image field is called *image* and it is of `media` type. At a later time, the client wants to retrieve the same document but, in order to optimize for speed and since the image is cached already, it does not want to download the image along with the document. It can do so by requesting the field to be trimmed out of the response payload:

```
$ curl -i http://example.com/people/<id>?projection={"image": 0}
HTTP/1.1 200 OK
```

The document will be returned with all its fields except the *image* field.

Moreover, when setting the `datasource` property for any given resource endpoint it is possible to explicitly exclude fields (of `media` type, but also of any other type) from default responses:

```
people = {
    'datasource': {
        'projection': {'image': 0}
    },
    ...
}
```

Now clients will have to explicitly request the image field to be included with response payloads by sending requests like this one:

```
$ curl -i http://example.com/people/<id>?projection={"image": 1}
HTTP/1.1 200 OK
```

See also

- *Configuration*
- *Advanced Datasource Patterns*

for details on the `datasource` setting.

Note on media files as `multipart/form-data`

If you are uploading media files as `multipart/form-data` all the additional fields except the file fields will be treated as `strings` for all field validation purposes. If you have already defined some of the resource fields to be of different type (boolean, number, list etc) the validation rules for these fields would fail, preventing you to successffully submit your resource.

If you still want to be able to perform field validation in this case, you will have to turn on `MULTIPART_FORM_FIELDS_AS_JSON` in your settings file in order to treat the incoming fields as JSON encoded strings and still be able to validate your fields.

Please note, that in case you indeed turn on `MULTIPART_FORM_FIELDS_AS_JSON` you will have to submit all resource fields as properly encoded JSON strings.

For example a `number` should be submited as `1234` (as you would normally expect). A `boolean` will have to be send as `true` (note the lowercase `t`). A `list` of strings as `["abc", "xyz"]`. And finally a `string`, which is the thing that will most likely trip, you will have to be submitted as `"'abc'"` (note that it is surrounded with double quotes). If ever in doubt if what you are submitting is a valid JSON string you can try passing it from the JSON Validator at http://jsonlint.com/ to be sure that it is correct.

Using lists of media

When using lists of media, there is no way to submit these in the default configuration. Enable `AUTO_COLLAPSE_MULTI_KEYS` and `AUTO_CREATE_LISTS` to make this possible. This allows to send multiple values for one key in `multipart/form-data` requests and in this way upload a list of files.

2.5.33 GeoJSON

The MongoDB data layer supports geographic data structures encoded in GeoJSON format. All GeoJSON objects supported by MongoDB are available:

- `Point`
- `Multipoint`
- `LineString`
- `MultiLineString`
- `Polygon`
- `MultiPolygon`
- `GeometryCollection`

All these objects are implemented as native Eve data types (see *Schema Definition*) so they are are subject to the proper validation.

In the example below we are extending the *people* endpoint by adding a `location` field of type Point.

```
people = {
    ...
    'location': {
        'type': 'point'
    },
    ...
}
```

Storing a contact along with its location is pretty straightforward:

```
$ curl -d '[{"firstname": "barack", "lastname": "obama", "location": {"type":"Point",
↪"coordinates":[100.0,10.0]}}]' -H 'Content-Type: application/json' http://127.0.0.
↪1:5000/people
HTTP/1.1 201 OK
```

Eve also supports GeoJSON `Feature` and `FeatureCollection` objects, which are not explicitely mentioned in MongoDB documentation. GeoJSON specification allows object to contain any number of members (name/value pairs). Eve validation was implemented to be more strict, allowing only two members. This restriction can be disabled by setting `ALLOW_CUSTOM_FIELDS_IN_GEOJSON` to `True`.

Querying GeoJSON Data

As a general rule all MongoDB geospatial query operators and their associated geometry specifiers are supported. In this example we are using the $near operator to query for all contacts living in a location within 1000 meters from a certain point:

```
?where={"location": {"$near": {"$geometry": {"type":"Point", "coordinates": [10.0, 20.0]}
↪, "$maxDistance": 1000}}}
```

Please refer to MongoDB documentation for details on geo queries.

2.5.34 Internal Resources

By default responses to GET requests to the home endpoint will include all the resources. The `internal_resource` setting keyword, however, allows you to make an endpoint internal, available only for internal data manipulation: no HTTP calls can be made against it and it will be excluded from the `HATEOAS` links.

An usage example would be a mechanism for logging all inserts happening in the system, something that can be used for auditing or a notification system. First we define an `internal_transaction` endpoint, which is flagged as an `internal_resource`:

```
internal_transactions = {
    'schema': {
        'entities': {
            'type': 'list',
        },
        'original_resource': {
            'type': 'string',
        },
    },
    'internal_resource': True
}
```

Now, if we access the home endpoint and `HATEOAS` is enabled, we won't get the `internal-transactions` listed (and hitting the endpoint via HTTP will return a `404`.) We can use the data layer to access our secret endpoint. Something like this:

```
from eve import Eve

def on_generic_inserted(self, resource, documents):
    if resource != 'internal_transactions':
        dt = datetime.now()
        transaction = {
            'entities':  [document['_id'] for document in documents],
            'original_resource': resource,
            config.LAST_UPDATED: dt,
            config.DATE_CREATED: dt,
        }
        app.data.insert('internal_transactions', [transaction])

app = Eve()
app.on_inserted += self.on_generic_inserted

app.run()
```

I admit that this example is as rudimentary as it can get, but hopefully it will get the point across.

2.5.35 Enhanced Logging

A number of events are available for logging via the default application logger. The standard LogRecord attributes are extended with a few request attributes:

clientip	IP address of the client performing the request.
url	Full request URL, eventual query parameters included.
method	Request method (POST, GET, etc.)

You can use these fields when logging to a file or any other destination.

Callback functions can also take advantage of the builtin logger. The following example logs application events to a file, and also logs custom messages every time a custom function is invoked.

```
import logging

from eve import Eve

def log_every_get(resource, request, payload):
    # custom INFO-level message is sent to the log file
    app.logger.info('We just answered to a GET request!')

app = Eve()
app.on_post_GET += log_every_get

if __name__ == '__main__':

    # enable logging to 'app.log' file
    handler = logging.FileHandler('app.log')

    # set a custom log format, and add request
    # metadata to each log line
    handler.setFormatter(logging.Formatter(
        '%(asctime)s %(levelname)s: %(message)s '
        '[in %(filename)s:%(lineno)d] -- ip: %(clientip)s, '
        'url: %(url)s, method:%(method)s'))

    # the default log level is set to WARNING, so
    # we have to explicitly set the logging level
    # to INFO to get our custom message logged.
    app.logger.setLevel(logging.INFO)

    # append the handler to the default application logger
    app.logger.addHandler(handler)

    # let's go
    app.run()
```

Currently only exceptions raised by the MongoDB layer and POST, PATCH and PUT methods are logged. The idea is to also add some INFO and possibly DEBUG level events in the future.

2.5.36 Operations Log

The OpLog is an API-wide log of all edit operations. Every `POST`, `PATCH` `PUT` and `DELETE` operation can be recorded to the oplog. At its core the oplog is simply a server log. What makes it a little bit different is that it can be exposed as a read-only endpoint, thus allowing clients to query it as they would with any other API endpoint.

Every oplog entry contains information about the document and the operation:

- Operation performed
- Unique ID of the document
- Update date
- Creation date
- Resource endpoint URL
- User token, if *User-Restricted Resource Access* is enabled for the endpoint
- Optional custom data

Like any other API-maintained document, oplog entries also expose:

- Entry ID
- ETag
- HATEOAS fields if that's enabled.

If `OPLOG_AUDIT` is enabled entries also expose:

- client IP
- Username or token, if available
- changes applied to the document (for `DELETE` the whole document is included).

A typical oplog entry looks like this:

```
{
    "o": "DELETE",
    "r": "people",
    "i": "542d118938345b614ea75b3c",
    "c": {...},
    "ip": "127.0.0.1",
    "u": "admin",
    "_updated": "Fri, 03 Oct 2014 08:16:52 GMT",
    "_created": "Fri, 03 Oct 2014 08:16:52 GMT",
    "_etag": "e17218fbca41cb0ee6a5a5933fb9ee4f4ca7e5d6"
    "_id": "542e5b7438345b6dadf95ba5",
    "_links": {...},
}
```

To save a little space (at least on MongoDB) field names have been shortened:

- `o` stands for operation performed
- `r` stands for resource endpoint
- `i` stands for document id
- `ip` is the client IP
- `u` stands for user (or token)

- `c` stands for changes occurred
- `extra` is an optional field which you can use to store custom data

`_created` and `_updated` are relative to the target document, which comes handy in a variety of scenarios (like when the oplog is available to clients, more on this later).

Please note that by default the `c` (changes) field is not included for `POST` operations. You can add `POST` to the `OPLOG_CHANGE_METHODS` setting (see *Global Configuration*) if you wish the whole document to be included on every insertion.

How is the oplog operated?

Seven settings are dedicated to the OpLog:

- `OPLOG` switches the oplog feature on and off. Defaults to `False`.
- `OPLOG_NAME` is the name of the oplog collection on the database. Defaults to `oplog`.
- `OPLOG_METHODS` is a list of HTTP methods to be logged. Defaults to all of them.
- `OPLOG_ENDPOINT` is the endpoint name. Defaults to `None`.
- `OPLOG_AUDIT` if enabled, IP addresses and changes are also logged. Defaults to `True`.
- `OPLOG_CHANGE_METHODS` determines which methods will log changes. Defaults to ['PATCH', 'PUT', 'DELETE'].
- `OPLOG_RETURN_EXTRA_FIELD` determines if the optional `extra` field should be returned by the `OPLOG_ENDPOINT`. Defaults to `False`.

As you can see the oplog feature is turned off by default. Also, since `OPLOG_ENDPOINT` defaults to `None`, even if you switch the feature on no public oplog endpoint will be available. You will have to explicitly set the endpoint name in order to expose your oplog to the public.

The Oplog endpoint

Since the oplog endpoint is nothing but a standard API endpoint, you can customize it. This allows for setting up custom authentication (you might want this resource to be only accessible for administrative purposes) or any other useful setting.

Note that while you can change most of its settings, the endpoint will always be read-only so setting either `resource_methods` or `item_methods` to something other than `['GET']` will serve no purpose. Also, unless you need to customize it, adding an oplog entry to the domain is not really necessary as it will be added for you automatically.

Exposing the oplog as an endpoint could be useful in scenarios where you have multiple clients (say phone, tablet, web and desktop apps) which need to stay in sync with each other and the server. Instead of hitting every single endpoint they could just access the oplog to learn all that's happened since their last access. That's a single request versus several. This is not always the best approach a client could take. Sometimes it is probably better to only query for changes on a certain endpoint. That's also possible, just query the oplog for changes occured on that endpoint.

Extending Oplog entries

Every time the oplog is about to be updated the `on_oplog_push` event is fired. You can hook one or more callback functions to this event. Callbacks receive `resource` and `entries` as arguments. The former is the resource name while the latter is a list of oplog entries which are about to be written to disk.

Your callback can add an optional `extra` field to canonical oplog entries. The field can be of any type. In this example we are adding a custom dict to each entry:

```
def oplog_extras(resource, entries):
    for entry in entries:
        entry['extra'] = {'myfield': 'myvalue'}

app = Eve()

app.on_oplog_push += oplog_extras
app.run()
```

Please note that unless you explicitly set `OPLOG_RETURN_EXTRA_FIELD` to `True`, the `extra` field will *not* be returned by the `OPLOG_ENDPOINT`.

Note: Are you on MongoDB? Consider making the oplog a capped collection. Also, in case you are wondering yes, the Eve oplog is blatantly inspired by the awesome Replica Set Oplog.

2.5.37 The Schema Endpoint

Resource schema can be exposed to API clients by enabling Eve's schema endpoint. To do so, set the `SCHEMA_ENDPOINT` configuration option to the API endpoint name from which you want to serve schema data. Once enabled, Eve will treat the endpoint as a read only resource containing JSON encoded Cerberus schema definitions, indexed by resource name. Resource visibility and authorization settings are honored, so internal resources or resources for which a request does not have read authentication will not be accessible at the schema endpoint. By default, `SCHEMA_ENDPOINT` is set to `None`.

2.5.38 MongoDB Aggregation Framework

Support for the MongoDB Aggregation Framework is built-in. In the example below (taken from PyMongo) we'll perform a simple aggregation to count the number of occurrences for each tag in the tags array, across the entire collection. To achieve this we need to pass in three operations to the pipeline. First, we need to unwind the tags array, then group by the tags and sum them up, finally we sort by count.

As python dictionaries don't maintain order you should use `SON` or collections `OrderedDict` where explicit ordering is required eg `$sort`:

```
posts = {
    'datasource': {
        'aggregation': {
            'pipeline': [
                {"$unwind": "$tags"},
                {"$group": {"_id": "$tags", "count": {"$sum": 1}}},
                {"$sort": SON([("count", -1), ("_id", -1)])}
            ]
```

(continues on next page)

(continued from previous page)

```
        }
    }
}
```

The pipeline above is static. You have the option to allow for dynamic pipelines, whereas the client will directly influence the aggregation results. Let's update the pipeline a little bit:

```
posts = {
    'datasource': {
        'aggregation': {
            'pipeline': [
                {"$unwind": "$tags"},
                {"$group": {"_id": "$tags", "count": {"$sum": "$value"}}},
                {"$sort": SON([("count", -1), ("_id", -1)])}
            ]
        }
    }
}
```

As you can see the *count* field is now going to sum the value of `$value`, which will be set by the client upon performing the request:

```
$ curl -i http://example.com/posts?aggregate={"$value": 2}
```

The request above will cause the aggregation to be executed on the server with a *count* field configured as if it was a static `{"$sum": 2}`. The client simply adds the `aggregate` query parameter and then passes a dictionary with field/value pairs. Like with all other keywords, you can change `aggregate` to a keyword of your liking, just set `QUERY_AGGREGATION` in your settings.

You can also set all options natively supported by PyMongo. For more information on aggregation see *Advanced Datasource Patterns*.

You can pass `{}` to fields which you want to ignore. Considering the following pipelines:

```
posts = {
    'datasource': {
        'aggregation': {
            'pipeline': [
                {"$match": { "name": "$name", "time": "$time"}}
                {"$unwind": "$tags"},
                {"$group": {"_id": "$tags", "count": {"$sum": 1}}},
            ]
        }
    }
}
```

If performing the following request:

```
$ curl -i http://example.com/posts?aggregate={"$name": {"$regex": "Apple"}, "$time": {}}
```

The stage `{"$match": { "name": "$name", "time": "$time"}}` in the pipeline will be executed as `{"$match": { "name": {"$regex": "Apple"}}}`. And for the following request:

```
$ curl -i http://example.com/posts?aggregate={"$name": {}, "$time": {}}
```

The stage `{"$match": { "name": "$name", "time": "$time"}}` in the pipeline will be completely skipped.

The request above will ignore `"count": {"$sum": "$value"}}`. A Custom callback functions can be attached to the `before_aggregation` and `after_aggregation` event hooks. For more information, see *Aggregation event hooks*.

Limitations

Client pagination (`?page=2`) is enabled by default. This is currently achieved by injecting a `$facet` stage contianing two sub-pipelines, total_count (`$count`) and paginated_results (`$limit` first, then `$skip`) to the very end of the aggregation pipeline after the `before_aggregation` hook. You can turn pagination off by setting `pagination` to `False` for the endpoint. Keep in mind that, when pagination is disabled, all aggregation results are included with every response. Disabling pagination might be appropriate (and actually advisable) only if the expected response payload is not huge.

Client sorting (`?sort=field1`) is not supported at aggregation endpoints. You can of course add one or more `$sort` stages to the pipeline, as we did with the example above. If you do add a `$sort` stage to the pipeline, consider adding it at the end of the pipeline. According to MongoDB's `$limit` documentation (link):

> When a `$sort` immediately precedes a `$limit` in the pipeline, the sort operation only maintains the top **n** results as it progresses, where **n** is the specified limit, and MongoDB only needs to store **n** items in memory.

As we just saw earlier, pagination adds a `$limit` stage to the end of the pipeline. So if pagination is enabled and `$sort` is the last stage of your pipeline, then the resulting combined pipeline should be optimized.

A single endpoint cannot serve both regular and aggregation results. However, since it is possible to setup multiple endpoints all serving from the same datasource (see *Multiple API Endpoints, One Datasource*), similar functionality can be easily achieved.

2.5.39 MongoDB and SQL Support

Support for single or multiple MongoDB database/servers comes out of the box. An SQLAlchemy extension provides support for SQL backends. Additional data layers can can be developed with relative ease. Visit the extensions page for a list of community developed data layers and extensions.

2.5.40 Powered by Flask

Eve is based on the Flask micro web framework. Actually, Eve itself is a Flask subclass, which means that Eve exposes all of Flask functionalities and niceties, like a built-in development server and debugger, integrated support for unittesting and an extensive documentation.

2.6 Configuration

Generally Eve configuration is best done with configuration files. The configuration files themselves are actual Python files. However, Eve will give precedence to dictionary-based settings first, then it will try to locate a file passed in `EVE_SETTINGS` environmental variable (if set) and finally it will try to locate *settings.py* or a file with filename passed to *settings* flag in constructor.

2.6.1 Configuration With Files

On startup, if *settings* flag is omitted in constructor, Eve will try to locate file named *settings.py*, first in the application folder and then in one of the application's subfolders. You can choose an alternative filename/path, just pass it as an argument when you instantiate the application. If the file path is relative, Eve will try to locate it recursively in one of the folders in your *sys.path*, therefore you have to be sure that your application root is appended to it. This is useful, for example, in testing environments, when settings file is not necessarily located in the root of your application.

```
from eve import Eve

app = Eve(settings='my_settings.py')
app.run()
```

2.6.2 Configuration With a Dictionary

Alternatively, you can choose to provide a settings dictionary. Unlike configuring Eve with the settings file, dictionary-based approach will only update Eve's default settings with your own values, rather than overwriting all the settings.

```
from eve import Eve

my_settings = {
    'MONGO_HOST': 'localhost',
    'MONGO_PORT': 27017,
    'MONGO_DBNAME': 'the_db_name',
    'DOMAIN': {'contacts': {}}
}

app = Eve(settings=my_settings)
app.run()
```

2.6.3 Development / Production

Most applications need more than one configuration. There should be at least separate configurations for the production server and the one used during development. The easiest way to handle this is to use a default configuration that is always loaded and part of the version control, and a separate configuration that overrides the values as necessary.

This is the main reason why you can override or extend the settings with the contents of the file the `EVE_SETTINGS` environment variable points to. The development/local settings could be stored in *settings.py* and then, in production, you could export EVE_SETTINGS=/path/to/production_setting.py, and you are done.

There are many alternative ways to handle development/production however. Using Python modules for configuration is very convenient, as they allow for all kinds of nice tricks, like being able to seamlessly launch the same API on both local and production systems, connecting to the appropriate database instance as needed. Consider the following example:

```
# We want to run seamlessly our API both locally and on Heroku, so:
if os.environ.get('PORT'):
    # We're hosted on Heroku! Use the MongoHQ sandbox as our backend.
    MONGO_HOST = 'alex.mongohq.com'
    MONGO_PORT = 10047
    MONGO_USERNAME = '<user>'
    MONGO_PASSWORD = '<pw>'
```

(continues on next page)

(continued from previous page)

```
    MONGO_DBNAME = '<dbname>'
else:
    # Running on local machine. Let's just use the local mongod instance.

    # Please note that MONGO_HOST and MONGO_PORT could very well be left
    # out as they already default to a bare bones local 'mongod' instance.
    MONGO_HOST = 'localhost'
    MONGO_PORT = 27017
    MONGO_USERNAME = 'user'
    MONGO_PASSWORD = 'user'
    MONGO_DBNAME = 'apitest'
```

2.6.4 Global Configuration

Besides defining the general API behavior, most global configuration settings are used to define the standard endpoint ruleset, and can be fine-tuned later, when configuring individual endpoints. Global configuration settings are always uppercase.

URL_PREFIX	URL prefix for all API endpoints. Will be used in conjunction with API_VERSION to build API endpoints (e.g., api will be rendered to /api/<endpoint>). Defaults to ''.
API_VERSION	API version. Will be used in conjunction with URL_PREFIX to build API endpoints (e.g., v1 will be rendered to /v1/<endpoint>). Defaults to ''.
ALLOWED_FILTERS	List of fields on which filtering is allowed. Entries in this list work in a hierarchical way. This means that, for instance, filtering on 'dict.sub_dict.foo' is allowed if ALLOWED_FILTERS contains any of 'dict.sub_dict.foo', 'dict.sub_dict' or 'dict'. Instead filtering on 'dict' is allowed if ALLOWED_FILTERS contains 'dict'. Can be set to [] (no filters allowed) or ['*'] (filters allowed on every field). Unless your API is comprised of just one endpoint, this global setting should be used as an on/off switch, delegating explicit whitelisting at the local level (see allowed_filters below). Defaults to ['*']. *Please note:* If API scraping or DB DoS attacks are a concern, then globally disabling filters and whitelisting valid ones at the local level is the way to go.
VALIDATE_FILTERS	Whether to validate the filters against the resource schema. Invalid filters will throw an exception. Defaults to False. Word of caution: validation on filter expressions involving fields with custom rules or types might have a considerable impact on performance. This is the case, for example, with data_relation-rule fields. Consider excluding heavy-duty fields from filters (see ALLOWED_FILTERS).
SORTING	True if sorting is supported for GET requests, otherwise False. Can be overridden by resource settings. Defaults to True.

continues on next page

Table 1 – continued from previous page

PAGINATION	`True` if pagination is enabled for `GET` requests, otherwise `False`. Can be overridden by resource settings. Defaults to `True`.
PAGINATION_LIMIT	Maximum value allowed for `QUERY_MAX_RESULTS` query parameter. Values exceeding the limit will be silently replaced with this value. You want to aim for a reasonable compromise between performance and transfer size. Defaults to 50.
PAGINATION_DEFAULT	Default value for `QUERY_MAX_RESULTS`. Defaults to 25.
OPTIMIZE_PAGINATION_FOR_SPEED	Set this to `True` to improve pagination performance. When optimization is active no count operation, which can be slow on large collections, is performed on the database. This does have a few consequences. Firstly, no document count is returned. Secondly, `HATEOAS` is less accurate: no last page link is available, and next page link is always included, even on last page. On big collections, switching this feature on can greatly improve performance. Defaults to `False` (slower performance; document count included; accurate `HATEOAS`).
QUERY_WHERE	Key for the filters query parameter. Defaults to `where`.
QUERY_SORT	Key for the sort query parameter. Defaults to `sort`.
QUERY_PROJECTION	Key for the projections query parameter. Defaults to `projection`.
QUERY_PAGE	Key for the pages query parameter. Defaults to `page`.
QUERY_MAX_RESULTS	Key for the max results query parameter. Defaults to `max_results`.
QUERY_EMBEDDED	Key for the embedding query parameter. Defaults to `embedded`.
QUERY_AGGREGATION	Key for the aggregation query parameter. Defaults to `aggregate`.
DATE_FORMAT	A Python date format used to parse and render datetime values. When serving requests, matching JSON strings will be parsed and stored as `datetime` values. In responses, `datetime` values will be rendered as JSON strings using this format. Defaults to the RFC1123 (ex RFC 822) standard `a, %d %b %Y %H:%M:%S GMT` ("Tue, 02 Apr 2013 10:29:13 GMT").
RESOURCE_METHODS	A list of HTTP methods supported at resource endpoints. Allowed values: `GET`, `POST`, `DELETE`. `POST` is used for insertions. `DELETE` will delete *all* resource contents (enable with caution). Can be overridden by resource settings. Defaults to `['GET']`.
PUBLIC_METHODS	A list of HTTP methods supported at resource endpoints, open to public access even when *Authentication and Authorization* is enabled. Can be overridden by resource settings. Defaults to `[]`.
ITEM_METHODS	A list of HTTP methods supported at item endpoints. Allowed values: `GET`, `PATCH`, `PUT` and `DELETE`. `PATCH` or, for clients not supporting PATCH, `POST` with the `X-HTTP-Method-Override` header tag, is used for item updates; `DELETE` for item deletion. Can be overridden by resource settings. Defaults to `['GET']`.

continues on next page

Table 1 – continued from previous page

PUBLIC_ITEM_METHODS	A list of HTTP methods supported at item endpoints, left open to public access when when *Authentication and Authorization* is enabled. Can be overridden by resource settings. Defaults to [].
ALLOWED_ROLES	A list of allowed *roles* for resource endpoints. Can be overridden by resource settings. See *Authentication and Authorization* for more information. Defaults to [].
ALLOWED_READ_ROLES	A list of allowed *roles* for resource endpoints with GET and OPTIONS methods. Can be overridden by resource settings. See *Authentication and Authorization* for more information. Defaults to [].
ALLOWED_WRITE_ROLES	A list of allowed *roles* for resource endpoints with POST, PUT and DELETE methods. Can be overridden by resource settings. See *Authentication and Authorization* for more information. Defaults to [].
ALLOWED_ITEM_ROLES	A list of allowed *roles* for item endpoints. See *Authentication and Authorization* for more information. Can be overridden by resource settings. Defaults to [].
ALLOWED_ITEM_READ_ROLES	A list of allowed *roles* for item endpoints with GET and OPTIONS methods. See *Authentication and Authorization* for more information. Can be overridden by resource settings. Defaults to [].
ALLOWED_ITEM_WRITE_ROLES	A list of allowed *roles* for item endpoints with PUT, PATCH and DELETE methods. See *Authentication and Authorization* for more information. Can be overridden by resource settings. Defaults to [].
ALLOW_OVERRIDE_HTTP_METHOD	Enables / Disables global the possibility to override the sent method with a header X-HTTP-METHOD-OVERRIDE.
CACHE_CONTROL	Value of the Cache-Control header field used when serving GET requests (e.g., max-age=20,must-revalidate). Leave empty if you don't want to include cache directives with API responses. Can be overridden by resource settings. Defaults to ''.
CACHE_EXPIRES	Value (in seconds) of the Expires header field used when serving GET requests. If set to a non-zero value, the header will always be included, regardless of the setting of CACHE_CONTROL. Can be overridden by resource settings. Defaults to 0.
X_DOMAINS	CORS (Cross-Origin Resource Sharing) support. Allows API maintainers to specify which domains are allowed to perform CORS requests. Allowed values are: None, a list of domains, or '*' for a wide-open API. Defaults to None.
X_DOMAINS_RE	The same setting as X_DOMAINS, but a list of regexes is allowed. This is useful for websites with dynamic ranges of subdomains. Make sure to properly anchor and escape the regexes. Invalid regexes (such as '*') are ignored. Defaults to None.
X_HEADERS	CORS (Cross-Origin Resource Sharing) support. Allows API maintainers to specify which headers are allowed to be sent with CORS requests. Allowed values are: None or a list of headers names. Defaults to None.

continues on next page

Table 1 – continued from previous page

X_EXPOSE_HEADERS	CORS (Cross-Origin Resource Sharing) support. Allows API maintainers to specify which headers are exposed within a CORS response. Allowed values are: `None` or a list of headers names. Defaults to `None`.
X_ALLOW_CREDENTIALS	CORS (Cross-Origin Resource Sharing) support. Allows API maintainers to specify if cookies can be sent by clients. The only allowed value is: `True`, any other will be ignored. Defaults to `None`.
X_MAX_AGE	CORS (Cross-Origin Resource Sharing) support. Allows to set max age for the access control allow header. Defaults to 21600.
LAST_UPDATED	Name of the field used to record a document's last update date. This field is automatically handled by Eve. Defaults to `_updated`.
DATE_CREATED	Name for the field used to record a document creation date. This field is automatically handled by Eve. Defaults to `_created`.
ID_FIELD	Name of the field used to uniquely identify resource items within the database. You want this field to be properly indexed on the database. Can be overridden by resource settings. Defaults to `_id`.
ITEM_LOOKUP	`True` if item endpoints should be generally available across the API, `False` otherwise. Can be overridden by resource settings. Defaults to `True`.
ITEM_LOOKUP_FIELD	Document field used when looking up a resource item. Can be overridden by resource settings. Defaults to `ID_FIELD`.
ITEM_URL	URL rule used to construct default item endpoint URLs. Can be overridden by resource settings. Defaults `regex("[a-f0-9]{24}")` which is MongoDB standard `Object_Id` format.
ITEM_TITLE	Title to be used when building item references, both in XML and JSON responses. Defaults to resource name, with the plural 's' stripped if present. Can and most likely will be overridden when configuring single resource endpoints.
AUTH_FIELD	Enables *User-Restricted Resource Access*. When the feature is enabled, users can only read/update/delete resource items created by themselves. The keyword contains the actual name of the field used to store the id of the user who created the resource item. Can be overridden by resource settings. Defaults to `None`, which disables the feature.
ALLOW_UNKNOWN	When `True`, this option will allow insertion of arbitrary, unknown fields to any API endpoint. Use with caution. See *Allowing the Unknown* for more information. Defaults to `False`.
PROJECTION	When `True`, this option enables the *Projections* feature. Can be overridden by resource settings. Defaults to `True`.
EMBEDDING	When `True`, this option enables the *Embedded Resource Serialization* feature. Defaults to `True`.

continues on next page

Table 1 – continued from previous page

BANDWIDTH_SAVER	When True, POST, PUT, and PATCH responses only return automatically handled fields and EXTRA_RESPONSE_FIELDS. When False, the entire document will be sent. Defaults to True.
EXTRA_RESPONSE_FIELDS	Allows to configure a list of additional document fields that should be provided with every POST response. Normally only automatically handled fields (ID_FIELD, LAST_UPDATED, DATE_CREATED, ETAG) are included in response payloads. Can be overridden by resource settings. Defaults to [], effectively disabling the feature.
RATE_LIMIT_GET	A tuple expressing the rate limit on GET requests. The first element of the tuple is the number of requests allowed, while the second is the time window in seconds. For example, (300, 60 * 15) would set a limit of 300 requests every 15 minutes. Defaults to None.
RATE_LIMIT_POST	A tuple expressing the rate limit on POST requests. The first element of the tuple is the number of requests allowed, while the second is the time window in seconds. For example (300, 60 * 15) would set a limit of 300 requests every 15 minutes. Defaults to None.
RATE_LIMIT_PATCH	A tuple expressing the rate limit on PATCH requests. The first element of the tuple is the number of requests allowed, while the second is the time window in seconds. For example (300, 60 * 15) would set a limit of 300 requests every 15 minutes. Defaults to None.
RATE_LIMIT_DELETE	A tuple expressing the rate limit on DELETE requests. The first element of the tuple is the number of requests allowed, while the second is the time window in seconds. For example (300, 60 * 15) would set a limit of 300 requests every 15 minutes. Defaults to None.
DEBUG	True to enable Debug Mode, False otherwise.
ERROR	Allows to customize the error_code field. Defaults to _error.
HATEOAS	When False, this option disables *HATEOAS*. Defaults to True.
ISSUES	Allows to customize the issues field. Defaults to _issues.
STATUS	Allows to customize the status field. Defaults to _status.
STATUS_OK	Status message returned when data validation is successful. Defaults to OK.
STATUS_ERR	Status message returned when data validation failed. Defaults to ERR.
ITEMS	Allows to customize the items field. Defaults to _items.
META	Allows to customize the meta field. Defaults to _meta
INFO	String value to include an info section, with the given INFO name, at the Eve homepage (suggested value _info). The info section will include Eve server version and API version (API_VERSION, if set). None otherwise, if you do not want to expose any server info. Defaults to None.
LINKS	Allows to customize the links field. Defaults to _links.
ETAG	Allows to customize the etag field. Defaults to _etag.

continues on next page

Table 1 – continued from previous page

IF_MATCH	True to enable concurrency control, False otherwise. Defaults to True. See *Data Integrity and Concurrency Control*.
ENFORCE_IF_MATCH	True to always enforce concurrency control when it is enabled, False otherwise. Defaults to True. See *Data Integrity and Concurrency Control*.
RENDERERS	Allows to change enabled renderers. Defaults to ['eve.render.JSONRenderer', 'eve.render.XMLRenderer'].
JSON_SORT_KEYS	True to enable JSON key sorting, False otherwise. Defaults to False.
JSON_REQUEST_CONTENT_TYPES	Supported JSON content types. Useful when you need support for vendor-specific json types. Please note: responses will still carry the standard application/json type. Defaults to ['application/json'].
VALIDATION_ERROR_STATUS	The HTTP status code to use for validation errors. Defaults to 422.
VERSIONING	Enabled documents version control when True. Can be overridden by resource settings. Defaults to False.
VERSIONS	Suffix added to the name of the primary collection to create the name of the shadow collection to store document versions. Defaults to _versions. When VERSIONING is enabled , a collection such as myresource_versions would be created for a resource with a datasource of myresource.
VERSION_PARAM	The URL query parameter used to access the specific version of a document. Defaults to version. Omit this parameter to get the latest version of a document or use *?version=all* ` to get a list of all version of the document. Only valid for individual item endpoints.
VERSION	Field used to store the version number of a document. Defaults to _version.
LATEST_VERSION	Field used to store the latest version number of a document. Defaults to _latest_version.
VERSION_ID_SUFFIX	Used in the shadow collection to store the document id. Defaults to _document. If ID_FIELD is set to _id, the document id will be stored in field _id_document.
MONGO_URI	A MongoDB URI which is used in preference of the other configuration variables.
MONGO_HOST	MongoDB server address. Defaults to localhost.
MONGO_PORT	MongoDB port. Defaults to 27017.
MONGO_USERNAME	MongoDB user name.
MONGO_PASSWORD	MongoDB password.
MONGO_DBNAME	MongoDB database name.
MONGO_OPTIONS	MongoDB keyword arguments to passed to MongoClient class __init__. Defaults to {'connect': True, 'tz_aware': True, 'appname': 'flask_app_name', 'uuidRepresentation': 'standard'}. See PyMongo mongo_client for reference.
MONGO_AUTH_SOURCE	MongoDB authorization database. Defaults to None.
MONGO_AUTH_MECHANISM	MongoDB authentication mechanism. See PyMongo Authentication Mechanisms. Defaults to None.

continues on next page

Table 1 – continued from previous page

MONGO_AUTH_MECHANISM_PROPERTIES	Specify MongoDB extra authentication mechanism properties if required. Defaults to None.
MONGO_QUERY_BLACKLIST	A list of Mongo query operators that are not allowed to be used in resource filters (?where=). Defaults to ['$where', '$regex']. Mongo JavaScript operators are disabled by default, as they might be used as vectors for injection attacks. Javascript queries also tend to be slow and generally can be easily replaced with the (very rich) Mongo query dialect.
MONGO_QUERY_WHITELIST	A list of extra Mongo query operators to allow besides the official list of allowed operators. Defaults to []. Can be overridden at endpoint (Mongo collection) level. See mongo_query_whitelist below.
MONGO_WRITE_CONCERN	A dictionary defining MongoDB write concern settings. All standard write concern settings (w, wtimeout, j, fsync) are supported. Defaults to {'w': 1}, which means 'do regular acknowledged writes' (this is also the Mongo default). Please be aware that setting 'w' to a value of 2 or greater requires replication to be active or you will be getting 500 errors (the write will still happen; Mongo will just be unable to check that it's being written to multiple servers). Can be overridden at endpoint (Mongo collection) level. See mongo_write_concern below.
DOMAIN	A dict holding the API domain definition. See *Domain Configuration*.
EXTENDED_MEDIA_INFO	A list of properties to forward from the file upload driver.
RETURN_MEDIA_AS_BASE64_STRING	Controls the embedding of the media type in the endpoint response. This is useful when you have other means of getting the binary (like custom Flask endpoints) but still want clients to be able to POST/PATCH it. Defaults to True.
RETURN_MEDIA_AS_URL	Set it to True to enable serving media files at a dedicated media endpoint. Defaults to False.
MEDIA_BASE_URL	Base URL to be used when RETURN_MEDIA_AS_URL is active. Combined with MEDIA_ENDPOINT and MEDIA_URL dictates the URL returned for media files. If None, which is the default value, the API base address will be used instead.
MEDIA_ENDPOINT	The media endpoint to be used when RETURN_MEDIA_AS_URL is enabled. Defaults to media.
MEDIA_URL	Format of a file url served at the dedicated media endpoints. Defaults to regex("[a-f0-9]{24}").
MULTIPART_FORM_FIELDS_AS_JSON	In case you are submitting your resource as multipart/form-data all form data fields will be submitted as strings, breaking any validation rules you might have on the resource fields. If you want to treat all submitted form data as JSON strings you will have to activate this setting. In that case field validation will continue working correctly. Read more about how the fields should be formatted at *Note on media files as multipart/form-data*. Defaults to False.

continues on next page

Table 1 – continued from previous page

AUTO_COLLAPSE_MULTI_KEYS	If set to True, multiple values sent with the same key, submitted using the application/x-www-form-urlencoded or multipart/form-data content types, will automatically be converted to a list of values. When using this together with AUTO_CREATE_LISTS it becomes possible to use lists of media fields. Defaults to False
AUTO_CREATE_LISTS	When submitting a non list type value for a field with type list, automatically create a one element list before running the validators. Defaults to False
OPLOG	Set it to True to enable the *Operations Log*. Defaults to False.
OPLOG_NAME	This is the name of the database collection where the *Operations Log* is stored. Defaults to oplog.
OPLOG_METHODS	List of HTTP methods which operations should be logged in the *Operations Log*. Defaults to ['DELETE', 'POST', 'PATCH', 'PUT'].
OPLOG_CHANGE_METHODS	List of HTTP methods which operations will include changes into the *Operations Log* entry. Defaults to ['DELETE', 'PATCH', 'PUT'].
OPLOG_ENDPOINT	Name of the *Operations Log* endpoint. If the endpoint is enabled it can be configured like any other API endpoint. Set it to None to disable the endpoint. Defaults to None.
OPLOG_AUDIT	Set it to True to enable the audit feature. When audit is enabled client IP and document changes are also logged to the *Operations Log*. Defaults to True.
OPLOG_RETURN_EXTRA_FIELD	When enabled, the optional extra field will be included in the payload returned by the OPLOG_ENDPOINT. Defaults to False.
SCHEMA_ENDPOINT	Name of the *The Schema Endpoint*. Defaults to None.
HEADER_TOTAL_COUNT	Custom header containing total count of items in response payloads for collection GET requests. This is handy for HEAD requests when client wants to know items count without retrieving response body. An example use case is to get the count of unread posts using where query without loading posts themselves. Defaults to X-Total-Count.
JSONP_ARGUMENT	This option will cause the response to be wrapped in a JavaScript function call if the argument is set in the request. For example if you set JSON_ARGUMENT = 'callback', then all responses to ?callback=funcname requests will be wrapped in a funcname call. Defaults to None.
BULK_ENABLED	Enables bulk insert when set to True. See *Bulk Inserts* for more information. Defaults to True.
SOFT_DELETE	Enables soft delete when set to True. See *Soft Delete* for more information. Defaults to False.
DELETED	Field name used to indicate if a document has been deleted when SOFT_DELETE is enabled. Defaults to _deleted.
SHOW_DELETED_PARAM	The URL query parameter used to include soft deleted items in resource level GET responses. Defaults to 'show_deleted'.

continues on next page

Table 1 – continued from previous page

STANDARD_ERRORS	This is a list of HTTP error codes for which a standard API response will be provided. Canonical error response includes a JSON body with actual error code and description. Set this to an empty list if you want to disable canonical responses altogether. Defaults to `[400, 401, 403, 404, 405, 406, 409, 410, 412, 422, 428]`
VALIDATION_ERROR_AS_LIST	If `True` even single field errors will be returned in a list. By default single field errors are returned as strings while multiple field errors are bundled in a list. If you want to standardize the field errors output, set this setting to `True` and you will always get a list of field issues. Defaults to `False`.
UPSERT_ON_PUT	PUT attempts to create a document if it does not exist. The URL endpoint will be used as `ID_FIELD` value (if `ID_FIELD` is included with the payload, it will be ignored). Normal validation rules apply. The response will be a `201 Created` on successful creation. Response payload will be identical the one you would get by performing a single document POST to the resource endpoint. Set to `False` to disable this feature, and a `404` will be returned instead. Defaults to `True`.
MERGE_NESTED_DOCUMENTS	If `True`, updates to nested fields are merged with the current data on `PATCH`. If `False`, the updates overwrite the current data. Defaults to `True`.
NORMALIZE_DOTTED_FIELDS	If `True`, dotted fields are parsed and processed as subdocument fields. If `False`, dotted fields are left unparsed and unprocessed, and the payload is passed to the underlying data-layer as-is. Please note that with the default Mongo layer, setting this to `False` will result in an error. Defaults to `True`.
NORMALIZE_ON_PATCH	If `True`, the patch document will be normalized according to schema. This means if a field is not included in the patch body, it will be reset to the default value in its schema. If `False`, the field which is not included in the patch body will be kept untouched. Defaults to `True`.

2.6.5 Domain Configuration

In Eve terminology, a *domain* is the definition of the API structure, the area where you design your API, fine-tune resources endpoints, and define validation rules.

`DOMAIN` is a *global configuration setting*: a Python dictionary where keys are API resources and values their definitions.

```
# Here we define two API endpoints, 'people' and 'works', leaving their
# definitions empty.
DOMAIN = {
    'people': {},
    'works': {},
    }
```

In the following two sections, we will customize the *people* resource.

Resource / Item Endpoints

Endpoint customization is mostly done by overriding some *global settings*, but other unique settings are also available. Resource settings are always lowercase.

`url`	The endpoint URL. If omitted the resource key of the `DOMAIN` dict will be used to build the URL. As an example, `contacts` would make the *people* resource available at `/contacts` (instead of `/people`). URL can be as complex as needed and can be nested relative to another API endpoint (you can have a `/contacts` endpoint and then a `/contacts/overseas` endpoint. Both are independent of each other and freely configurable). You can also use regexes to setup subresource-like endpoints. See *Sub Resources*.
`allowed_filters`	List of fields on which filtering is allowed. Entries in this list work in a hierarchical way. This means that, for instance, filtering on `'dict.sub_dict.foo'` is allowed if `allowed_filters` contains any of `'dict.sub_dict.foo`, `'dict.sub_dict'` or `'dict'`. Instead filtering on `'dict'` is allowed if `allowed_filters` contains `'dict'`. Can be set to `[]` (no filters allowed), or `['*']` (fields allowed on every field). Defaults to `['*']`. *Please note:* If API scraping or DB DoS attacks are a concern, then globally disabling filters (see `ALLOWED_FILTERS` above) and then whitelisting valid ones at the local level is the way to go.
`sorting`	`True` if sorting is enabled, `False` otherwise. Locally overrides `SORTING`.
`pagination`	`True` if pagination is enabled, `False` otherwise. Locally overrides `PAGINATION`.
`pagination_limit`	Maximum value allowed for `QUERY_MAX_RESULTS` query parameter. Values exceeding the limit will be silently replaced with this value. You want to aim for a reasonable compromise between performance and transfer size. Defaults to 50.
`resource_methods`	A list of HTTP methods supported at resource endpoint. Allowed values: `GET`, `POST`, `DELETE`. Locally overrides `RESOURCE_METHODS`. *Please note:* if you're running version 0.0.5 or earlier use the now unsupported `methods` keyword instead.
`public_methods`	A list of HTTP methods supported at resource endpoint, open to public access even when *Authentication and Authorization* is enabled. Locally overrides `PUBLIC_METHODS`.
`item_methods`	A list of HTTP methods supported at item endpoint. Allowed values: `GET`, `PATCH`, `PUT` and `DELETE`. `PATCH` or, for clients not supporting PATCH, `POST` with the `X-HTTP-Method-Override` header tag. Locally overrides `ITEM_METHODS`.
`public_item_methods`	A list of HTTP methods supported at item endpoint, left open to public access when *Authentication and Authorization* is enabled. Locally overrides `PUBLIC_ITEM_METHODS`.

continues on next page

Table 2 – continued from previous page

allowed_roles	A list of allowed *roles* for resource endpoint. See *Authentication and Authorization* for more information. Locally overrides ALLOWED_ROLES.
allowed_read_roles	A list of allowed *roles* for resource endpoint with GET and OPTIONS methods. See *Authentication and Authorization* for more information. Locally overrides ALLOWED_READ_ROLES.
allowed_write_roles	A list of allowed *roles* for resource endpoint with POST, PUT and DELETE. See *Authentication and Authorization* for more information. Locally overrides ALLOWED_WRITE_ROLES.
allowed_item_read_roles	A list of allowed *roles* for item endpoint with GET and OPTIONS methods. See *Authentication and Authorization* for more information. Locally overrides ALLOWED_ITEM_READ_ROLES.
allowed_item_write_roles	A list of allowed *roles* for item endpoint with PUT, PATH and DELETE methods. See *Authentication and Authorization* for more information. Locally overrides ALLOWED_ITEM_WRITE_ROLES.
allowed_item_roles	A list of allowed *roles* for item endpoint. See *Authentication and Authorization* for more information. Locally overrides ALLOWED_ITEM_ROLES.
cache_control	Value of the Cache-Control header field used when serving GET requests. Leave empty if you don't want to include cache directives with API responses. Locally overrides CACHE_CONTROL.
cache_expires	Value (in seconds) of the Expires header field used when serving GET requests. If set to a non-zero value, the header will always be included, regardless of the setting of CACHE_CONTROL. Locally overrides CACHE_EXPIRES.
id_field	Field used to uniquely identify resource items within the database. Locally overrides ID_FIELD.
item_lookup	True if item endpoint should be available, False otherwise. Locally overrides ITEM_LOOKUP.
item_lookup_field	Field used when looking up a resource item. Locally overrides ITEM_LOOKUP_FIELD.
item_url	Rule used to construct item endpoint URL. Locally overrides ITEM_URL.
resource_title	Title used when building resource links (HATEOAS). Defaults to resource's url.
item_title	Title to be used when building item references, both in XML and JSON responses. Overrides ITEM_TITLE.
additional_lookup	Besides the standard item endpoint which defaults to /<resource>/<ID_FIELD_value>, you can optionally define a secondary, read-only, endpoint like /<resource>/<person_name>. You do so by defining a dictionary comprised of two items *field* and *url*. The former is the name of the field used for the lookup. If the field type (as defined in the resource *schema*) is a string, then you put a URL rule in *url*. If it is an integer, then you just omit *url*, as it is automatically handled. See the code snippet below for an usage example of this feature.

continues on next page

Table 2 – continued from previous page

datasource	Explicitly links API resources to database collections. See *Advanced Datasource Patterns*.
auth_field	Enables *User-Restricted Resource Access*. When the feature is enabled, users can only read/update/delete resource items created by themselves. The keyword contains the actual name of the field used to store the id of the user who created the resource item. Locally overrides AUTH_FIELD.
allow_unknown	When True, this option will allow insertion of arbitrary, unknown fields to the endpoint. Use with caution. Locally overrides ALLOW_UNKNOWN. See *Allowing the Unknown* for more information. Defaults to False.
projection	When True, this option enables the *Projections* feature. Locally overrides PROJECTION. Defaults to True.
embedding	When True this option enables the *Embedded Resource Serialization* feature. Defaults to True.
extra_response_fields	Allows to configure a list of additional document fields that should be provided with every POST response. Normally only automatically handled fields (ID_FIELD, LAST_UPDATED, DATE_CREATED, ETAG) are included in response payloads. Overrides EXTRA_RESPONSE_FIELDS.
hateoas	When False, this option disables *HATEOAS* for the resource. Defaults to True.
mongo_query_whitelist	A list of extra Mongo query operators to allow for this endpoint besides the official list of allowed operators. Defaults to [].
mongo_write_concern	A dictionary defining MongoDB write concern settings for the endpoint datasource. All standard write concern settings (w, wtimeout, j, fsync) are supported. Defaults to {'w': 1} which means 'do regular acknowledged writes' (this is also the Mongo default.) Please be aware that setting 'w' to a value of 2 or greater requires replication to be active or you will be getting 500 errors (the write will still happen; Mongo will just be unable to check that it's being written to multiple servers.)
mongo_prefix	Allows overriding of the default MONGO prefix, which is used when retrieving MongoDB settings from configuration. For example if mongo_prefix is set to MONGO2 then, when serving requests for the endpoint, MONGO2 prefixed settings will be used to access the database. This allows for eventually serving data from a different database/server at every endpoint. See also: *Auth-driven Database Access*.

continues on next page

Table 2 – continued from previous page

mongo_indexes	Allows to specify a set of indexes to be created for this resource before the app is launched. Indexes are expressed as a dict where keys are index names and values are either a list of tuples of (field, direction) pairs, or a tuple with a list of field/direction pairs *and* index options expressed as a dict, such as `{'index name': [('field', 1)], 'index with args': ([('field', 1)], {"sparse": True})}`. Multiple pairs are used to create compound indexes. Direction takes all kind of values supported by PyMongo, such as `ASCENDING` = 1 and `DESCENDING` = -1. All index options such as `sparse`, `min`, `max`, etc. are supported (see PyMongo documentation.) *Please note:* keep in mind that index design, creation and maintenance is a very important task and should be planned and executed with great care. Usually it is also a very resource intensive operation. You might therefore want to handle this task manually, out of the context of API instantiation. Also remember that, by default, any already existent index for which the definition has been changed, will be dropped and re-created.
authentication	A class with the authorization logic for the endpoint. If not provided the eventual general purpose auth class (passed as application constructor argument) will be used. For details on authentication and authorization see *Authentication and Authorization*. Defaults to `None`,
embedded_fields	A list of fields for which *Embedded Resource Serialization* is enabled by default. For this feature to work properly fields in the list must be `embeddable`, and `embedding` must be active for the resource.
query_objectid_as_string	When enabled the Mongo parser will avoid automatically casting electable strings to ObjectIds. This can be useful in those rare occurrences where you have string fields in the database whose values can actually be casted to ObjectId values, but shouldn't. It effects queries (`?where=`) and parsing of payloads. Defaults to `False`.
internal_resource	When `True`, this option makes the resource internal. No HTTP action can be performed on the endpoint, which is still accessible from the Eve data layer. See *Internal Resources* for more information. Defaults to `False`.
etag_ignore_fields	List of fields that should not be used to compute the ETag value. Defaults to `None` which means that by default all fields are included in the computation. It looks like `['field1', 'field2', 'field3.nested_field', ...]`.
schema	A dict defining the actual data structure being handled by the resource. Enables data validation. See *Schema Definition*.
bulk_enabled	When `True` this option enables the *Bulk Inserts* feature for this resource. Locally overrides `BULK_ENABLED`.
soft_delete	When `True` this option enables the *Soft Delete* feature for this resource. Locally overrides `SOFT_DELETE`.

continues on next page

Table 2 – continued from previous page

merge_nested_documents	If True, updates to nested fields are merged with the current data on PATCH. If False, the updates overwrite the current data. Locally overrides MERGE_NESTED_DOCUMENTS.
normalize_dotted_fields	If True, dotted fields are parsed and processed as subdocument fields. If False, dotted fields are left unparsed and unprocessed, and the payload is passed to the underlying data-layer as-is. Please note that with the default Mongo layer, setting this to False will result in an error. Defaults to True.
normalize_on_patch	If True, the patch document will be normalized according to schema. This means if a field is not included in the patch body, it will be reset to the default value in its schema. If False, the field which is not included in the patch body will be kept untouched. Defaults to True.

Here's an example of resource customization, mostly done by overriding global API settings:

```
people = {
    # 'title' tag used in item links. Defaults to the resource title minus
    # the final, plural 's' (works fine in most cases but not for 'people')
    'item_title': 'person',

    # by default, the standard item entry point is defined as
    # '/people/<ObjectId>/'. We leave it untouched, and we also enable an
    # additional read-only entry point. This way consumers can also perform
    # GET requests at '/people/<lastname>'.
    'additional_lookup': {
        'url': 'regex("[\w]+")',
        'field': 'lastname'
    },

    # We choose to override global cache-control directives for this resource.
    'cache_control': 'max-age=10,must-revalidate',
    'cache_expires': 10,

    # we only allow GET and POST at this resource endpoint.
    'resource_methods': ['GET', 'POST'],
}
```

2.6.6 Schema Definition

Unless your API is read-only, you probably want to define resource *schemas*. Schemas are important because they enable proper validation for incoming streams.

```
# 'people' schema definition
schema = {
    'firstname': {
        'type': 'string',
        'minlength': 1,
        'maxlength': 10,
    },
    'lastname': {
```

(continues on next page)

(continued from previous page)

```
        'type': 'string',
        'minlength': 1,
        'maxlength': 15,
        'required': True,
        'unique': True,
    },
    # 'role' is a list, and can only contain values from 'allowed'.
    'role': {
        'type': 'list',
        'allowed': ["author", "contributor", "copy"],
    },
    # An embedded 'strongly-typed' dictionary.
    'location': {
        'type': 'dict',
        'schema': {
            'address': {'type': 'string'},
            'city': {'type': 'string'}
        },
    },
    'born': {
        'type': 'datetime',
    },
}
```

As you can see, schema keys are the actual field names, while values are dicts defining the field validation rules. Allowed validation rules are:

`type`	Field data type. Can be one of the following: • `string` • `boolean` • `integer` • `float` • `number` (integer and float values allowed) • `datetime` • `dict` • `list` • `media` If the MongoDB data layer is used then `objectid`, `dbref` and geographic data structures are also allowed: • `objectid` • `dbref` • `point` • `multipoint` • `linestring` • `multilinestring` • `polygon` • `multipolygon` • `geometrycollection` • `decimal` See *GeoJSON* for more information geo fields.
`required`	If `True`, the field is mandatory on insertion.
`readonly`	If `True`, the field is readonly.
`minlength`, `maxlength`	Minimum and maximum length allowed for `string` and `list` types.
`min`, `max`	Minimum and maximum values allowed for `integer`, `float` and `number` types.
`allowed`	List of allowed values for `string` and `list` types.
`empty`	Only applies to string fields. If `False`, validation will fail if the value is empty. Defaults to `True`.
`items`	Defines a list of values allowed in a `list` of fixed length, see docs.
`schema`	Validation schema for `dict` types and arbitrary length `list` types. For details and usage examples, see Cerberus documentation.
`unique`	The value of the field must be unique within the collection. Please note: validation constraints are checked against the database, and not between the payload documents themselves. This causes an interesting corner case: in the event of a multiple documents payload where two or more documents carry the same value for a field where the 'unique' constraint is set, the payload will validate successfully, as there are no duplicates in the database (yet). If this is an issue, the client can always send the documents one at a time for insertion, or validate locally before submitting the payload to the API.
`unique_to_user`	The field value is unique to the user. This is useful when *User-Restricted Resource Access* is enabled on an endpoint. The rule will be validated against *user data only*. So in this scenario duplicates are allowed as long as they are stored by different users. Conversely, a single user cannot store duplicate values. ~~If URRA is not active on the endpoint, this rule behaves like~~ `unique`
`unique_within_resource`	The value of the field must be unique within the resource. This differs from the `unique` rule in that it will use the datasource filter when searching for documents with the same

Schema syntax is based on Cerberus and yes, it can be extended. In fact, Eve itself extends the original grammar by adding the `unique` and `data_relation` keywords, along with the `objectid` datatype. For more information on custom validation and usage examples see *Data Validation*.

In *Resource / Item Endpoints* you customized the *people* endpoint. Then, in this section, you defined *people* validation rules. Now you are ready to update the domain which was originally set up in *Domain Configuration*:

```
# add the schema to the 'people' resource definition
people['schema'] = schema
# update the domain
DOMAIN['people'] = people
```

2.6.7 Advanced Datasource Patterns

The `datasource` keyword allows to explicitly link API resources to database collections. If omitted, the domain resource key is assumed to also be the name of the database collection. It is a dictionary with four allowed keys:

`source`	Name of the database collection consumed by the resource. If omitted, the resource name is assumed to also be a valid collection name. See *Multiple API Endpoints, One Datasource*.
`filter`	Database query used to retrieve and validate data. If omitted, by default the whole collection is retrieved. See *Predefined Database Filters*.
`projection`	Fieldset exposed by the endpoint. If omitted, by default all fields will be returned to the client. See *Limiting the Fieldset Exposed by the API Endpoint*.
`default_sort`	Default sorting for documents retrieved at the endpoint. If omitted, documents will be returned with the default database order. A valid statement would be: `'datasource': {'default_sort': [('name', 1)]}` For more information on sort and filters see *Filtering*.
`aggregation`	Aggregation pipeline and options. When used all other `datasource` settings are ignored, except `source`. The endpoint will be read-only and no item lookup will be available. Defaults to `None`. This is a dictionary with one or more of the following keys: • `pipeline`. The aggregation pipeline. Syntax must match the one supported by PyMongo. For more information see PyMongo Aggregation Examples and the official MongoDB Aggregation Framework documentation. • `options`. Aggregation options. Must be a dictionary with one or more of these keys: – `allowDiskUse` (bool) – `maxTimeMS` (int) – `batchSize` (int) – `useCursor` (bool) You only need to set `options` if you want to change any of PyMongo aggregation defaults.

Predefined Database Filters

Database filters for the API endpoint are set with the `filter` keyword.

```
people = {
    'datasource': {
        'filter': {'username': {'$exists': True}}
        }
    }
```

In the example above, the API endpoint for the *people* resource will only expose and update documents with an existing *username* field.

Predefined filters run on top of user queries (GET requests with *where* clauses) and standard conditional requests (*If-Modified-Since*, etc.)

Please note that datasource filters are applied on GET, PATCH and DELETE requests. If your resource allows POST requests (document insertions), then you will probably want to set the validation rules accordingly (in our example, 'username' should probably be a required field).

Static vs Dynamic filters

Predefined filters are static. You can also exploit the *Event Hooks* system (specifically, `on_pre_<METHOD>` hooks) to set up dynamic filters instead.

Multiple API Endpoints, One Datasource

Multiple API endpoints can target the same database collection. For example you can set both `/admins` and `/users` to read and write from the same *people* collection on the database.

```
people = {
    'datasource': {
        'source': 'people',
        'filter': {'userlevel': 1}
        }
    }
```

The above setting will retrieve, edit and delete only documents from the *people* collection with a *userlevel* of 1.

Limiting the Fieldset Exposed by the API Endpoint

By default API responses to GET requests will include all fields defined by the corresponding resource *schema*. The `projection` setting of the *datasource* resource keyword allows you to redefine the fieldset.

When you want to hide some *secret fields* from client, you should use inclusive projection setting and include all fields should be exposed. While, when you want to limit default responses to certain fields but still allow them to be accessible through client-side projections, you should use exclusive projection setting and exclude fields should be omitted.

The following is an example for inclusive projection setting:

```
people = {
    'datasource': {
        'projection': {'username': 1}
```

(continues on next page)

(continued from previous page)

```
        }
    }
```

The above setting will expose only the *username* field to GET requests, no matter the *schema* defined for the resource. And other fields **will not** be exposed even by client-side projection. The following API call will not return *lastname* or *born*.

```
$ curl -i http://myapi/people?projection={"lastname": 1, "born": 1}
HTTP/1.1 200 OK
```

You can also exclude fields from API responses. But this time, the excluded fields **will be** exposed to client-side projection. The following is an example for exclusive projection setting:

```
people = {
    'datasource': {
        'projection': {'username': 0}
        }
    }
```

The above will include all document fields but *username*. However, the following API call will return *username* this time. Thus, you can exploit this behaviour to serve media fields or other expensive fields.

In most cases, none or inclusive projection setting is preferred. With inclusive projection, secret fields are taken care from server side, and default fields returned can be defined by short-cut functions from client-side.

```
$ curl -i http://myapi/people?projection={"username": 1}
HTTP/1.1 200 OK
```

Please note that POST and PATCH methods will still allow the whole schema to be manipulated. This feature can come in handy when, for example, you want to protect insertion and modification behind an *Authentication and Authorization* scheme while leaving read access open to the public.

See also

- *Projections*
- *Leveraging Projections to optimize the handling of media files*

2.7 Data Validation

Data validation is provided out-of-the-box. Your configuration includes a schema definition for every resource managed by the API. Data sent to the API to be inserted/updated will be validated against the schema, and a resource will only be updated if validation passes.

```
$ curl -d '[{"firstname": "bill", "lastname": "clinton"}, {"firstname": "mitt", "lastname
↪": "romney"}]' -H 'Content-Type: application/json' http://myapi/people
HTTP/1.1 201 OK
```

The response will contain a success/error state for each item provided in the request:

```
{
    "_status": "ERR",
    "_error": "Some documents contains errors",
    "_items": [
        {
            "_status": "ERR",
            "_issues": {"lastname": "value 'clinton' not unique"}
        },
        {
            "_status": "OK",
        }
    ]
]
```

In the example above, the first document did not validate so the whole request has been rejected.

When all documents pass validation and are inserted correctly the response status is `201 Created`. If any document fails validation the response status is `422 Unprocessable Entity`, or any other error code defined by `VALIDATION_ERROR_STATUS` configuration.

For information on how to define documents schema and standard validation rules, see *Schema Definition*.

2.7.1 Extending Data Validation

Data validation is based on the Cerberus validation system and it is therefore extensible. As a matter of fact, Eve's MongoDB data-layer itself extends Cerberus validation, implementing the `unique` and `data_relation` constraints, the `ObjectId` data type and the `decimal128` on top of the standard rules.

2.7.2 Custom Validation Rules

Suppose that in your specific and very peculiar use case, a certain value can only be expressed as an odd integer. You decide to add support for a new `isodd` rule to our validation schema. This is how you would implement that:

```
from eve.io.mongo import Validator

class MyValidator(Validator):
    def _validate_isodd(self, isodd, field, value):
        if isodd and not bool(value & 1):
            self._error(field, "Value must be an odd number")

app = Eve(validator=MyValidator)

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

By subclassing the base Mongo validator class and then adding a custom `_validate_<rulename>` method, you extended the available *Schema Definition* grammar and now the new custom rule `isodd` is available in your schema. You can now do something like:

```
'schema': {
    'oddity': {
        'isodd': True,
```

(continues on next page)

(continued from previous page)

```
        'type': 'integer'
    }
}
```

Cerberus and Eve also offer function-based validation and type coercion, lightweight alternatives to class-based custom validation.

2.7.3 Custom Data Types

You can also add new data types by simply adding `_validate_type_<typename>` methods to your subclass. Consider the following snippet from the Eve source code.

```
def _validate_type_objectid(self, value):
    """ Enables validation for `objectid` schema attribute.

    :param value: field value.
    """
    if isinstance(value, ObjectId):
        return True
```

This method enables support for MongoDB `ObjectId` type in your schema, allowing something like this:

```
'schema': {
    'owner': {
        'type': 'objectid',
        'required': True,
    },
}
```

You can also check the source code for Eve custom validation, where you will find more advanced use cases, such as the implementation of the `unique` and `data_relation` constraints.

For more information on

Note: We have only scratched the surface of data validation. Please make sure to check the Cerberus documentation for a complete list of available validation rules and data types.

Also note that Cerberus requirement is pinned to version 0.9.2, which still supports the `validate_update` method used for `PATCH` requests. Upgrade to Cerberus 1.0+ is scheduled for Eve version 0.8.

2.7.4 Allowing the Unknown

Normally you don't want clients to inject unknown fields in your documents. However, there might be circumstances where this is desirable. During the development cycle, for example, or when you are dealing with very heterogeneous data. After all, not forcing normalized information is one of the selling points of MongoDB and many other NoSQL data stores.

In Eve, you achieve this by setting the `ALLOW_UNKNOWN` option to `True`. Once this option is enabled, fields matching the schema will be validated normally, while unknown fields will be quietly stored without a glitch. You can also enable this feature only for certain endpoints by setting the `allow_unknown` local option.

Consider the following domain:

```
DOMAIN: {
    'people': {
        'allow_unknown': True,
        'schema': {
            'firstname': {'type': 'string'},
            }
        }
    }
```

Normally you can only add (POST) or edit (PATCH) *firstnames* to the `/people` endpoint. However, since `allow_unknown` has been enabled, even a payload like this will be accepted:

```
$ curl -d '[{"firstname": "bill", "lastname": "clinton"}, {"firstname": "bill", "age":70}
↪]' -H 'Content-Type: application/json' http://myapi/people
HTTP/1.1 201 OK
```

Please note

Use this feature with extreme caution. Also be aware that, when this option is enabled, clients will be capable of actually *adding* fields via PATCH (edit).

`ALLOW_UNKNOWN` is also useful for read-only APIs or endpoints that need to return the whole document, as found in the underlying database. In this scenario you don't want to bother with validation schemas. For the whole API just set `ALLOW_UNKNOWN` to `True`, then `schema: {}` at every endpoint. For a single endpoint, use `allow_unknown: True` instead.

2.7.5 Schema validation

By default, schemas are validated to ensure they conform to the structure documented in *Schema Definition*.

In order to deal with non-conforming schemas, add *Custom Validation Rules* for non-conforming keys used in the schema.

2.8 Authentication and Authorization

2.8.1 Introduction to Security

Authentication is the mechanism whereby systems may securely identify their users. Eve supports several authentication schemes: Basic Authentication, Token Authentication, HMAC Authentication. *OAuth2 integration* is easily accomplished.

Authorization is the mechanism by which a system determines what level of access a particular (authenticated) user should have access to resources controlled by the system. In Eve, you can restrict access to all API endpoints, or only some of them. You can protect some HTTP verbs while leaving others open. For example, you can allow public read-only access while leaving item creation and edition restricted to authorized users only. You can also allow `GET` access for certain requests and `POST` access for others by checking the method parameter. There is also support for role-based access control.

Security is one of those areas where customization is very important. This is why you are provided with a handful of base authentication classes. They implement the basic authentication mechanism and must be subclassed in order to

implement authorization logic. No matter which authentication scheme you pick the only thing that you need to do in your subclass is override the `check_auth()` method.

2.8.2 Global Authentication

To enable authentication for your API just pass the custom auth class on app instantiation. In our example we're going to use the `BasicAuth` base class, which implements the *Basic Authentication* scheme:

```
from eve.auth import BasicAuth

class MyBasicAuth(BasicAuth):
    def check_auth(self, username, password, allowed_roles, resource,
                   method):
        return username == 'admin' and password == 'secret'

app = Eve(auth=MyBasicAuth)
app.run()
```

All your API endpoints are now secured, which means that a client will need to provide the correct credentials in order to consume the API:

```
$ curl -i http://example.com
HTTP/1.1 401 UNAUTHORIZED
Please provide proper credentials.

$ curl -H "Authorization: Basic YWRtaW46c2VjcmV0" -i http://example.com
HTTP/1.1 200 OK
```

By default access is restricted to all endpoints for all HTTP verbs (methods), effectively locking down the whole API.

But what if your authentication logic is more complex, and you only want to secure some endpoints or apply different logics depending on the endpoint being consumed? You could get away with just adding logic to your authentication class, maybe with something like this:

```
class MyBasicAuth(BasicAuth):
    def check_auth(self, username, password, allowed_roles, resource, method):
        if resource in ('zipcodes', 'countries'):
            # 'zipcodes' and 'countries' are public
            return True
        else:
            # all the other resources are secured
            return username == 'admin' and password == 'secret'
```

If needed, this approach also allows to take the request `method` into consideration, for example to allow `GET` requests for everyone while forcing validation on edits (`POST`, `PUT`, `PATCH`, `DELETE`).

2.8.3 Endpoint-level Authentication

The *one class to bind them all* approach seen above is probably good for most use cases but as soon as authorization logic gets more complicated it could easily lead to complex and unmanageable code, something you don't really want to have when dealing with security.

Wouldn't it be nice if we could have specialized auth classes that we could freely apply to selected endpoints? This way the global level auth class, the one passed to the Eve constructor as seen above, would still be active on all endpoints except those where different authorization logic is needed. Alternatively, we could even choose to *not* provide a global auth class, effectively making all endpoints public, except the ones we want protected. With a system like this we could even choose to have some endpoints protected with, say, Basic Authentication while others are secured with Token, or HMAC Authentication!

Well, turns out this is actually possible by simply enabling the resource-level `authentication` setting when we are defining the API *domain*.

```
DOMAIN = {
    'people': {
        'authentication': MySuperCoolAuth,
        ...
        },
    'invoices': ...
    }
```

And that's it. The *people* endpoint will now be using the `MySuperCoolAuth` class for authentication, while the `invoices` endpoint will be using the general-purpose auth class if provided or else it will just be open to the public.

There are other features and options that you can use to reduce complexity in your auth classes, especially (but not only) when using the global level authentication system. Lets review them.

2.8.4 Global Endpoint Security

You might want a public read-only API where only authorized users can write, edit and delete. You can achieve that by using the `PUBLIC_METHODS` and `PUBLIC_ITEM_METHODS` *global settings*. Add the following to your *settings.py*:

```
PUBLIC_METHODS = ['GET']
PUBLIC_ITEM_METHODS = ['GET']
```

And run your API. POST, PATCH and DELETE are still restricted, while GET is publicly available at all API endpoints. `PUBLIC_METHODS` refers to resource endpoints, like `/people`, while `PUBLIC_ITEM_METHODS` refers to individual items like `/people/id`.

2.8.5 Custom Endpoint Security

Suppose that you want to allow public read access to only certain resources. You do that by declaring public methods at resource level, while declaring the API *domain*:

```
DOMAIN = {
    'people': {
        'public_methods': ['GET'],
        'public_item_methods': ['GET'],
        },
    }
```

Be aware that, when present, *resource settings* override global settings. You can use this to your advantage. Suppose that you want to grant read access to all endpoints with the only exception of `/invoices`. You first open read access for all endpoints:

```
PUBLIC_METHODS = ['GET']
PUBLIC_ITEM_METHODS = ['GET']
```

Then you protect the private endpoint:

```
DOMAIN = {
    'invoices': {
        'public_methods': [],
        'public_item_methods': [],
        }
    }
```

Effectively making *invoices* a restricted resource.

2.8.6 Basic Authentication

The `eve.auth.BasicAuth` class allows the implementation of Basic Authentication (RFC2617). It should be subclassed in order to implement custom authentication.

Basic Authentication with bcrypt

Encoding passwords with bcrypt is a great idea. It comes at the cost of performance, but that's precisely the point, as slow encoding means very good resistance to brute-force attacks. For a faster (and less safe) alternative, see the SHA1/MAC snippet further below.

This script assumes that user accounts are stored in an *accounts* MongoDB collection, and that passwords are stored as bcrypt hashes. All API resources/methods will be secured unless they are made explicitly public.

Please note

You will need to install *py-bcrypt* for this to work.

```
# -*- coding: utf-8 -*-

"""
    Auth-BCrypt
    ~~~~~~~~~~~

    Securing an Eve-powered API with Basic Authentication (RFC2617).

    You will need to install py-bcrypt: ``pip install py-bcrypt``

    This snippet by Nicola Iarocci can be used freely for anything you like.
    Consider it public domain.
"""

import bcrypt
from eve import Eve
```

(continues on next page)

(continued from previous page)

```
from eve.auth import BasicAuth
from flask import current_app as app

class BCryptAuth(BasicAuth):
    def check_auth(self, username, password, allowed_roles, resource, method):
        # use Eve's own db driver; no additional connections/resources are used
        accounts = app.data.driver.db['accounts']
        account = accounts.find_one({'username': username})
        return account and \
            bcrypt.hashpw(password, account['password']) == account['password']


if __name__ == '__main__':
    app = Eve(auth=BCryptAuth)
    app.run()
```

Basic Authentication with SHA1/HMAC

This script assumes that user accounts are stored in an *accounts* MongoDB collection, and that passwords are stored as SHA1/HMAC hashes. All API resources/methods will be secured unless they are made explicitly public.

```
# -*- coding: utf-8 -*-

"""
    Auth-SHA1/HMAC
    ~~~~~~~~~~~~~~

    Securing an Eve-powered API with Basic Authentication (RFC2617).

    Since we are using werkzeug we don't need any extra import (werkzeug being
    one of Flask/Eve prerequisites).

    This snippet by Nicola Iarocci can be used freely for anything you like.
    Consider it public domain.
"""

from eve import Eve
from eve.auth import BasicAuth
from werkzeug.security import check_password_hash
from flask import current_app as app

class Sha1Auth(BasicAuth):
    def check_auth(self, username, password, allowed_roles, resource, method):
        # use Eve's own db driver; no additional connections/resources are used
        accounts = app.data.driver.db['accounts']
        account = accounts.find_one({'username': username})
        return account and \
            check_password_hash(account['password'], password)


if __name__ == '__main__':
```

(continues on next page)

(continued from previous page)

```
    app = Eve(auth=Sha1Auth)
    app.run()
```

2.8.7 Token-Based Authentication

Token-based authentication can be considered a specialized version of Basic Authentication. The Authorization header tag will contain the auth token as the username, and no password.

This script assumes that user accounts are stored in an *accounts* MongoDB collection. All API resources/methods will be secured unless they are made explicitly public (by fiddling with some settings you can open one or more resources and/or methods to public access -see docs).

```
# -*- coding: utf-8 -*-

"""
    Auth-Token
    ~~~~~~~~~~

    Securing an Eve-powered API with Token based Authentication.

    This snippet by Nicola Iarocci can be used freely for anything you like.
    Consider it public domain.
"""

from eve import Eve
from eve.auth import TokenAuth
from flask import current_app as app

class TokenAuth(TokenAuth):
    def check_auth(self, token, allowed_roles, resource, method):
        """For the purpose of this example the implementation is as simple as
        possible. A 'real' token should probably contain a hash of the
        username/password combo, which should then be validated against the account
        data stored on the DB.
        """
        # use Eve's own db driver; no additional connections/resources are used
        accounts = app.data.driver.db['accounts']
        return accounts.find_one({'token': token})


if __name__ == '__main__':
    app = Eve(auth=TokenAuth)
    app.run()
```

2.8.8 HMAC Authentication

The `eve.auth.HMACAuth` class allows for custom, Amazon S3-like, HMAC (Hash Message Authentication Code) authentication, which is basically a very secure custom authentication scheme built around the *Authorization* header.

How HMAC Authentication Works

The server provides the client with a user id and a secret key through some out-of-band technique (e.g., the service sends the client an e-mail containing the user id and secret key). The client will use the supplied secret key to sign all requests.

When the client wants to send a request, he builds the complete request and then, using the secret key, computes a hash over the complete message body (and optionally some of the message headers if required)

Next, the client adds the computed hash and his userid to the message in the Authorization header:

```
Authorization: johndoe:uCMfSzkjue+HSDygYB5aEg==
```

and sends it to the service. The service retrieves the userid from the message header and searches the private key for that user in its own database. Next it computes the hash over the message body (and selected headers) using the key to generate its hash. If the hash the client sends matches the hash the server computes, then the server knows the message was sent by the real client and was not altered in any way.

Really the only tricky part is sharing a secret key with the user and keeping that secure. That is why some services allow for generation of shared keys with a limited life time so you can give the key to a third party to temporarily work on your behalf. This is also the reason why the secret key is generally provided through out-of-band channels (often a webpage or, as said above, an email or plain old paper).

The `eve.auth.HMACAuth` class also support access roles.

HMAC Example

The snippet below can also be found in the *examples/security* folder of the Eve repository.

```
from eve import Eve
from eve.auth import HMACAuth
from flask import current_app as app
from hashlib import sha1
import hmac


class HMACAuth(HMACAuth):
    def check_auth(self, userid, hmac_hash, headers, data, allowed_roles,
                   resource, method):
        # use Eve's own db driver; no additional connections/resources are
        # used
        accounts = app.data.driver.db['accounts']
        user = accounts.find_one({'userid': userid})
        if user:
            secret_key = user['secret_key']
        # in this implementation we only hash request data, ignoring the
        # headers.
        return user and \
            hmac.new(str(secret_key), str(data), sha1).hexdigest() == \
```

(continues on next page)

(continued from previous page)

```
                hmac_hash


if __name__ == '__main__':
    app = Eve(auth=HMACAuth)
    app.run()
```

2.8.9 Role Based Access Control

The code snippets above deliberately ignore the `allowed_roles` parameter. You can use this parameter to restrict access to authenticated users who also have been assigned specific roles.

First, you would use the new `ALLOWED_ROLES` and `ALLOWED_ITEM_ROLES` *global settings* (or the corresponding `allowed_roles` and `allowed_item_roles` *resource settings*).

```
ALLOWED_ROLES = ['admin']
```

Then your subclass would implement the authorization logic by making good use of the aforementioned `allowed_roles` parameter.

The snippet below assumes that user accounts are stored in an *accounts* MongoDB collection, that passwords are stored as SHA1/HMAC hashes and that user roles are stored in a 'roles' array. All API resources/methods will be secured unless they are made explicitly public.

```
# -*- coding: utf-8 -*-

"""
    Auth-SHA1/HMAC-Roles
    ~~~~~~~~~~~~~~~~~~~~

    Securing an Eve-powered API with Basic Authentication (RFC2617) and user
    roles.

    Since we are using werkzeug we don't need any extra import (werkzeug being
    one of Flask/Eve prerequisites).

    This snippet by Nicola Iarocci can be used freely for anything you like.
    Consider it public domain.
"""

from eve import Eve
from eve.auth import BasicAuth
from werkzeug.security import check_password_hash
from flask import current_app as app

class RolesAuth(BasicAuth):
    def check_auth(self, username, password, allowed_roles, resource, method):
        # use Eve's own db driver; no additional connections/resources are used
        accounts = app.data.driver.db['accounts']
        lookup = {'username': username}
        if allowed_roles:
            # only retrieve a user if his roles match ``allowed_roles``
```

(continues on next page)

(continued from previous page)

```
            lookup['roles'] = {'$in': allowed_roles}
        account = accounts.find_one(lookup)
        return account and check_password_hash(account['password'], password)


if __name__ == '__main__':
    app = Eve(auth=RolesAuth)
    app.run()
```

2.8.10 User-Restricted Resource Access

When this feature is enabled, each stored document is associated with the account that created it. This allows the API to transparently serve only account-created documents on all kinds of requests: read, edit, delete and of course create. User authentication needs to be enabled for this to work properly.

At the global level this feature is enabled by setting `AUTH_FIELD` and locally (at the endpoint level) by setting `auth_field`. These properties define the name of the field used to store the id of the user who created the document. So for example by setting `AUTH_FIELD` to `user_id`, you are effectively (and transparently to the user) adding a `user_id` field to every stored document. This will then be used to retrieve/edit/delete documents stored by the user.

But how do you set the `auth_field` value? By invoking the `set_request_auth_value()` class method. Let us revise our BCrypt-authentication example from above:

```
# -*- coding: utf-8 -*-

"""
    Auth-BCrypt
    ~~~~~~~~~~~

    Securing an Eve-powered API with Basic Authentication (RFC2617).

    You will need to install py-bcrypt: ``pip install py-bcrypt``

    This snippet by Nicola Iarocci can be used freely for anything you like.
    Consider it public domain.
"""

import bcrypt
from eve import Eve
from eve.auth import BasicAuth


class BCryptAuth(BasicAuth):
    def check_auth(self, username, password, allowed_roles, resource, method):
        # use Eve's own db driver; no additional connections/resources are used
        accounts = app.data.driver.db['accounts']
        account = accounts.find_one({'username': username})
        # set 'auth_field' value to the account's ObjectId
        # (instead of _id, you might want to use ID_FIELD)
        if account and '_id' in account:
            self.set_request_auth_value(account['_id'])
        return account and \
```

(continues on next page)

(continued from previous page)

```
            bcrypt.hashpw(password, account['password']) == account['password']


if __name__ == '__main__':
    app = Eve(auth=BCryptAuth)
    app.run()
```

2.8.11 Auth-driven Database Access

Custom authentication classes can also set the database that should be used when serving the active request.

Normally you either use a single database for the whole API or you configure which database each endpoint consumes by setting `mongo_prefix` to the desired value (see *Resource / Item Endpoints*).

However, you might opt to select the target database based on the active token, user or client. This is handy if your use-case includes user-dedicated database instances. All you have to do is set invoke the `set_mongo_prefix()` method when authenticating the request.

A trivial example would be:

```
from eve.auth import BasicAuth

class MyBasicAuth(BasicAuth):
    def check_auth(self, username, password, allowed_roles, resource, method):
        if username == 'user1':
            self.set_mongo_prefix('MONGO1')
        elif username == 'user2':
            self.set_mongo_prefix('MONGO2')
        else:
            # serve all other users from the default db.
            self.set_mongo_prefix(None)
        return username is not None and password == 'secret'

app = Eve(auth=MyBasicAuth)
app.run()
```

The above class will serve `user1` with data coming from the database which configuration settings are prefixed by `MONGO1` in `settings.py`. Same happens with `user2` and `MONGO2` while all other users are served with the default database.

Since values set by `set_mongo_prefix()` have precedence over both default and endpoint-level `mongo_prefix` settings, what happens here is that the two users will always be served from their reserved databases, no matter the eventual database configuration for the endpoint.

2.8.12 OAuth2 Integration

Since you have total control over the Authorization process, integrating OAuth2 with Eve is easy. Make yourself comfortable with the topics illustrated in this page, then head over to Eve-OAuth2, an example project which leverages Flask-Sentinel to demonstrate how you can protect your API with OAuth2.

Please note

The snippets in this page can also be found in the *examples/security* folder of the Eve repository.

2.9 Funding

We believe that collaboratively funded software can offer outstanding returns on investment, by encouraging users to collectively share the cost of development.

The Eve REST framework continues to be open-source and permissively licensed, but we firmly believe it is in the commercial best-interest for users of the project to invest in its ongoing development.

Signing up as a Backer or Sponsor will:

- Directly contribute to faster releases, more features, and higher quality software.
- Allow more time to be invested in documentation, issue triage, and community support.
- Safeguard the future development of the Eve REST framework.

If you run a business and is using Eve in a revenue-generating product, it would make business sense to sponsor Eve development: it ensures the project that your product relies on stays healthy and actively maintained. It can also help your exposure in the Eve community and makes it easier to attract Eve developers.

Of course, individual users are also welcome to make a recurring pledge if Eve has helped you in your work or personal projects. Alternatively, consider donating as a sign of appreciation - like buying me coffee once in a while :)

2.9.1 Support Eve development

You can support Eve development by pledging on GitHub, Patreon, or PayPal.

- Become a Backer on GitHub
- Become a Backer on Patreon
- Donate via PayPal (one time)

2.9.2 Eve Course at TalkPython Training

There is a 5 hours-long Eve course available for you at the fine TalkPython Training website. The teacher is Nicola, Eve author and maintainer. Taking this course will directly support the project.

- Take the Eve Course at TalkPython Training

2.9.3 Custom Sponsorship and Consulting

If you are a business that is building core products using Eve, I am also open to conversations regarding custom sponsorship / consulting arrangements. Just get in touch with me.

Backers

Backers who actively support Eve and Cerberus development:

- Gabriel Wainer
- Jon Kelled

Generous Backers

Generous backers who actively support Eve and Cerberus development:

2.10 Tutorials

2.10.1 RESTful Account Management

Please note

This tutorial assumes that you've read the *Quickstart* and the *Authentication and Authorization* guides.

Except for the relatively rare occurrence of open (and generally read-only) public APIs, most services are only accessible to authenticated users. A common pattern is that users create their account on a website or with a mobile application. Once they have an account, they are allowed to consume one or more APIs. This is the model followed by most social networks and service providers (Twitter, Facebook, Netflix, etc.) So how do you, the service provider, manage to create, edit and delete accounts while using the same API that is being consumed by the accounts themselves?

In the following paragraphs we'll see a couple of possible Account Management implementations, both making intensive use of a host of Eve features such as *Custom Endpoint Security*, *Role Based Access Control*, *User-Restricted Resource Access*, *Event Hooks*.

We assume that SSL/TLS is enabled, which means that our transport layer is encrypted, making both *Basic Authentication* and *Token-Based Authentication* valid options to secure API endpoints.

Let's say we're upgrading the API we defined in the *Quickstart* tutorial.

Accounts with Basic Authentication

Our tasks are as follows:

1. Make an endpoint available for all account management activities (`/accounts`).
2. Secure the endpoint, so that it is only accessible to clients that we control: our own website, mobile apps with account management capabilities, etc.
3. Make sure that all other API endpoints are only accessible to authenticated accounts (created by means of the above mentioned endpoint).
4. Allow authenticated users to only access resources created by themselves.

1. The `/accounts` endpoint

The account management endpoint is no different than any other API endpoint. It is just a matter of declaring it in our settings file. Let's declare the resource schema first.

```
schema =  {
    'username': {
        'type': 'string',
        'required': True,
        'unique': True,
        },
    'password': {
        'type': 'string',
        'required': True,
    },
},
```

Then, let's define the endpoint.

```
accounts = {
    # the standard account entry point is defined as
    # '/accounts/<ObjectId>'. We define  an additional read-only entry
    # point accessible at '/accounts/<username>'.
    'additional_lookup': {
        'url': 'regex("[\w]+")',
        'field': 'username',
    },

    # We also disable endpoint caching as we don't want client apps to
    # cache account data.
    'cache_control': '',
    'cache_expires': 0,

    # Finally, let's add the schema definition for this endpoint.
    'schema': schema,
}
```

We defined an additional read-only entry point at `/accounts/<username>`. This isn't really a necessity, but it can come in handy to easily verify if a username has been taken already, or to retrieve an account without knowing its `ObjectId` beforehand. Of course, both pieces of information can also be found by querying the resource endpoint (`/accounts?where={"username": "johndoe"}`), but then we would need to parse the response payload, whereas

by hitting our new endpoint with a GET request we will obtain the bare account data, or a `404 Not Found` if the account does not exist.

Once the endpoint has been configured, we need to add it to the API domain:

```
DOMAIN['accounts'] = accounts
```

2. Securing the `/accounts/` endpoint

2a. Hard-coding our way in

Securing the endpoint can be achieved by allowing only well-known *superusers* to operate on it. Our authentication class, which is defined in the launch script, can be hard-coded to handle the case:

```
import bcrypt
from eve import Eve
from eve.auth import BasicAuth


class BCryptAuth(BasicAuth):
    def check_auth(self, username, password, allowed_roles, resource, method):
        if resource == 'accounts':
            return username == 'superuser' and password == 'password'
        else:
            # use Eve's own db driver; no additional connections/resources are used
            accounts = app.data.driver.db['accounts']
            account = accounts.find_one({'username': username})
            return account and \
                bcrypt.hashpw(password, account['password']) == account['password']


if __name__ == '__main__':
    app = Eve(auth=BCryptAuth)
    app.run()
```

Thus, only the `superuser` account will be allowed to consume the `accounts` endpoint, while standard authentication logic will apply to all other endpoints. Our mobile app (say) will add accounts by hitting the endpoint with simple POST requests, of course authenticating itself as a *superuser* by means of the *Authorization* header. The script assumes that stored passwords are encrypted with *bcrypt* (storing passwords as plain text is *never* a good idea). See *Basic Authentication* for an alternative, faster but less secure SHA1/MAC example.

2b. User Roles Access Control

Hard-coding usernames and passwords might very well do the job, but it is hardly the best approach that we can take here. What if another *superurser* account needs access to the endpoint? Updating the script each time a privileged user joins the ranks does not seem appropriate (it isn't). Fortunately, the *Role Based Access Control* feature can help us here. You see where we are going with this: the idea is that only accounts with *superuser* and *admin* roles will be granted access to the endpoint.

Let's start by updating our resource schema.

```
    schema =  {
        'username': {
            'type': 'string',
            'required': True,
            },
        'password': {
            'type': 'string',
            'required': True,
        },
        'roles': {
            'type': 'list',
            'allowed': ['user', 'superuser', 'admin'],
            'required': True,
        }
    },
```

We just added a new `roles` field which is a required list. From now on, one or more roles will have to be assigned on account creation.

Now we need to restrict endpoint access to *superuser* and *admin* accounts only so let's update the endpoint definition accordingly.

```
accounts = {
    # the standard account entry point is defined as
    # '/accounts/<ObjectId>'. We define  an additional read-only entry
    # point accessible at '/accounts/<username>'.
    'additional_lookup': {
        'url': 'regex("[\w]+")',
        'field': 'username',
    },

    # We also disable endpoint caching as we don't want client apps to
    # cache account data.
    'cache_control': '',
    'cache_expires': 0,

    # Only allow superusers and admins.
    'allowed_roles': ['superuser', 'admin'],

    # Finally, let's add the schema definition for this endpoint.
    'schema': schema,
}
```

Finally, a rewrite of our authentication class is in order.

```
from eve import Eve
from eve.auth import BasicAuth
from werkzeug.security import check_password_hash


class RolesAuth(BasicAuth):
    def check_auth(self, username, password, allowed_roles, resource, method):
        # use Eve's own db driver; no additional connections/resources are used
        accounts = app.data.driver.db['accounts']
```

(continues on next page)

(continued from previous page)

```
        lookup = {'username': username}
        if allowed_roles:
            # only retrieve a user if his roles match ``allowed_roles``
            lookup['roles'] = {'$in': allowed_roles}
        account = accounts.find_one(lookup)
        return account and check_password_hash(account['password'], password)


if __name__ == '__main__':
    app = Eve(auth=RolesAuth)
    app.run()
```

What the above snippet does is secure all API endpoints with role-base access control. It is, in fact, the same snippet seen in *Role Based Access Control*. This technique allows us to keep the code untouched as we add more *superuser* or *admin* accounts (and we'll probably be adding them by accessing our very own API). Also, should the need arise, we could easily restrict access to more endpoints just by updating the settings file, again without touching the authentication class.

3. Securing other API endpoints

This will be quick, as both the *hard-coding* and the *role-based* access control approaches above effectively secure all API endpoints already. Passing an authentication class to the `Eve` object enables authentication for the whole API: every time an endpoint is hit with a request, the class instance is invoked.

Of course, you can still fine-tune security, for example by allowing public access to certain endpoints, or to certain HTTP methods. See *Authentication and Authorization* for more details.

4. Only allowing access to account resources

Most of the time when you allow Authenticated users to store data, you only want them to access their own data. This can be conveniently achieved by using the *User-Restricted Resource Access* feature. When enabled, each stored document is associated with the account that created it. This allows the API to transparently serve only account-created documents on all kind of requests: read, edit, delete and of course create.

There are only two things that we need to do in order to activate this feature:

1. Configure the name of the field that will be used to store the owner of the document;
2. Set the document owner on each incoming POST request.

Since we want to enable this feature for all of our API endpoints we'll just update our `settings.py` file by setting a proper `AUTH_FIELD` value:

```
# Name of the field used to store the owner of each document
AUTH_FIELD = 'user_id'
```

Then, we want to update our authentication class to properly update the field's value:

```
from eve import Eve
from eve.auth import BasicAuth
from werkzeug.security import check_password_hash
```

(continues on next page)

(continued from previous page)

```
class RolesAuth(BasicAuth):
    def check_auth(self, username, password, allowed_roles, resource, method):
        # use Eve's own db driver; no additional connections/resources are used
        accounts = app.data.driver.db['accounts']
        lookup = {'username': username}
        if allowed_roles:
            # only retrieve a user if his roles match ``allowed_roles``
            lookup['roles'] = {'$in': allowed_roles}
        account = accounts.find_one(lookup)
        # set 'AUTH_FIELD' value to the account's ObjectId
        # (instead of _Id, you might want to use ID_FIELD)
        self.set_request_auth_value(account['_id'])
        return account and check_password_hash(account['password'], password)


if __name__ == '__main__':
    app = Eve(auth=RolesAuth)
    app.run()
```

This is all we need to do. Now when a client hits say the `/invoices` endpoint with a GET request, it will only be served with invoices created by its own account. The same will happen with DELETE and PATCH, making it impossible for an authenticated user to accidentally retrieve, edit or delete other people's data.

Accounts with Token Authentication

As seen in *Token-Based Authentication*, token authentication is just a specialized version of Basic Authentication. It is actually executed as a standard Basic Authentication request where the value of the *username* field is used for the token, and the password field is not provided (if included, it is ignored).

Consequently, handling accounts with Token Authentication is very similar to what we saw in *Accounts with Basic Authentication*, but there's one little caveat: tokens need to be generated and stored along with the account, and eventually returned to the client.

In light of this, let's review our updated task list:

1. Make an endpoint available for all account management activities (`/accounts`).
2. Secure the endpoint so that it is only accessible to clients (tokens) that we control.
3. On account creation, generate and store its token.
4. Optionally, return the new token with the response.
5. Make sure that all other API endpoints are only accessible to authenticated tokens.
6. Allow authenticated users to only access resources created by themselves

1. The /accounts/ endpoint

This isn't any different than what we did in *Accounts with Basic Authentication*. We just need to add the *token* field to our schema:

```
    schema =  {
        'username': {
            'type': 'string',
            'required': True,
            'unique': True,
            },
        'password': {
            'type': 'string',
            'required': True,
        },
        'roles': {
            'type': 'list',
            'allowed': ['user', 'superuser', 'admin'],
            'required': True,
        },
        'token': {
            'type': 'string',
            'required': True,
        }
    }
```

2. Securing the /accounts/ endpoint

We defined the *roles* field for the *accounts* schema in the previous step. We also need to define the endpoint, making sure that we set the allowed user roles.

```
accounts = {
    # the standard account entry point is defined as
    # '/accounts/<ObjectId>'. We define  an additional read-only entry
    # point accessible at '/accounts/<username>'.
    'additional_lookup': {
        'url': 'regex("[\w]+")',
        'field': 'username',
    },

    # We also disable endpoint caching as we don't want client apps to
    # cache account data.
    'cache_control': '',
    'cache_expires': 0,

    # Only allow superusers and admins.
    'allowed_roles': ['superuser', 'admin'],

    # Finally, let's add the schema definition for this endpoint.
    'schema': schema,
}
```

And finally, here is our launch script which is, of course, using a `TokenAuth` subclass this time around:

```
from eve import Eve
from eve.auth import TokenAuth


class RolesAuth(TokenAuth):
    def check_auth(self, token,  allowed_roles, resource, method):
        # use Eve's own db driver; no additional connections/resources are used
        accounts = app.data.driver.db['accounts']
        lookup = {'token': token}
        if allowed_roles:
            # only retrieve a user if his roles match ``allowed_roles``
            lookup['roles'] = {'$in': allowed_roles}
        account = accounts.find_one(lookup)
        return account


if __name__ == '__main__':
    app = Eve(auth=RolesAuth)
    app.run()
```

3. Building custom tokens on account creation

The code above has a problem: it won't authenticate anybody, as we aren't generating any token yet. Consequently, clients aren't getting their auth tokens back so they don't really know how to authenticate. Let's fix that by using the awesome *Event Hooks* feature. We'll update our launch script by registering a callback function that will be called when a new account is about to be stored to the database.

```
from eve import Eve
from eve.auth import TokenAuth
import random
import string


class RolesAuth(TokenAuth):
    def check_auth(self, token,  allowed_roles, resource, method):
        # use Eve's own db driver; no additional connections/resources are used
        accounts = app.data.driver.db['accounts']
        lookup = {'token': token}
        if allowed_roles:
            # only retrieve a user if his roles match ``allowed_roles``
            lookup['roles'] = {'$in': allowed_roles}
        account = accounts.find_one(lookup)
        return account


def add_token(documents):
    # Don't use this in production:
    # You should at least make sure that the token is unique.
    for document in documents:
        document["token"] = (''.join(random.choice(string.ascii_uppercase)
                                     for x in range(10)))
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    app = Eve(auth=RolesAuth)
    app.on_insert_accounts += add_token
    app.run()
```

As you can see, we are subscribing to the `on_insert` event of the *accounts* endpoint with our `add_token` function. This callback will receive *documents* as an argument, which is a list of validated documents accepted for database insertion. We simply add (or replace in the unlikely case that the request contained it already) a token to every document, and we're done! For more information on callbacks, see Event Hooks.

4. Returning the token with the response

Optionally, you might want to return the tokens with the response. Truth be told, this isn't a very good idea. You generally want to send access information out-of-band, with an email for example. However we're assuming that we are on SSL, and there are cases where sending the auth token just makes sense, like when the client is a mobile application and we want the user to use the service right away.

Normally, only automatically handled fields (`ID_FIELD`, `LAST_UPDATED`, `DATE_CREATED`, `ETAG`) are included with POST response payloads. Fortunately, there's a setting which allows us to inject additional fields in responses, and that is `EXTRA_RESPONSE_FIELDS`, with its endpoint-level equivalent, `extra_response_fields`. All we need to do is update our endpoint definition accordingly:

```
accounts = {
    # the standard account entry point is defined as
    # '/accounts/<ObjectId>'. We define  an additional read-only entry
    # point accessible at '/accounts/<username>'.
    'additional_lookup': {
        'url': 'regex("[\w]+")',
        'field': 'username',
    },

    # We also disable endpoint caching as we don't want client apps to
    # cache account data.
    'cache_control': '',
    'cache_expires': 0,

    # Only allow superusers and admins.
    'allowed_roles': ['superuser', 'admin'],

    # Allow 'token' to be returned with POST responses
    'extra_response_fields': ['token'],

    # Finally, let's add the schema definition for this endpoint.
    'schema': schema,
}
```

From now on responses to POST requests aimed at the `/accounts` endpoint will include the newly generated auth token, allowing the client to consume other API endpoints right away.

5. Securing other API endpoints

As we've seen before, passing an authentication class to the `Eve` object enables authentication for all API endpoints. Again, you can still fine-tune security by allowing public access to certain endpoints or to certain HTTP methods. See *Authentication and Authorization* for more details.

6. Only allowing access to account resources

This is achieved with the *User-Restricted Resource Access* feature, as seen in *Accounts with Basic Authentication*. You might want to store the user token as your `AUTH_FIELD` value, but if you want user tokens to be easily revocable, then your best option is to use the account unique id for this.

Basic vs Token: Final Considerations

Despite being a little more tricky to set up on the server side, Token Authentication offers significant advantages. First, you don't have passwords stored on the client and being sent over the wire with every request. If you're sending your tokens out-of-band, and you're on SSL/TLS, that's quite a lot of additional security.

2.10.2 Handling custom ID fields

When it comes to individual document endpoints, in most cases you don't have anything to do besides defining the parent resource endpoint. So let's say that you configure a `/invoices` endpoint, which will allow clients to query the underlying *invoices* database collection. The `/invoices/<ObjectId>` endpoint will be made available by the framework, and will be used by clients to retrieve and/or edit individual documents. By default, Eve provides this feature seamlessly when `ID_FIELD` fields are of `ObjectId` type.

However, you might have collections where your unique identifier is not an `ObjectId`, and you still want individual document endpoints to work properly. Don't worry, it's doable, it only requires a little tinkering.

Handling `UUID` fields

In this tutorial we will consider a scenario in which one of our database collections (invoices) uses UUID fields as unique identifiers. We want our API to expose a document endpoint like `/invoices/uuid`, which translates to something like:

`/invoices/48c00ee9-4dbe-413f-9fc3-d5f12a91de1c`.

These are the steps we need to follow:

1. Craft a custom JSONEncoder that is capable of serializing UUIDs as strings and pass it to our Eve application.
2. Add support for a new `uuid` data type so we can properly validate incoming uuid values.
3. Configure our invoices endpoint so Eve knows how to properly parse UUID urls.

Custom JSONEncoder

The Eve default JSON serializer is perfectly capable of serializing common data types like `datetime` (serialized to a RFC1123 string, like `Sat, 23 Feb 1985 12:00:00 GMT`) and `ObjectId` values (also serialized to strings).

Since we are adding support for an unknown data type, we also need to instruct our Eve instance on how to properly serialize it. This is as easy as subclassing a standard `JSONEncoder` or, even better, Eve's own `BaseJSONEncoder`, so our custom serializer will preserve all of Eve's serialization magic:

```
from eve.io.base import BaseJSONEncoder
from uuid import UUID

class UUIDEncoder(BaseJSONEncoder):
    """ JSONEconder subclass used by the json render function.
    This is different from BaseJSONEoncoder since it also addresses
    encoding of UUID
    """

    def default(self, obj):
        if isinstance(obj, UUID):
            return str(obj)
        else:
            # delegate rendering to base class method (the base class
            # will properly render ObjectIds, datetimes, etc.)
            return super(UUIDEncoder, self).default(obj)
```

`UUID` Validation

By default Eve creates a unique identifier for each newly inserted document, and that is of `ObjectId` type. This is not what we want to happen at this endpoint. Here we want the client itself to provide the unique identifiers, and we also want to validate that they are of UUID type. In order to achieve that, we first need to extend our data validation layer (see *Data Validation* for details on custom validation):

```
from eve.io.mongo import Validator
from uuid import UUID

class UUIDValidator(Validator):
    """
    Extends the base mongo validator adding support for the uuid data-type
    """
    def _validate_type_uuid(self, value):
        try:
            UUID(value)
            return True
        except ValueError:
            pass
```

UUID URLs

Now Eve is capable of rendering and validating UUID values but it still doesn't know which resources are going to use these features. We also need to set `item_url` so uuid formed urls can be properly parsed. Let's pick our `settings.py` module and update the API domain accordingly:

```
invoices = {
    # this resource item endpoint (/invoices/<id>) will match a UUID regex.
    'item_url': 'regex("[a-f0-9]{8}-?[a-f0-9]{4}-?4[a-f0-9]{3}-?[89ab][a-f0-9]{3}-?[a-f0-
↪9]{12}")',
    'schema': {
        # set our _id field of our custom uuid type.
        '_id': {'type': 'uuid'},
    },
}

DOMAIN = {
    'invoices': invoices
}
```

If all your API resources are going to support uuid as unique document identifiers then you might just want to set the global `ITEM_URL` to the uuid regex in order to avoid setting it for every single resource endpoint.

Passing the UUID juice to Eve

Now all the missing pieces are there we only need to instruct Eve on how to use them. Eve needs to know about the new data type when its building the URL map, so we need to pass our custom classes right at the beginning, when we are instancing the application:

```
app = Eve(json_encoder=UUIDEncoder, validator=UUIDValidator)
```

Remember, if you are using custom `ID_FIELD` values then you should not rely on MongoDB (and Eve) to auto-generate the `ID_FIELD` for you. You are supposed to pass the value, like so:

```
POST
{"name":"bill", "_id":"48c00ee9-4dbe-413f-9fc3-d5f12a91de1c"}
```

Note: By default, Eve sets PyMongo's `UuidRepresentation` to `standard`. This allows for seamlessly handling of modern Python-generated UUID values. You can change the default by setting the `uuidRepresentation` value of `MONGO_OPTIONS` as desired. For more informations, see PyMongo documentation.

2.10.3 Learn Eve at TalkPython Training

There is a 5 hours-long Eve course available for you at the fine TalkPython Training website. The teacher is Nicola, Eve author and maintainer. Taking this course will directly support the project.

- Take the Eve Course at TalkPython Training

2.11 Snippets

Welcome to the Eve snippet archive. This is the place where anyone can drop helpful pieces of code for others to use.

2.11.1 Available Snippets

Using Eve Event Hooks from your Blueprint

by Pau Freixes

The use of Flask Blueprints helps us to extend our Eve applications with new endpoints that do not fit as a typical Eve resource. Pulling these endpoints out of the Eve scope allows us to write specific code in order to handle specific situations.

In the context of a Blueprint we could expect Eve features not be available, but often that is not the case. We can continue to use a bunch of features, such as *Event Hooks*.

Next snippet displays how the `users` module has a blueprint which performs some custom actions and then uses the `users_deleted` signal to notify and invoke all callback functions which are registered to the Eve application.

```
from flask import Blueprint, current_app as app

blueprint = Blueprint('prefix_uri', __name__)

@blueprint.route('/users/<username>', methods=['DELETE'])
def del_user(username):
    # some specific code goes here
    # ...

    # call Eve-hooks consumers for this  event
    getattr(app, "users_deleted")(username)
```

Next snippet displays how the blueprint is binded over our main Eve application and how the specific `set_username_as_none` function is registered to be called each time an user is deleted using the Eve events, to update the properly MongoDB collection.

```
from eve import Eve
from users import blueprint
from flask import current_app, request

def set_username_as_none(username):
    resource = request.endpoint.split('|')[0]
    return  current_app.data.driver.db[resource].update(
        {"user" : username},
        {"$set": {"user": None}},
        multi=True
```

(continues on next page)

(continued from previous page)

```
    )

app = Eve()
# register the blueprint to the main Eve application
app.register_blueprint(blueprint)
# bind the callback function so it is invoked at each user deletion
app.users_deleted += set_username_as_none
app.run()
```

Supporting both list-level and item-level CRUD operations

by John Chang

This is an example of how to implement a simple list of items that supports both list-level and item-level CRUD operations.

Specifically, it should be possible to use a single GET to get the entire list (including all items) but also a single POST to append an item (rather than PATCHing the list).

The solution was to database event hooks to inject the embedded child documents (`items`) into the parent list before it's returned to the client and also delete the child items when the parent list is deleted. This works, although it results in two DB queries.

main.py

```
from eve import Eve
from bson.objectid import ObjectId

app = Eve()
mongo = app.data.driver


def after_fetching_lists(response):
    list_id = response['_id']
    f = {'list_id': ObjectId(list_id)}
    response['items'] = list(mongo.db.items.find(f))


def after_deleting_lists(item):
    list_id = item['_id']
    f = {'list_id': ObjectId(list_id)}
    mongo.db.items.delete_many(f)

app.on_fetched_item_lists += after_fetching_lists
app.on_deleted_item_lists += after_deleting_lists

app.run()
```

settings.py

```
import os

DEBUG = True

MONGO_HOST = os.environ.get('MONGO_HOST', 'localhost')
MONGO_PORT = os.environ.get('MONGO_PORT', 27017)
MONGO_USERNAME = os.environ.get('MONGO_USERNAME', 'user')
MONGO_PASSWORD = os.environ.get('MONGO_PASSWORD', 'user')
MONGO_DBNAME = os.environ.get('MONGO_DBNAME', 'listtest')

RESOURCE_METHODS = ['GET', 'POST', 'DELETE']
ITEM_METHODS = ['GET', 'PUT', 'PATCH', 'DELETE']

DOMAIN = {
    'lists': {
        'schema': {
            'title': {
                'type': 'string'
            }
        }
    },
    'items': {
        'url': 'lists/<regex("[a-f0-9]{24}"):list_id>/items',
        'schema': {
            'list_id': {
                'type': 'objectid',
                'data_relation': {
                    'resource': 'lists',
                    'field': '_id'
                }
            },
            'name': {
                'type': 'string',
                'required': True
            }
        }
    }
}
```

Usage

```
$ curl -i -X POST http://127.0.0.1:5000/lists -d title="My List"
HTTP/1.0 201 CREATED

{
    "_id": "58960f83a663e2e6746dfa6a",
    :
}
```

(continues on next page)

(continued from previous page)

```
$ curl -i -X POST http://127.0.0.1:5000/lists/58960f83a663e2e6746dfa6a/items -d
↪'name=Alice'
HTTP/1.0 201 CREATED

$ curl -i -X POST http://127.0.0.1:5000/lists/58960f83a663e2e6746dfa6a/items -d 'name=Bob
↪'
HTTP/1.0 201 CREATED

$ curl -i -X GET http://127.0.0.1:5000/lists/58960f83a663e2e6746dfa6a
HTTP/1.0 200 OK

{
    "_created": "Sat, 04 Feb 2017 17:29:39 GMT",
    "_etag": "01799f6be25a044ab95cfeb2dc0f834d11b796d8",
    "_id": "58960f83a663e2e6746dfa6a",
    "_updated": "Sat, 04 Feb 2017 17:29:39 GMT",
    "items": [
        {
            "_created": "Sat, 04 Feb 2017 17:30:06 GMT",
            "_etag": "72ad9248ad5bf45c7bfe3e03a1b9bc384d94572f",
            "_id": "58960f9ea663e2e6746dfa6b",
            "_updated": "Sat, 04 Feb 2017 17:30:06 GMT",
            "list_id": "58960f83a663e2e6746dfa6a",
            "name": "Alice",
            "quantity": 1
        },
        {
            "_created": "Sat, 04 Feb 2017 17:30:13 GMT",
            "_etag": "447f51b057fb5e0a70472e96ff883c64b5e2e308",
            "_id": "58960fa5a663e2e6746dfa6c",
            "_updated": "Sat, 04 Feb 2017 17:30:13 GMT",
            "list_id": "58960f83a663e2e6746dfa6a",
            "name": "Bob",
            "quantity": 1
        }
    ],
    "title": "My List"
}

$ curl -i -X DELETE http://127.0.0.1:5000/lists/58960f83a663e2e6746dfa6a/items/
↪58960f9ea663e2e6746dfa6b -H "If-Match: 72ad9248ad5bf45c7bfe3e03a1b9bc384d94572f"
HTTP/1.0 204 NO CONTENT

$ curl -i -X GET http://127.0.0.1:5000/lists/58960f83a663e2e6746dfa6a
HTTP/1.0 200 OK

{
    "_created": "Sat, 04 Feb 2017 17:29:39 GMT",
    "_etag": "01799f6be25a044ab95cfeb2dc0f834d11b796d8",
    "_id": "58960f83a663e2e6746dfa6a",
    "_updated": "Sat, 04 Feb 2017 17:29:39 GMT",
    "items": [
```

(continues on next page)

(continued from previous page)

```
        {
            "_created": "Sat, 04 Feb 2017 17:30:13 GMT",
            "_etag": "447f51b057fb5e0a70472e96ff883c64b5e2e308",
            "_id": "58960fa5a663e2e6746dfa6c",
            "_updated": "Sat, 04 Feb 2017 17:30:13 GMT",
            "list_id": "58960f83a663e2e6746dfa6a",
            "name": "Bob",
            "quantity": 1
        }
    ],
    "title": "My List"
}
```

2.11.2 Add your snippet

Want to add your snippet? Just add your own .rst file to the snippets folder (see the template below for reference), update the TOC in this page (see source), and then submit a pull request.

Snippet Template

by Firstname Lastname

This is a snippet template. Put your snippet explanation here. If this is going to be long, make sure to split it into paragraphs for enhanced reading experience. Make your code snippet follow, like so:

```
from eve import Eve

# just an example of a code snippet
app = Eve()
app.run()
```

Add closing comments as needed.

2.12 Extensions

Welcome to the Eve extensions registry. Here you can find a list of packages that extend Eve. This list is moderated and updated on a regular basis. If you wrote a package for Eve and want it to show up here, just get in touch and show me your tool!

- Eve-Auth-JWT
- Eve-Elastic
- Eve-Healthcheck
- Eve-Mocker
- Eve-Mongoengine
- Eve-Neo4j
- Eve-OAuth2 and Flask-Sentinel
- Eve-SQLAlchemy

- Eve-Swagger
- Eve.NET
- EveGenie
- *REST Layer for Golang*

2.12.1 Eve-Auth-JWT

by Olivier Poitrey

Eve-Auth-JWT is An OAuth 2 JWT token validation module for Eve.

2.12.2 Eve-Elastic

by Petr Jašek

Eve-Elastic is an elasticsearch data layer for the Eve REST framework. Features facets support and the generation of mapping for schema.

2.12.3 Eve-Healthcheck

by LuisComS

Eve-Healthcheck is project that servers healthcheck urls used to monitor your Eve application.

2.12.4 Eve-Mocker

by Thomas Sileo

Eve-Mocker is a mocking tool for Eve powered REST APIs, based on the excellent HTTPretty, aimed to be used in your unit tests, when you rely on an Eve API. Eve-Mocker has been featured on the Eve blog: Mocking tool for Eve APIs

2.12.5 Eve-Mongoengine

by Stanislav Heller

Eve-Mongoengine is an Eve extension, which enables Mongoengine ORM models to be used as eve schema. If you use mongoengine in your application and simultaneously want to use Eve, instead of writing schema again in Cerberus format (DRY!), you can use this extension, which takes your mongoengine models and auto-transforms them into Cerberus schema under the hood.

2.12.6 Eve-Neo4j

by Abraxas Biosystems

Eve-Neo4j is an Eve extension aiming to enable it's users to build and deploy highly customizable, fully featured RESTful Web Services using Neo4j as backend. Powered by Eve, Py2neo, flask-neo4j and good intentions.

2.12.7 Eve-OAuth2

by Nicola Iarocci

Eve-OAuth2 is not an extension per-se, but rather an example of how you can leverage Flask-Sentinel to protect your API endpoints with OAuth2.

2.12.8 Eve-SQLAlchemy

by Andrew Mleczko et al.

Powered by Eve, SQLAlchemy and good intentions Eve-SQLALchemy allows to effortlessly build and deploy highly customizable, fully featured RESTful Web Services with SQL-based backends.

2.12.9 Eve-Swagger

by Nicola Iarocci

Eve-Swagger is a swagger.io extension for Eve. With a Swagger-enabled API, you get interactive documentation, client SDK generation and discoverability. From Swagger website:

> Swagger is a simple yet powerful representation of your RESTful API. With the largest ecosystem of API tooling on the planet, thousands of developers are supporting Swagger in almost every modern programming language and deployment environment. With a Swagger-enabled API, you get interactive documentation, client SDK generation and discoverability.

For more information, see also the Meet Eve-Swagger article.

2.12.10 Eve.NET

by Nicola Iarocci

Eve.NET is a simple HTTP and REST client for Web Services powered by the Eve Framework. It leverages both `System.Net.HttpClient` and `Json.NET` to provide the best possible Eve experience on the .NET platform. Written and maintained by the same author of the Eve Framework itself, Eve.NET is delivered as a portable library (PCL) and runs seamlessly on .NET4, Mono, Xamarin.iOS, Xamarin.Android, Windows Phone 8 and Windows 8. We use Eve.NET internally to power our iOS, Web and Windows applications.

2.12.11 EveGenie

by Erin Corson and Matt Tucker, maintained by David Zisky.

EveGenie is a tool for generating Eve schemas. It accepts a json document of one or more resources and provides you with a starting schema definition.

2.12.12 REST Layer for Golang

If you are into Golang, you should also check REST Layer. Developed by Olivier Poitrey, a long time Eve contributor and sustainer. REST Layer is

> a REST API framework heavily inspired by the excellent Python Eve. It lets you automatically generate a comprehensive, customizable, and secure REST API on top of any backend storage with no boiler plate code. You can focus on your business logic now.

2.13 How to contribute

Contributions are welcome! Not familiar with the codebase yet? No problem! There are many ways to contribute to open source projects: reporting bugs, helping with the documentation, spreading the word and of course, adding new features and patches.

2.13.1 Support questions

Please, don't use the issue tracker for this. Use one of the following resources for questions about your own code:

- Ask on Stack Overflow. Search with Google first using: `site:stackoverflow.com eve {search term, exception message, etc.}`
- The mailing list is intended to be a low traffic resource for both developers/contributors and API maintainers looking for help or requesting feedback.
- The IRC channel `#python-eve` on FreeNode.

2.13.2 Reporting issues

- Describe what you expected to happen.
- If possible, include a minimal, complete, and verifiable example to help us identify the issue. This also helps check that the issue is not with your own code.
- Describe what actually happened. Include the full traceback if there was an exception.
- List your Python and Eve versions. If possible, check if this issue is already fixed in the repository.

2.13.3 Submitting patches

- Include tests if your patch is supposed to solve a bug, and explain clearly under which circumstances the bug happens. Make sure the test fails without your patch.
- Enable and install pre-commit to ensure styleguides and codechecks are followed. CI will reject a change that does not conform to the guidelines.

First time setup

- Download and install the latest version of git.
- Configure git with your username and email:

```
git config --global user.name 'your name'
git config --global user.email 'your email'
```

- Make sure you have a GitHub account.
- Fork Eve to your GitHub account by clicking the Fork button.
- Clone your GitHub fork locally:

```
git clone https://github.com/{username}/eve
cd eve
```

- Add the main repository as a remote to update later:

```
git remote add pyeve https://github.com/pyeve/eve
git fetch pyeve
```

- Create a virtualenv:

```
python3 -m venv env
. env/bin/activate
# or "env\Scripts\activate" on Windows
```

- Install Eve in editable mode with development dependencies:

```
pip install -e ".[dev]"
```

- Install pre-commit and then activate its hooks. pre-commit is a framework for managing and maintaining multi-language pre-commit hooks. Eve uses pre-commit to ensure code-style and code formatting is the same:

```
$ pip install --user pre-commit
$ pre-commit install
```

  Afterwards, pre-commit will run whenever you commit.

Start coding

- Create a branch to identify the issue you would like to work on (e.g. `fix_for_#1280`)
- Using your favorite editor, make your changes, committing as you go.
- Follow PEP8.
- Include tests that cover any code changes you make. Make sure the test fails without your patch. *Run the tests.*.
- Push your commits to GitHub and create a pull request.
- Celebrate

Running the tests

You should have Python 3.7+ available in your system. Now running tests is as simple as issuing this command:

```
$ tox -e linting,py37,py38
```

This command will run tests via the "tox" tool against Python 3.7 and 3.8 and also perform "lint" coding-style checks.

You can pass different options to `tox`. For example, to run tests on Python 3.10 and pass options to pytest (e.g. enter pdb on failure) to pytest you can do:

```
$ tox -e py310 -- --pdb
```

Or to only run tests in a particular test module on Python 3.6:

```
$ tox -e py310 -- -k TestGet
```

CI will run the full suite when you submit your pull request. The full test suite takes a long time to run because it tests multiple combinations of Python and dependencies. You need to have Python 3.7, 3.8, 3.9, 3.10 and PyPy installed to run all of the environments. Then run:

```
tox
```

Please note that you need an active MongoDB instance running on localhost in order for the tests run. Save yourself some time and headache by creating a MongoDB user with the password defined in the *test_settings.py* file in the admin database (the pre-commit process is unforgiving if you don't want to commit your admin credentials but still have the file modified, which would be necessary for tox). If you want to run a local MongoDB instance along with an SSH tunnel to a remote instance, if you can, have the local use the default port and the remote use some other port. If you can't, fixing the tests that won't play nicely is probably more trouble than connecting to the remote and local instances one at a time. Also, be advised that in order to execute the *Rate Limiting* tests you need a running Redis server. The Rate-Limiting tests are silently skipped if any of the two conditions are not met.

Building the docs

Build the docs in the `docs` directory using Sphinx:

```
cd docs
make html
```

Open `_build/html/index.html` in your browser to view the docs.

Read more about Sphinx.

make targets

Eve provides a `Makefile` with various shortcuts. They will ensure that all dependencies are installed.

- `make test` runs the basic test suite with `pytest`
- `make test-all` runs the full test suite with `tox`
- `make docs` builds the HTML documentation
- `make check` performs some checks on the package
- `make install-dev` install Eve in editable mode with all development dependencies.

2.13.4 First time contributor?

It's alright. We've all been there. See next chapter.

2.13.5 Don't know where to start?

There are usually several TODO comments scattered around the codebase, maybe check them out and see if you have ideas, or can help with them. Also, check the open issues in case there's something that sparks your interest. And what about documentation? I suck at English, so if you're fluent with it (or notice any typo and/or mistake), why not help with that? In any case, other than GitHub help pages, you might want to check this excellent Effective Guide to Pull Requests

2.14 Support

Please keep in mind that the issues on GitHub are reserved for bugs and feature requests. If you have general or usage questions about Eve, there are several options:

2.14.1 Stack Overflow

Stack Overflow has a eve tag. It is generally followed by Eve developers and users.

2.14.2 Mailing List

The mailing list is intended to be a low traffic resource for both developers/contributors and API maintainers looking for help or requesting feedback.

2.14.3 IRC

There is an official Freenode channel for Eve at #python-eve.

2.14.4 File an Issue

If you notice some unexpected behavior in Eve, or want to see support for a new feature, file an issue on GitHub.

2.15 Updates

If you'd like to stay up to date on the community and development of Eve, there are several options:

2.15.1 Blog

Eve News is the official blog of the Eve project.

2.15.2 Twitter

I often tweet about new features and releases of Eve. Follow @nicolaiarocci.

2.15.3 Mailing List

The mailing list is intended to be a low traffic resource for both developers/contributors and API maintainers looking for help or requesting feedback.

2.15.4 GitHub

Of course the best way to track the development of Eve is through the GitHub repo.

2.16 Authors

Eve is written and maintained by Nicola Iarocci and various contributors:

2.16.1 Development Lead

- Nicola Iarocci <eve@nicolaiarocci.com>

2.16.2 Patches and Contributions

- Aayush Sarva
- Adam Walsh
- Adrian Cin
- Alberto Marin
- Alex Misk
- Alexander Dietmüller
- Alexander Hendorf

- Alexander Miskaryan
- Amedeo Bussi
- Andreas Røssland
- Andrés Martano
- Antonio Lourenco
- Arnau Orriols
- Artem Kolesnikov
- Arthur Burkart
- Ashley Roach
- Ben Demaree
- Bjorn Andersson
- Brad P. Crochet
- Brian Mego
- Bryan Cattle
- Carl George
- Carles Bruguera
- Chen Rotem
- Christian Henke
- Christoph Witzany
- Christopher Larsen
- Chuck Turco
- Conrad Burchert
- Cyprien Pannier
- Cyril Bonnard
- DHuan
- Daniel Lytkin
- Daniele Pizzolli
- Danse
- David Arnold
- David Booss
- David Buchmann
- David Murphy
- David Wood
- Dmitry Anoshin
- Dominik Kellner
- Dong Wei Ming

- Dougal Matthews
- Einar Huseby
- Elias García
- Emmanuel Leblond
- Eugene Prikazchikov
- Ewan Higgs
- Felix Peppert
- Florian Rathgeber
- Fouad Chennou
- Francisco Corrales Morales
- Garrin Kimmell
- George Lestaris
- Gianfranco Palumbo
- Gino Zhang
- Giorgos Margaritis
- Gonéri Le Bouder
- Grisha K.
- Guillaume Royer
- Gustavo Vargas
- Hamdy
- Hannes Tiede
- Harro van der Klauw
- Hasan Pekdemir
- Henrique Barroso
- Huan Di
- Hugo Larcher
- Hung Le
- James Stewart
- Jaroslav Semančík
- Javier Gonel
- Javier Jiménez
- Jean Boussier
- Jeff Zhang
- Jen Montes
- Jeremy Solbrig
- Joakim Uddholm

- Johan Bloemberg
- John Chang
- John Deng
- Jorge Morales
- Jorge Puente Sarrín
- Joseph Heck
- Josh Villbrandt
- Juan Madurga
- Julian Hille
- Julien Barbot
- Junior Vidotti
- Kai Danielmeier
- Kelly Caylor
- Ken Carpenter
- Kevin Bowrin
- Kevin Funk
- Kevin Roy
- Kracekumar
- Kris Lambrechts
- Kurt Bonne
- Kurt Doherty
- Luca Di Gaspero
- Luca Moretto
- Luis Fernando Gomes
- Magdas Adrian
- Mamurjon Saitbaev
- Mandar Vaze
- Manquer
- Marc Abramowitz
- Marcelo Trylesinski
- Marcin Puhacz
- Marcus Cobden
- Marica Odagaki
- Mario Kralj
- Mark Mayo
- Marsch Huynh

- Martin Fous
- Massimo Scamarcia
- Mateusz Łoskot
- Matt Creenan
- Matt Tucker
- Matthew Ellison
- Matthieu Prat
- Mattias Lundberg
- Mayur Dhamanwala
- Michael Maxwell
- Mikael Berg
- Miroslav Šedivý
- Moritz Schneider
- Mugur Rus
- Nathan Reynolds
- Niall Donegan
- Nick Park
- Nicolas Bazire
- Nicolas Carlier
- Oleg Pshenichniy
- Olivier Carrère
- Olivier Poitrey
- Olof Johansson
- Ondrej Slinták
- Or Neeman
- Orange Tsai
- Pahaz Blinov
- Patricia Ramos
- Patrick Decat
- Pau Freixes
- Paul Doucet
- Pedro Rodrigues
- Peter Darrow
- Petr Jašek
- Phone Myint Kyaw
- Pieter De Clercq

- Prajjwal Nijhara
- Prayag Verma
- Qiang Zhang
- Raghuram Devarakonda
- Rahul Salgare
- Ralph Smith
- Raychee
- Robert Wlodarczyk
- Roberto 'Kalamun' Pasini
- Rodrigo Rodriguez
- Roller Angel
- Roman Gavrilov
- Ronan Delacroix
- Roy Smith
- Ryan Shea
- Sam Luu
- Samuel Sutch
- Samuli Tuomola
- Saurabh Shandilya
- Sebastien Estienne
- Sebastián Magrí
- Serge Kir
- Shaoyu Meng
- Simon Schönfeld
- Sobolev Nikita
- Stanislav Filin
- Stanislav Heller
- Stefaan Ghysels
- Stratos Gerakakis
- Sybren A. Stüvel
- Tadej Magajn
- Tano Abeleyra
- Taylor Brown
- Thomas Sileo
- Tim Gates
- Tim Jacobi

- Tomasz Jezierski
- Tyler Kennedy
- Valerie Coffman
- Vasilis Lolis
- Vincent Bisserie
- Wael M. Nasreddine
- Wan Bachtiar
- Wei Guan
- Wytamma Wirth
- Xavi Cubillas
- boosh
- dccrazyboy
- kinuax
- kreynen
- mmizotin
- quentinpraz
- smeng9
- tgm
- xgdgsc

2.17 Licensing

Also see *Authors*.

2.17.1 BSD License

DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE AND DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2.17.2 Artwork License

Eve artwork 2013 by Roberto Pasini "Kalamun" released under the Creative Commons BY-SA license.

2.18 Changelog

Here you can see the full list of changes between each Eve release.

2.18.1 In Development

- *hic sunt leones*

2.18.2 Version v2.1.0

Released on Mar 14, 2023.

New

- Ability to customize the pagination limit on a per-resource basis (#1498)

Fixed

- fix: Flask 2.2+ support (#1497)
- fix: CI test runs fail with `mongo: command not found` on Ubuntu 22.04 (#1499)

2.18.3 Version v2.0.4

Released on Nov 10, 2022.

Fixed

- Comparison of incompatible types (#1492)
- Python 3 updates, and some refactoring (#1493)

2.18.4 Version v2.0.3

Released on Nov 2, 2022.

Fixed

- Malformed `LAST_UPDATED` field (#1490)

2.18.5 Version v2.0.2

Released on Sep 23, 2022.

Fixed

- Fix: etag generation fails if `uuidRepresentation` is not set in MONGO_OPTIONS (#1486)

2.18.6 Version v2.0.1

Released on Sep 7, 2022.

Fixed

- `MONGO_URI` username, password, and authSource are not parsed correctly (#1478)
- Lock Flask dependency to version 2.1 (#1485)
- Fix documentation typos (#1481)
- Only build Python 3 wheels.

2.18.7 Version v2.0

Released on Jun 8, 2022.

Breaking

Starting from this release, Eve supports Python 3.7 and above.

- Drop Python 2 (#1440)
- Drop Python 3.5 (#1440, #1438)
- Drop Python 3.6 (#1440)

New

- Add Python 3.9 support (#1437)
- Add Python 3.10 support (#1440)
- `MONGO_OPTIONS` acquires a new `uuidRepresentation` setting, with `standard` as its default value. This is needed by PyMongo 4+ in order to seamlessly process eventual `uuid` values. See PyMongo documentation for details (#1461, #1464).

Fixed

- AttributeError: module 'werkzeug.utils' has no attribute 'escape' (#1474)
- Starting with Werkzeug 2.1, HATEOAS links are relative instead of absolute (#1475)
- Eve doesn't work with latest PyMongo (v4) (#1461, #1464)
- Fix 500 error with empty token/bearer (#1456)
- Do not return related fields if field is a empty list (#1441)
- PyMongo 3.12+ supports keys that include dotted fields (#1466)
- Pin pymongo version in dependencies (#1461)
- Prepare for Python 3 switch (#1445)
- Update docs and tests regarding pagination of empty resources (#1463)
- Fix fork link in contributing info (#1447)
- Tutorial mistake on custom IDs values with UUIDs (#1451)
- Documentation typos (#1462, #1469)
- Switch to GitHub Actions from Travis CI (#1439, #1444)

2.18.8 Version 1.1.5

Released on January 25, 2021.

Fixed

- Nested unique field validation still don't work (#1435)
- Documentation: corrected variable name (#1426)
- Versioning: support for dynamic datasources (#1423)
- Disable MD5 support in GridFS, as it is deprecated (#1410)
- Demo application has been terminated by Heroku. Dropped any reference to it.

2.18.9 Version 1.1.4

Released on October 22, 2020.

Fixed

- Error raised when using `embedded` with nested dict (#1416)
- Expose media endpoint only if `RETURN_MEDIA_AS_URL` is set to `True` (#1415)
- Use `**mongo_options` in `with_options` (#1413)

2.18.10 Version 1.1.3

Released on September 19, 2020.

Fixed

- Fix: Race condition in PATCH on newly created documents with clustered mongo (#1411)

2.18.11 Version 1.1.2

Released on July 9, 2020.

Fixed

- Add missed condition when projection is disabled per domain (#1398)
- Removed unnecessary comprehension (#1391)

2.18.12 Version 1.1.1

Released on May 10, 2020.

Fixed

- Disabling `merge_nested_documents` breaks versioning on PATCH (#1389)
- Tests failing with Flask 1.1.2 (#1378)
- `BANDWIDTH_SAVER` no longer works with `resolve_resource_projection` (#1338)
- `unique_within_resource` rule used in resources without datasource filter (#1368)
- dicts without `schema` rule are broken since `b8d8fcd` (#1366)
- 403 Forrbidden added to `STANDARD_ERRORS` (#1362)
- `unique` constraint doesn't work when inside of a dict or a list (#1360)
- Documentation typos (#1375)

2.18.13 Version 1.1

Released on February 7, 2020.

New

- `MONGO_QUERY_WHITELIST` and `mongo_query_whitelist`. A list of extra Mongo query operators to allow besides the official list of allowed operators. Defaults to `[]`. (#1351)

Fixed

- Starup crash with Werkzeug 1.0 (#1359)
- `$eq` is missing from supported query operators (#1351)
- Documentation typos (#1348, #1350)

2.18.14 Version 1.0.1

Released on January 26, 2020.

- Fix: Mixing foreign and local object ids breaks querying (#1345)

2.18.15 Version 1.0

Released on December 19, 2019.

New

- Python 3.8 added to CI matrix (#1326)
- Drop support for Python 3.4 (#1297)
- `unique_within_resource` validation rule. Enforces the uniqueness of an attribute only at API resource level, contrasting with the `unique` rule that enforces uniqueness at database collection level (#1291)
- Add doc8 to dev-requirements (#1343)

Fixed

- Pin to Cerberus < 2.0 (#1342)
- 500 error when PATCH or PUT are performed on Mongo 4.2 and `_id` is included with payload (#1341)
- Minor style improvements and 2 test fixes (#1330)
- Werkzeug 0.15.4 crashes with Python 3.8 (#1325)
- Curl request in projection examples do not work (#1298)
- Update installation instructions (#1303)
- (*breaking*) Delete on empty resource returns 404, should return 204 (#1299)
- `MONGO_REPLICA_SET` ignored (#1302)

- Documentation typo (#1293, #1315, #1322, #1324, #1327)
- Flask 1.1.1 breaks `test_logging_info` test (#1296)
- Display the full release number on Eve frontpage.
- Update link to EveGenie repository. New maintainer: David Zisky.

2.18.16 Version 0.9.2

Released on June 14, 2019.

Fixed

- Geo queries lack support for the `$minDistance` mongo operator (#1281)
- Lookup argument does not get passed to `pre_<event>` hook with certain resource urls (#1283)
- PUT requests doesn't set default values for fields that have one defined (#1280)
- PATCH crashes when normalizing default fields (#1275, #1274)
- The condition that avoids returning `X-Total-Count` when counting is disabled also filters out the case where the resource is empty and count is 0 (#1279)
- First example of Eve use doesn't really work (#1277)

2.18.17 Version 0.9.1

Released on May 22, 2019.

New

- `NORMALIZE_ON_PATCH` switches normalization on patch requests (#1234)

Fixed

- Document count broken with concurrent requests (#1271)
- Document count broken when embedded resources are requested (#1268)
- If `ignore_fields` contains a nested field, document is mutated (#1266)
- Crash with Werzeug >= 0.15.3 (#1267)
- Fix crash when trying to ignore a nested field that doesn't exist (#1263)

Improved

- Remove unsupported `transparent_schema_rules` option from docs (#1264)
- Bump (and pin) Wekzeug to 0.15.4 (#1267)
- Quickstart: a better `MONGO_AUTH_SOURCE` explanation (#1168)

Breaking Changes

No known breaking changes for the standard framework user. However, if you are consuming the developer API:

- Be aware that `io.base.DataLayer.find()` signature has changed and an optional `perform_count` argument has been added. The method return value is now a tuple `(cursor, count)`; `cursor` is the query result as before while `count` is the document count, which is expected to have a consistent value when `perform_count = True`.

2.18.18 Version 0.9

Released on April 11, 2019.

Breaking changes

- Werkzeug v0.15.1+ is required. You want to upgrade, otherwise your Eve environment is likely to break. For the full story, see #1245 and #1251.

New

- HATEOAS support added to aggregation results (#1208)
- `on_fetched_diffs` event hooks (#1224)
- Support for Mongo 3.6+ `$expr` query operator.
- Support for Mongo 3.6+ `$center` query operator.

Fixed

- Insertion failure when replacing unknown field with dbref value (#1255, #1257)
- `max_results=1` should be honored on aggregation endpoints (#1250)
- PATCH incorrectly normalizes default values in subdocuments (#1234)
- Unauthorized Exception not working with Werkzeug >= 15.0 (#1245, #1251)
- Embedded documents not being sorted correctly (#1217)
- Eve crashes on malformed sort parameters (#1248)
- Insertion failure when replacing a same document containing dbref (#1216)
- Datasource projection is not respected for POST requests (#1189)
- Soft delete removes `auth_field` from document (#1188)
- On Mongo 3.6+, we don't return 400 'immutable field' on PATCH and PUT (#1243)

- Expecting JSON response for rate limit exceeded scenario (#1227)
- Multiple concurrent patches to the same record, from different processes, should result in at least one patch failing with a 412 error (Precondition Failed) (#1231)
- Embedding only does not follow `data_relation.field` (#1069)
- HATEOAS `_links` seems to get an extra `&version=diffs` (#1228)
- Do not alter ETag when performing an oplog_push (#1206)
- CORS response headers missing for media endpoint (#1197)
- Warning: Unexpected keys present on black: `python_version` (#1244)
- UserWarning: JSON setting is deprecated. Use RENDERERS instead (#1241).
- DeprecationWarning: decodestring is deprecated, use decodebytes (#1242)
- DeprecationWarning: count is deprecated. Use Collection.count_documents instead (#1202)
- Documentation typos (#1218, #1240)

Improved

- Eve package is now distributed as a Python wheel (#1260)
- Bump Werkzeug version to v0.15.1+ (#1245, #1251)
- Bump PyMongo version to v3.7+ (#1202)
- Python 3.7 added to the CI matrix (#1199)
- Option to omit the aggregation stage when its parameter is empty/unset (#1209)
- HATEOAS: now the `_links` dictionary may have a `related` dictionary inside, and each key-value pair yields the related links for a data relation field (#1204)
- XML renderer now supports data field tag attributes such as `href` and `title` (#1204)
- Make the parsing of `req.sort` and `req.where` easily reusable by moving their logic to dedicated methods (#1194)
- Add a "Python 3 is highly preferred" note on the homepage (#1198)
- Drop sphinx-contrib-embedly when building docs.

2.18.19 Version 0.8.1

Released on October 4, 2018.

New

- Add support for Mongo `$centerSphere` query operator (#1181)
- `NORMALIZE_DOTTED_FIELDS`. If `True`, dotted fields are parsed and processed as subdocument fields. If `False`, dotted fields are left unparsed and unprocessed and the payload is passed to the underlying data-layer as-is. Please note that with the default Mongo layer, setting this to `False` will result in an error. Defaults to `True`. (#1173)
- `normalize_dotted_fields`. Endpoint-level override for `NORMALIZE_DOTTED_FIELDS`. (#1173)

Fixed

- `mongo_indexes`: "OperationFailure" when changing the keys of an existing index (#1180)
- v0.8: "OperationFailure" performing MongoDB full text searches (#1176)
- "AttributeError" on Python 2.7 when obsolete `JSON` or `XML` settings are used (#1175).
- "TypeError argument of type 'NoneType' is not iterable" error when using document embedding in conjuction with soft deletes (#1120)
- `allow_unknown` validation rule fails with nested dict fields (#1163)
- Updating a field with a nullable data relation fails when value is null (#1159)
- "cerberus.schema.SchemaError" when `VALIDATE_FILTERS = True`. (#1154)
- Serializers fails when array of types is in schema. (#1112)
- Replace the broken `make audit` shortcut with `make check`, add the command to `CONTRIBUTING.rst` it was missing. (#1144)

Improved

- Perform lint checks and fixes on staged files, as a pre-commit hook. (#1157)
- On CI, perform linting checks first. If linting checks are successful, execute the test suite on the whole matrix. (#1156)
- Reformat code to match Black code-style. (#1155)
- Use `simplejson` everywhere in the codebase. (#1148)
- Install a bot that flags and closes stale issues/pull requests. (#1145)
- Only set the package version in `__init__.py`. (#1142)

Docs

- Typos (#1183, #1184, #1185)
- Add `MONGO_AUTH_SOURCE` to Quickstart. (#1168)
- Fix Sphinx-embedly error when embedding speakerdeck.com slide deck (#1158)
- Fix broken link to the Postman app. (#1150)
- Update obsolete PyPI link in docs sidebar. (#1152)
- Only display the version number on the docs homepage. (#1151)
- Fix documentation builds on Read the Docs. (#1147)
- Add a `ISSUE_TEMPLATE.md` GitHub template file. (#1146)
- Improve changelog format to reduce noise and increase readability. (#1143)

2.18.20 Version 0.8

Released on May 10, 2018.

Note: Make sure you read the *Breaking Changes* section below.

- New: support for partial media requests. Clients can request partial file downloads by adding a `Range` header to their media request (#1050).
- New: Renderer classes. `RENDERER` allows to change enabled renderers. Defaults to `['eve.render.JSONRenderer', 'eve.render.XMLRenderer']`. You can create your own renderer by subclassing `eve.render.Renderer`. Each renderer should set valid mime attr and have `.render()` method implemented. Please note that at least one renderer must always be enabled (#1092).
- New: `on_delete_resource_originals` fired when soft deletion occurs (#1030).
- New: `before_aggregation` and `after_aggregation` event hooks allow to attach custom callbacks to aggregation endpoints (#1057).
- New: `JSON_REQUEST_CONTENT_TYPES` or supported JSON content types. Useful when you need support for vendor-specific json types. Please note: responses will still carry the standard `application/json` type. Defaults to `['application/json']` (#1024).
- New: when the media endpoint is enabled, the default authentication class will be used to secure it. (#1083; #1049).
- New: `MERGE_NESTED_DOCUMENTS`. If `True`, updates to nested fields are merged with the current data on `PATCH`. If `False`, the updates overwrite the current data. Defaults to `True` (#1140).
- New: support for MongoDB decimal type `bson.decimal128.Decimal128` (#1045).
- New: Support for `Feature` and `FeatureCollection` GeoJSON objects (#769).
- New: Add support for MongoDB `$box` geo query operator (#1122).
- New: `ALLOW_CUSTOM_FIELDS_IN_GEOJSON` allows custom fields in GeoJSON (#1004).
- New: Add support for MongoDB `$caseSensitive` and `$diactricSensitive` query operators (#1126).
- New: Add support for MongoDB bitwise query operators `$bitsAllClear`, `$bitsAllSet`, `$bitsAnyClear`, `$bitsAnySet` (#1053).
- New: support for `MONGO_AUTH_MECHANISM` and `MONGO_AUTH_MECHANISM_PROPERTIES`.
- New: `MONGO_DBNAME` can now be used in conjuction with `MONGO_URI`. Previously, if `MONGO_URI` was missing the database name, an exception would be rised (#1037).
- Fix: OPLOG skipped even if `OPLOG = True` (#1074).
- Fix: Cannot define default projection and request specific field. (#1036).
- Fix: `VALIDATE_FILTERS` and `ALLOWED_FILTERS` do not work with sub-document fields. (#1123).
- Fix: Aggregation query parameter does not replace keys in the lists (#1025).
- Fix: serialization bug that randomly skips fields if "x_of" is encountered (#1042)
- Fix: PUT behavior with User Restricted Resource Access. Ensure that, under every circumstance, users are unable to overwrite items owned by other users (#1130).
- Fix: Crash with Cerberus 1.2 (#1137).
- Fix documentation typos (#1114, #1102)

- Fix: broken documentation links to Cerberus validation rules.
- Fix: add sphinxcontrib-embedly to dev-requirements.txt.
- Fix: Removed OrderedDict dependency; use `OrderedDict` from `backport_collections` instead (#1070).
- Performance improved on retrieving a list of embedded documents (#1029).
- Dev: Refactor index creation. We now have a new `eve.io.mongo.ensure_mongo_indexes()` function which ensures that eventual `mongo_indexes` defined for a resource are created on the active database. The function can be imported and invoked, for example in multi-db workflows where a db is activated based on the authenticated user performing the request (via custom auth classes).
- Dev: Add a Makefile with shortcuts for testing, docs building, and development install.
- Dev: Switch to pytest as the standard testing tool.
- Dev: Drop `requiments.txt` and `dev-requirements.txt`. Use `pip install -e .[dev|tests|docs]` instead.
- Tests: finally acknowledge the existence of modern APIs for both Mongo and Python (get rid of most deprecation warnings).
- Change: Support for Cerberus 1.0+ (#776).
- Change: `JSON` and `XML` settings are deprecated and will be removed in a future update. Use `RENDERERS` instead (#1092).
- Flask dependency set to >=1.0 (#1111).
- PyMongo dependency set to >=3.5.
- Events dependency set to >=v0.3.
- Drop Flask-PyMongo dependency, use custom code instead (#855).
- Docs: Comprehensive rewrite of the How to contribute page.
- Docs: Drop the testing page; merge its contents with How to contribute.
- Docs: Add link to the Eve course. It was authored by the project author, and it is hosted by TalkPython Training.
- Docs: code snippets are now Python 3 compatibile (Pahaz Blinov).
- Dev: Delete and cleanup of some unnecessary code.
- Dev: after the latest update (May 4th) travis-ci would not run tests on Python 2.6.
- Dev: all branches are now tested on travis-ci. Previously, only 'master' was being tested.
- Dev: fix insidious bug in `tests.methods.post.TestPost` class.

Breaking Changes

- Python 2.6 and Python 3.3 are no longer supported (#1129).
- Eve now relies on Cerberus 1.1+ (#776). It allows for many new powerful validation and trasformation features (like schema registries), improved performance and, in general, a more streamlined API. It also brings some notable breaking changes.
  - `keyschema` was renamed to `valueschema`, and `propertyschema` to `keyschema`.
  - A PATCH on a document which misses a field having a default value will now result in setting this value, even if the field was not provided in the PATCH's payload.

  - Error messages for `keyschema` are now returned as dictionary. Example: `{'a_dict': {'a_field': "value does not match regex '[a-z]+'"}}`.
  - Error messages for type validations are different now.
  - It is no longer valid to have a field with `default = None` and `nullable = False` (see *patch.py:test_patch_nested_document_nullable_missing*).
  - And more. A complete list of breaking changes is available here. For detailed upgrade instructions, see Cerberus upgrade notes. An in-depth analysis of changes made to the codebase (useful if you wrote a custom validator which needs to be upgraded) is available with this commit message.
  - Special thanks to Dominik Kellner and Brad P. Crochet for the amazing job done on this upgrade.
- Config setting `MONGO_AUTHDBNAME` renamed into `MONGO_AUTH_SOURCE` for naming consistency with PyMongo.
- Config options `MONGO_MAX_POOL_SIZE`, `MONGO_SOCKET_TIMEOUT_MS`, `MONGO_CONNECT_TIMEOUT_MS`, `MONGO_REPLICA_SET`, `MONGO_READ_PREFERENCE` removed. Use `MONGO_OPTIONS` or `MONGO_URI` instead.
- Be aware that `DELETE` on sub-resource endpoint will now only delete the documents matching endpoint semantics. A delete operation on `people/51f63e0838345b6dcd7eabff/invoices` will delete all documents matching the followig query: `{'contact_id': '51f63e0838345b6dcd7eabff'}` (#1010).

Version 0.7.10

Released on July 15, 2018.

- Fix: Pin Flask-PyMongo dependency to avoid crash with Flask-PyMongo 2. Closes #1172.

Version 0.7.9

Released on May 10, 2018

- Python 2.6 and Python 3.3 are deprecated. Closes #1129.

Version 0.7.8

Released on 7 February, 2018

- Fix: breaking syntax error in v0.7.7

Version 0.7.7

Released on 7 February, 2018

- Fix: geo queries now properly support `$geometry` and `$maxDistance` operators. Closes #1103.

Version 0.7.6

Released on 14 January, 2018

- Improve query parsing robustness.

Version 0.7.5

Released on 4 December, 2017

- Fix: A query was not fully traversed in the sanitization. Therefore the blacklist for mongo wueries could be bypassed, allowing for dangerous `$where` queries (Moritz Schneider).

Version 0.7.4

Released on 24 May, 2017

- Fix: `post_internal` fails when using `URL_PREFIX` or `API_VERSION`. Closes #810.

Version 0.7.3

Released on 3 May, 2017

- Eve and Cerberus are now collaboratively funded projects, see: https://nicolaiarocci.com/eve-and-cerberus-funding-campaign/
- Fix: Internal resource, oplog enabled: a `*_internal` method defined in `OPLOG_METHODS` triggers keyerror (Einar Huseby).
- Dev: use official Alabaster theme instead of custom fork.
- Fix: docstrings typos (Martin Fous).
- Docs: explain that `ALLOW_UNKNOWN` can also be used to expose the whole document as found in the database, with no explicit validation schema. Addresses #995.
- Docs: add Eve-Healthcheck to extensions list (Luis Fernando Gomes).

Version 0.7.2

Released on 6 March, 2017

- Fix: Validation exceptions are returned in `doc_issues['validator exception']` across all edit methods (POST, PUT, PATCH). Closes #994.
- Fix: When there is `MONGO_URI` defined it will be used no matter if the resource is using a prefix or not (Petr Jašek).
- Docs: Add code snippet with an example of how to implement a simple list of items that supports both list-level and item-level CRUD operations (John Chang).

Version 0.7.1

Released on 14 February, 2017

- Fix: "Cannot create a consistent method resolution order" on Python 3.5.2 and 3.6 since Eve 0.7. Closes #984.
- Docs: update README with svg bade (Sobolev Nikita).
- Docs: fix typo and dead link to Nicola's website (Dominik Kellner).
- `develop` branch has been dropped. `master` is now the default project branch.

Version 0.7

Released on 6 February, 2017

- New: Add Python 3.6 as a supported interpreter.
- New: `OPTIMIZE_PAGINATION_FOR_SPEED`. Set this to `True` to improve pagination performance. When optimization is active no count operation, which can be slow on large collections, is performed on the database. This does have a few consequences. Firstly, no document count is returned. Secondly, `HATEOAS` is less accurate: no last page link is available, and next page link is always included, even on last page. On big collections, switching this feature on can greatly improve performance. Defaults to `False` (slower performance; document count included; accurate `HATEOAS`). Closes #944 and #853.
- New: `Location` header is returned on `201 Created` POST responses. If will contain the URI to the created document. If bulk inserts are enabled, only the first document URI is returned. Closes #795.
- New: Pretty printing.You can pretty print the response by specifying a query parameter named `?pretty` (Hasan Pekdemir).
- New: `AUTO_COLLAPSE_MULTI_KEYS`. If set to `True`, multiple values sent with the same key, submitted using the `application/x-www-form-urlencoded` or `multipart/form-data` content types, will automatically be converted to a list of values. When using this together with `AUTO_CREATE_LISTS` it becomes possible to use lists of media fields. Defaults to `False`. Closes #932 (Conrad Burchert).
- New: `AUTO_CREATE_LISTS`. When submitting a non `list` type value for a field with type `list`, automatically create a one element list before running the validators. Defaults to `False` (Conrad Burchert).
- New: Flask-PyMongo compatibility for for `MONGO_CONNECT` config setting (Massimo Scamarcia).
- New: Add Python 3.5 as a supported interpreter (Mattias Lundberg).
- New: `MONGO_OPTIONS` allows MongoDB arguments to be passed to the MongoClient object. Defaults to `{}` (Massimo Scamarcia).
- New: Regexes are allowed by setting `X_DOMAINS_RE` values. This allows CORS to support websites with dynamic ranges of subdomains. Closes #660 and #974.
- New: If `ENFORCE_IF_MATCH` option is active, then all requests are expected to include the `If-Match` or they will be rejected (same as old behavior). However, if `ENFORCE_IF_MATCH` is disabled, then client determines whether request is conditional. When `If-Match` is included, then request is conditional, otherwise the request is processed with no conditional checks. Closes #657 (Arthur Burkart).
- New: Allow old document versions to be cache validated using ETags (Nick Park).
- New: Support weak ETags, commonly applied by servers transmitting gzipped content (Nick Park).
- New: `on_oplog_push` event is fired when OPLOG is about to be updated. Callbacks receive two arguments: `resource` (resource name) and `entries` (list of oplog entries which are about to be written).
- New: optional `extra` field is available for OPLOG entries. Can be updated by callbacks hooked to the new `on_oplog_push` event.

- New: OPLOG audit now include the username or token when available. Closes #846.
- New `get_internal` and `getitem_internal` functions can be used for internal GET calls. These methods are not rate limited, authentication is not checked and pre-request events are not raised.
- New: Add support for MongoDB `DBRef` fields (Roman Gavrilov).
- New: `MULTIPART_FORM_FIELDS_AS_JSON`. In case you are submitting your resource as `multipart/form-data` all form data fields will be submitted as strings, breaking any validation rules you might have on the resource fields. If you want to treat all submitted form data as JSON strings you will have to activate this setting. Closes #806 (Stratos Gerakakis).
- New: Support for MongoDB Aggregation Framework. Endpoints can respond with aggregation results. Clients can optionally influence aggregation results by using the new `aggregate` option: `aggregate={"$year": 2015}`.
- New: Flask views (`@app.route`) can now set `mongo_prefix` via Flask's `g` object: `g.mongo_prefix = 'MONGO2'` (Gustavo Vargas).
- New: Query parameters not recognised by Eve are now returned in HATEOAS URLs (Mugur Rus).
- New: `OPLOG_CHANGE_METHODS` is a list of HTTP methods which operations will include changes into the OpLog (mmizotin).
- Change: Return `428 Precondition Required` instead of a generic `403 Forbidden` when the `If-Match` request header is missing (Arnau Orriols).
- Change: ETag response header now conforms to RFC 7232/2.3 and is surrounded by double quotes. Closes #794.
- Fix: Better locating of `settings.py`. On startup, if settings flag is omitted in constructor, Eve will try to locate file named settings.py, first in the application folder and then in one of the application's subfolders. You can choose an alternative filename/path, just pass it as an argument when you instantiate the application. If the file path is relative, Eve will try to locate it recursively in one of the folders in your sys.path, therefore you have to be sure that your application root is appended to it. This is useful, for example, in testing environments, when settings file is not necessarily located in the root of your application. Closes #820 (Mario Kralj).
- Fix: Versioning does not work with User Restricted Resource Access. Closes #967 (Kris Lambrechts)
- Fix: `test_create_indexes()` typo. Closes 960.
- Fix: fix crash when attempting to modify a document `_id` on MongoDB 3.4 (Giorgos Margaritis)
- Fix: improve serialization of boolean values. Closes #947 (NotSpecial).
- Fix: fix intermittently failing test. Closes #934 (Conrad Burchert).
- Fix: Multiple, fast (within a 1 second window) and neutral (no actual changes) PATCH requests should not raise `412 Precondition Failed`. Closes #920.
- Fix: Resource titles are not properly escaped during the XML rendering of the root document (Kris Lambrechts).
- Fix: ETag request headers which conform to RFC 7232/2.3 (double quoted value) are now properly processed. Addresses #794.
- Fix: Deprecation warning from Flask. Closes #898 (George Lestaris).
- Fix: add Support serialization on lists using anyof, oneof, allof, noneof. Closes #876 (Carles Bruguera).
- Fix: update security example snippets to match with current API (Stanislav Filin).
- Fix: `notifications.py` example snippet crashes due to lack of `DOMAIN` setting (Stanislav Filin).
- Docs: clarify documentation for custom validators: Cerberus dependency is still pinned to version 0.9.2. Upgrade to Cerberus 1.0+ is planned with v0.8. Closes #796.

- Docs: remove the deprecated `--ditribute` virtualenv option (Eugene Prikazchikov).
- Docs: add date and subdocument fields filtering examples. Closes #924.
- Docs: add Eve-Neo4j to the extensions page (Rodrigo Rodriguez).
- Docs: stress that alternate backends are supported via community extensions.
- Docs: clarify that Redis is an optional dependency (Mateusz Łoskot).
- Update license to 2017. Closes #955.
- Update: Flask 0.12. Closes #945, #904 and #963.
- Update: PyMongo 3.4 is now required. Closes #964.

Version 0.6.4

Released on 8 June, 2016

- Fix: Cannot serialize data when a field that has a `valueschema` that is of `dict` type. Closes #874.
- Fix: Authorization header bearer tokens not parsed correctly. Closes #866 (James Stewart).
- Fix: TokenAuth prevents base64 decoding of Tokens. Closes #840.
- Fix: If datasource source is specified no fields are included by default. Closes #842.
- Docs: streamline Quickstart guide. Closes #868.
- Docs: fix broken link in Installation page. Closes #861.
- Docs: Resource configuration doesn't mention `versioning` override. Closes #845.

Version 0.6.3

Released on 16 March, 2016

- Fix: Since 0.6.2, static projections are not honoured. Closes #837.

Version 0.6.2

Released on 14 March, 2016

- Fix: `Access-Control-Allow-Max-Age` should actually be `Access-Control-Max-Age`. Closes #829.
- Fix: `unique` validation rule is checked against soft deleted documents. Closes #831.
- Fix: Mongo does not allow `$` and `.` in field names. Apply this validation in schemas and dict fields. Closes #780.
- Fix: Remove "ensure uniqueness of (custom) id fields" feature. Addresses #788.
- Fix: `409 Conflict` not reported since upgrading to PyMongo 3. Closes #680.
- Fix: when a document is soft deleted, the OPLOG *_updated* field is not the time of the deletion but the time of the previous last update (Cyril Bonnard).
- Fix: TokenAuth. When the tokens are passed as "Authorization: " or "Authorization: Token " headers, werkzeug does not recognize them as valid authorization header, therefore the `request.authorization` field is empty (Luca Di Gaspero).
- Fix: `SCHEMA_ENDPOINT` does not work when schema has lambda function as `coerce` rule. Closes #790.
- Fix: CORS pre-flight requests malfunction on `SCHEMA_ENDPOINT` endpoint (Valerie Coffman).

- Fix: do not attempt to parse `number` values as strings when they are numerical (Nick Park).
- Fix: the `__init__.py` `ITEM_URL` does not match default_settings.py. Closes #786 (Ralph Smith).
- Fix: startup crash when both `SOFT_DELETE` and `ALLOW_UNKNOWN` are enabled. Closes #800.
- Fix: Serialize inside `of` and `of_type` rules new in Cerberus 0.9. Closes #692 (Arnau Orriols).
- Fix: In `put_internal` Validator is not set when `skip_validation` is `true` (Wei Guan).
- Fix: In `patch_internal` Validator is not set when `skip_validation` is `true` (Stratos Gerakakis).
- Fix: Add missing serializer for fields of type `number` (Arnau Orriols).
- Fix: Skip any null value from serialization (Arnau Orriols).
- Fix: When `SOFT_DELETE` is active an exclusive `datasource.projection` causes a `500` error. Closes #752.
- Update: PyMongo 3.2 is now required.
- Update: Flask-PyMongo 0.4+ is now required.
- Update: Werkzeug up to 0.11.4 is now required
- Change: simplejson v3.8.2 is now required.
- Docs: fix some typos (Manquer, Patrick Decat).
- Docs: add missing imports to authentication docs (Hamdy)
- Update license to 2016 (Prayag Verma)

Version 0.6.1

Released on 29 October, 2015

- New: `BULK_ENABLED` enables/disables bulk insert. Defaults to `True` (Julian Hille).
- New: `VALIDATE_FILTERS` enables/disables validating of query filters against resource schema. Closes #728 (Stratos Gerakakis).
- New: `TRANSPARENT_SCHEMA_RULES` enables/disables schema validation globally and `transparent_schema_rules` per resource (Florian Rathgeber).
- New: `ALLOW_OVERRIDE_HTTP_METHOD` enables/disables support for overriding request methods with `X-HTTP-Method-Override` headers (Julian Hille).
- Fix: flake8 fails on Python 3. Closes #747 (Simon Schönfeld).
- Fix: recursion for dotted field normalization (Matt Tucker).
- Fix: dependendencies on sub-document fields always return 422. Closes #706.
- Fix: invoking `post_internal` with `skpi_validation = True` causes a `422` response. Closes #726.
- Fix: explict inclusive datasource projection is ignored. Closes #722.
- Dev: fix rate limiting tests so they don't occasionally fail.
- Dev: make sure connections opened by test suite are properly closed on teardown.
- Dev: use middleware to parse overrides and eventually update request method (Julian Hille).
- Dev: optimize versioning by building specific versions without deepcopying the root document (Nick Park).
- Dev: `_client_projection` method has been moved up from the mongo layer to the base DataLayer class. It is now available for other data layers implementations, such as Eve-SQLAlchemy (Gonéri Le Bouder).

- Docs: add instructions for installing dependencies and building docs (Florian Rathgeber).
- Docs: fix link to contributing guidelines (Florian Rathgeber).
- Docs: fix some typos (Stratos Gerakakis, Julian Hille).
- Docs: add Eve-Swagger to Extensions page.
- Docs: fix broken link to Mongo's capped collections (Nathan Reynolds).

Version 0.6

Released on 28 September, 2015

- New: support for embedding simple ObjectId fields: you can now use the `data_relation` rule on them (Gonéri Le Bouder).
- New: support for multiple layers of embedding (Gonéri Le Bouder).
- New: `SCHEMA_ENDPOINT` allows resource schema to be returned from an API endpoint (Nick Park).
- New: HATEOAS links can be customized from within callback functions (Magdas Adrian).
- New: `_INFO`: string value to include an info section, with the given INFO name, at the Eve homepage (suggested value `_info`). The info section will include Eve server version and API version (API_VERSION, if set). `None` otherwise, if you do not want to expose any server info. Defaults to `None` (Stratos Gerakakis).
- New: `id_field` sets a field used to uniquely identify resource items within the database. Locally overrides `ID_FIELD` (Dominik Kellner).
- New: `UPSERT_ON_PUT` allows document creation on PUT if the document does not exist. Defaults to `True`. See below for details.
- New: PUT attempts to create a document if it does not exist. The URL endpoint will be used as `ID_FIELD` value (if `ID_FIELD` is included with the payload, it will be ignored). Normal validation rules apply. The response will be a `201 Created` on successful creation. Response payload will be identical the one you would get by performing a single document POST to the resource endpoint. Set `UPSET_ON_PUT` to `False` to disable this behaviour, and get a `404` instead. Closes #634.
- New: POST accepts documents which include `ID_FIELD` (`_id`) values. This is in addition to the old behaviour of auto-generating `ID_FIELD` values when the submitted document does not contain it. Please note that, while you can add `ID_FIELD` to the schema (previously not allowed), you don't really have to, unless its type is different from the `ObjectId` default. This means that in most cases you can start storing `ID_FIELD`-included documents right away, without making any changes.
- New: Log MongoDB and HTTP methods exceptions (Sebastien Estienne).
- New: Enhanced Logging.
- New: `VALIDATION_ERROR_AS_LIST`. If `True` even single field errors will be returned in a list. By default single field errors are returned as strings while multiple field errors are bundled in a list. If you want to standardize the field errors output, set this setting to `True` and you will always get a list of field issues. Defaults to `False`. Closes #536.
- New: `STANDARD_ERRORS` is a list of HTTP codes that will be served with the canonical API response format, which includes a JSON body providing both error code and description. Addresses #586.
- New: `anyof` validation rule allows you to list multiple sets of rules to validate against.
- New: `alloff` validation rule, same as `anyof` except that all rule collections in the list must validate.
- New: `noneof` validation rule. Same as `anyof` except that it requires no rule collections in the list to validate.
- New: `oneof` validation rule. Same as `anyof` except that only one rule collections in the list can validate.

- New: `valueschema` validation rules replaces the now deprecated `keyschema` rule.
- New: `propertyschema` is the counterpart to `valueschema` that validates the keys of a dict.
- New: `coerce` validation rule. Type coercion allows you to apply a callable to a value before any other validators run.
- New: `MONGO_AUTHDBNAME` allows to specify a MongoDB authorization database. Defaults to `None` (David Wood).
- New: `remove` method in Mongo data layer now returns the deletion status or `None` if write acknowledgement is disabled (Mayur Dhamanwala).
- New: `unique_to_user` validation rule allows to validate that a field value is unique to the user. Different users can share the same value for the field. This is useful when User Restricted Resource Access is enabled on an endpoint. If URRA is not active on the endpoint, this rule behaves like `unique`. Closes #646.
- New: `MEDIA_BASE_URL` allows to set a custom base URL to be used when `RETURN_MEDIA_AS_URL` is active (Henrique Barroso).
- New: `SOFT_DELETE` enables soft deletes when set to `True` (Nick Park.)
- New: `mongo_indexes` allows for creation of MongoDB indexes at application launch (Pau Freixes.)
- New: clients can opt out of default embedded fields: `?embedded={"author":0}` would cause the embedded author not to be included with response payload. (Tobias Betz.)
- New: CORS: Support for `X-ALLOW-CREDENTIALS` (Cyprien Pannier.)
- New: Support for dot notation in POST, PATCH and PUT methods. Be aware that, for PATCH and PUT, if dot notation is used even on just one field, the whole sub-document will be replaced. So if this document is stored:

  ```
  {"name": "john", "location": {"city": "New York", "address": "address"}}
  ```

  A PATCH like this:

  ```
  {"location.city": "Boston"}
  ```

  (which is exactly equivalent to:)

  ```
  {"location": {"city": "a nested city"}}
  ```

  Will update the document to:

  ```
  {"name": "john", "location": {"city": "Boston"}}
  ```

- New: JSONP Support (Tim Jacobi.)
- New: Support for multiple MongoDB databases and/or servers.
  - `mongo_prefix` resource setting allows overriding of the default `MONGO` prefix used when retrieving MongoDB settings from configuration. For example, set a resource `mongo_prefix` to `MONGO2` to read/write from the database configured with that prefix in your settings file (`MONGO2_HOST`, `MONGO2_DBNAME`, etc.)
  - `set_mongo_prefix()` and `get_mongo_prefix()` have been added to `BasicAuth` class and derivates. These can be used to arbitrarily set the target database depending on the token/client performing the request.

  Database connections are cached in order to not to loose performance. Also, this change only affects the MongoDB engine, so extensions currently targetting other databases should not need updates (they will not inherit this feature however.)
- New: Enable `on_pre_GET` hook for HEAD requests (Daniel Lytkin.).
- New: Add `X-Total-Count` header for collection GET/HEAD requests (Daniel Lytkin.).
- New: `RETURN_MEDIA_AS_URL`, `MEDIA_ENDPOINT` and `MEDIA_URL` allow for serving files at a dedicated media endpoint while urls are returned in document media fields (Daniel Lytkin.)

- New: `etag_ignore_fields`. Resource setting with a list of fields belonging to the schema that won't be used to compute the ETag value. Defaults to `None` (Olivier Carrère.)
- Change: when HATEOAS is off the home endpoint will respond with `200 OK` instead of `404 Not Found` (Stratos Gerakakis).
- Change: PUT does not return `404` if a document URL does not exist. It will attempt to create the document instead. Set `UPSET_ON_PUT` to `False` to disable this behaviour and get a `404` instead.
- Change: A PATCH including an `ID_FIELD` field which value is different than the original will get a `400 Bad Request`, along with an explanation in the message body that the field is immutable. Previously, it would get an `unknown field` validation error.
- Dev: Improve GET perfomance on large versioned documents (Nick Park.)
- Dev: The `MediaStorage` base class now accepts the active resource as an argument for its methods. This allows data-layers to avoid resorting to the Flask request object to determine the active resource. To preserve backward compatibility the new `resource` argument defaults to `None` (Magdas Adrian).
- Dev: The Mongo data-layer is not dependant on the Flask request object anymore. It will still fallback to it if the `resource` argument is `None`. Closes #632. (Magdas Adrian).
- Fix: store versions in the same mongo collection when `datasource` is used (Magdas Adrian).
- Fix: Update `serialize` to gracefully handle non-dictionary values in dict type fields (Nick Park).
- Fix: changes to the `updates` argument, applied by callbacks hooked to the `on_updated` event, were not persisted to the database (Magdas Adrian). Closes #682.
- Fix: Changes applied to the `updates` argument``on_updated`` returns the whole updated document. Previously, it was only returning the updates sent with the request. Closes #682.
- Fix: Replace the Cerberus rule `keyschema`, now deprecated, with the new `propertyschema` (Julian Hille).
- Fix: some error message are not filtered out of debug mode anymore, as they are useful for users and do not leak information. Closes #671 (Sebastien Estienne).
- Fix: reinforce Content-Type Header handling to avoid possible crash when it is missing (Sebastien Estienne).
- Fix: some schema errors were not being reported as SchemaError exceptions. A more generic 'DOMAIN missing or wrong' message was returned instead.
- Fix: When versioning is enabled on a resource with a custom ID_FIELD, versioning documents will inherit their ID from the versioned document, making any update of the document result in a DuplicateKeyError (Matthieu Prat).
- Fix: Filter validation fails to validate query selectors that contain a value of the list data-type, which is not a list of sub-queries. See #674 (Matthieu Prat).
- Fix: `_validate_dependencies` always returns `None`.
- Fix: `412 Precondition Failed` does not return a JSON body. Closes #661.
- Fix: `embedded_fields` may point on a field that come from another embedded document. For example, `['a.b.c', 'a.b', 'a']` (Gonéri Le Bouder).
- Fix: add handling of sub-resource resolving for PUT method (Olivier Poitrey).
- Fix: `dependencies` rule would mistakenly validate documents when target fields happened to also have a `default` value.
- Fix: According to RFC2617 the separator should be (=) instead of (:). This caused at least Chrome not to prompt user for the credentials, and not to send the Authorization header even when credentials were in the url (Samuli Tuomola).

- Fix: make sure `unique` validation rule is consistent between HTTP methods. A field value must be unique within the datasource, regardless of the user who created it. Closes #646.
- Fix: OpLog domain entry is not created if `OPLOG_ENDPOINT` is `None`. Closes #628.
- Fix: Do not overwrite `ID_FIELD` as it is not a sub resource. See #641 for details (Olivier Poitrey).
- Fix: ETag computation crash when non-standard json serializers are used (Kevin Roy.)
- Fix: Remove duplicate item in Mongo operators list. Closes #619.
- Fix: Versioning: invalidate cache when `_latest_version` changes in versioned doc (Nick Park.)
- Fix: snippet in account management tutorial (xgddsg.)
- Fix: `MONGO_REPLICA_SET` and other significant Flask-PyMongo settings have been added to the documentation. Closes #615.
- Fix: Serialization of lists of lists (Nick Park.)
- Fix: Make sure `original` is not modified during `PATCH`. Closes #611 (Petr Jašek.)
- Fix: Route parameters are applied to new documents before they are validated. This ensures that documents with required fields will be populated before they are validated. Addresses #354. (Matthew Ellison.)
- Fix: `GridFSMediaStorage` does not save filename. Closes #605 (Sam Luu).
- Fix: Reinforce GeoJSON validation (Joakim Uddholm.)
- Fix: Geopoint coordinates do not accept integers. Closes #591 (Joakim Uddholm.)
- Fix: OpLog enabled makes PUT return wrong Etag. Closes #590.
- Update: Cerberus 0.9.2 is now required.
- Update: PyMongo 2.8 is now required (which in turn supports MongoDB 3.0)

Version 0.5.3

Released on 17 March, 2015.

- Fix: Support for Cerberus 0.8.1.
- Fix: Don't block on first field serialization exception. Closes #568.
- Fix: Ignore read-only fields in `PUT` requests when their values aren't changed compared to the stored document (Bjorn Andersson.)
- Docs: replace `file` with `media` type. Closes #566.

Version 0.5.2

Released on 23 Feb, 2015. Codename: 'Giulia'.

- Fix: hardening of database concurrency checks. See #561 (Olivier Carrère.)
- Fix: `PATCH` and `PUT` do not include Etag header (Marcus Cobden.)
- Fix: endpoint-level authentication crash when a callable is passed. Closes #558.
- Fix: serialization of `keyschema` fields with `objetid` values. Closes #525.
- Fix: typos in schema rules might lead to arbitrary payloads being validated (Emmanuel Leblond.)
- Fix: ObjectId value in ID field of type string (Jaroslav Semančík.)

- Fix: User Restricted Resource Access does not work with HMAC Auth classes.
- Fix: Crash when `embedded` is used on subdocument with a missing field (Emmanuel Leblond.)
- Docs: add `MONGO_URI` as an alternative to other MongoDB connection options. Closes #551.
- Change: Werkzeug 0.10.1 is now required.
- Change: `DataLayer` API methods `update()` and `replace()` have a new `original` argument.

Version 0.5.1

Released on 16 Jan, 2015.

- Fix: dependencies with value checking seem broken (#547.)
- Fix: documentation typo (Marc Abramowitz.)
- Fix: pretty url for regex with a colon in the expression (Magdas Adrian.)

Version 0.5

Released on 12 Jan, 2015.

- New: Operations Log (http://python-eve.org/features#operations-log.)
- New: GeoJSON (http://python-eve.org/features.html#geojson) (Juan Madurga.)
- New: Internal Resources (http://python-eve.org/features#internal-resources) (Magdas Adrian.)
- New: Support for multiple origins when using CORS (Josh Villbrandt, #532.)
- New: Regexes are stripped out of HATEOAS urls when present. You now get `games/<game_id>/images` where previously you would get `games/<regex('[a-f0-9]{24}'):game_id>/images`). Closes #466.
- New: `JSON_SORT_KEYS` enables JSON key sorting (Matt Creenan).
- New: Add the current query string to the self link for responses with multiple documents. Closes #464 (Jen Montes).
- New: When document versioning is on, add `?version=<version_num>` to HATEOAS self links. Also adds pagination links for `?version=all` and `?version=diffs` requests when the number exceeds the max results. Partially addresses #475 (Jen Montes).
- New: `QUERY_WHERE` allows to set the query parameter key for filters. Defaults to `where`.
- New: `QUERY_SORT` allows to set the query parameter key for sorting. Defaults to `sort`.
- New: `QUERY_PAGE` allows to set the query parameter key for pagination. Defaults to `page`.
- New: `QUERY_PROJECTION` allows to set the query parameter key for projections. Defaults to `projection`.
- New: `QUERY_MAX_RESULTS` allows to set the query parameter key for max results. Defaults to `max_results`.
- New: `QUERY_EMBEDDED` allows to set the query parameter key embedded documents. Defaults to `embedded`.
- New: Fire `on_fetched` events for `version=all` requests (Jen Montes).
- New: Support for CORS `Access-Control-Expose-Headers` (Christian Henke).
- New: `post_internal()` can be used for intenral post calls. This method is not rate limited, authentication is not checked and pre-request events are not raised (Magdas Adrian).
- New: `put_internal()` can be used for intenral PUT calls. This method is not rate limited, authentication is not checked and pre-request events are not raised (Kevin Funk).

- New: `patch_internal()` can be used for intenral PATCH calls. This method is not rate limited, authentication is not checked and pre-request events are not raised (Kevin Funk).
- New: `delete_internal()` can be used for intenral DELETE calls. This method is not rate limited, authentication is not checked and pre-request events are not raised (Kevin Funk).
- New: Add an option to `_internal` methods to skip payload validation (Olivier Poitrey).
- New: Comma delimited sort syntax in queries. The MongoDB data layer now also supports queries like `?sort=lastname,-age`. Addresses #443.
- New: Add extra 4xx response codes for proper handling. Only `405` Method not allowed, `406` Not acceptable, `409` Conflict, and `410` Gone have been added to the list (Kurt Doherty).
- New: Add serializers for integer and float types (Grisha K.)
- New: dev-requirements.txt added to the repo.
- New: Embedding of documents by references located in any subdocuments. For example, query `embedded={"user.friends":1}` will return a document with "user" and all his "friends" embedded, but only if `user` is a subdocument and `friends` is a list of references (Dmitry Anoshin).
- New: Allow mongoengine to work properly with cursor counts (Johan Bloemberg)
- New: `ALLOW_UNKNOWN` allows unknown fields to be read, not only written as before. Closes #397 and #250.
- New: `VALIDATION_ERROR_STATUS` allows setting of the HTTP status code to use for validation errors. Defaults to `422` (Olivier Poitrey).
- New: Support for sub-document projections. Fixes #182 (Olivier Poitrey).
- New: Return `409 Conflict` on pymongo `DuplicateKeyError` for `POST` requests, as already happens with `PUT` requests (Matt Creenan, #537.)
- Change: `DELETE` returns `204 NoContent` on a successful delete.
- Change: SERVER_NAME removed as it is not needed anymore.
- Change: URL_PROTOCOL removed as it is not needed anymore.
- Change: HATEOAS links are now relative to the API root. Closes #398 #401.
- Change: If-Modified-Since has been disabled on resource (collections) endpoints. Same functionality is available with a `?where={"_udpated": {"$gt": "<RFC1123 date>"}}` request. The OpLog also allows retrieving detailed changes happened at any endpoint, deleted documents included. Closes #334.
- Change: etags are now persisted with the documents. This ensures that etags are consistent across queries, even when projection queries are issued. Please note that etags will only be stored along with new documents created and/or edited via API methods (POST/PUT/PATCH). Documents inserted by other means and those stored with v0.4 and below will keep working as previously: their etags will be computed on-the-fly and you will get still be getting inconsistent etags when projection queries are issued. Closes #369.
- Change: XML item, meta and link nodes are now ordered. Closes #441.
- Change: `put` method signature for `MediaStorage` base class has been updated. `filemame` is now optional. Closes #414.
- Change: CORS behavior to be compatible with browsers (Chrome). Eve is now echoing back the contents of the Origin header if said content is whitelisted in X_DOMAINS. This also safer as it avoids exposing internal server configuration. Closes #408. This commit was carefully handcrafed on a flight to EuroPython 2014.
- Change: Specify a range of dependant package versions. #379 (James Stewart).
- Change: Cerberus 0.8 is now required.
- Change: pymongo v2.7.2 is now required.

- Change: simplejson v3.6.5 is now required.
- Change: update `dev-requirements.txt` to most recent tools available.
- Fix: add `README.rst` to `MANIFEST.in` (Niall Donegan.)
- Fix: `LICENSE` variable in `setup.py` should be "shortstring". Closes #540 (Niall Donegan.)
- Fix: `PATCH` on fields with original value of `None` (Marcus Cobden, #534).
- Fix: Fix impossible version ranges in setup.py (Marcus Cobden, #531.)
- Fix: Bug with expanding lists of roles, compromising authorization (Mikael Berg, #527)
- Fix: `PATCH` on subdocument fields does not overwrite the whole subdocument anymore. Closes #519.
- Fix: Added support for validation on field attribute with type list (Jorge Morales).
- Fix: Fix a serialization bug with integer and float when value is 0 (Olivier Poitrey).
- Fix: Custom ID fields tutorial: if custom ID fields are being used, then MongoDB/Eve won't be able to create them automatically as it does with the *ObjectId* default type. Closes #511.
- Fix: Dependencies with default values were reported as missing if omitted. Closes #353.
- Fix: Dependencies always fails on PATCH if dependent field isn't part of the update. #363.
- Fix: client projections work when `allow_unknown` is active. Closes #497.
- Fix: datasource projections are active when `allow_unknown` is active. closes #497.
- Fix: Properly serialize nullable floats and integers. Closes #469.
- Fix: `_mongotize()` turns non-ObjectId strings (but not unicode) into ObjectIds. Closes #508 (Or Neeman).
- Fix: Fix validation of read-only fields inside dicts. Closes #474 (Arnau Orriols).
- Fix: Parent and collection links follow the scheme described in #475 (Jen Montes).
- Fix: Ignore read-only fields in `PATCH` requests when their values aren't changed compared to the stored document. Closes #479.
- Fix: Allow `EVE_SETTINGS` envvar to be used exclusively. Previously, a settings file in the working directory was always required. Closes #461.
- Fix: exception when trying to set nullable media field to null (Daniel Lytkin)
- Fix: Add missing `$options` and `$list` MongoDB operators to the allowed list (Jaroslav Semančík).
- Fix: Get document when it is missing embedded media. In case you try to embedd a document which has media fields and that document has been deleted, you would get an error (Petr Jašek).
- Fix: fix additional lookup regex in RESTful Account Management tutorial (Ashley Roach).
- Fix: `utils.weak_date` always returns a RFC-1123 date (Petr Jašek).
- Fix: Can't embed a ressource with a custom _id (non ObjectId). Closes #427.
- Fix: Do not follow DATE_FORMAT for HTTP headers. Closes #429 (Olivier Poitrey).
- Fix: Fix app initialization with resource level versioning #409 (Sebastián Magrí).
- Fix: KeyError when trying to use embedding on a field that is missing from document. It was fixed earlier in #319, but came back again after new embedding mechanism (Daniel Lytkin).
- Fix: Support for list of strings as default value for fields (hansotronic).
- Fix: Media fields are now properly returned even in embedded documents. Closes #305.
- Fix: auth in domain configuration can be either a callable or a class instance (Gino Zhang).

- Fix: Schema definition: a default value of [] for a list causes IndexError. Closes #417.
- Fix: Close file handles in setup.py (Harro van der Klauw)
- Fix: Querying a collection should always return pagination information (even when no data is being returned). Closes #415.
- Fix: Recursively validate the whole query string.
- Fix: If the data layer supports a list of allowed query operators, take them into consideration when validating a query string. Closes #388.
- Fix: Abort with 400 if unsupported query operators are used. Closes #387.
- Fix: Return the error if a blacklisted MongoDB operator is used in a query (debug mode).
- Fix: Invalid sort syntax raises 500 instead of 400. Addresses #378.
- Fix: Fix serialization when *type* is missing in schema. #404 (Jaroslav Semančík).
- Fix: When PUTting or PATCHing media fields, they would not be properly replaced as needed (Stanislav Heller).
- Fix: `test_get_sort_disabled` occasional failure.
- Fix: A POST with an empty array leads to a server crash. Now returns a 400 error isntead and ensure the server won't crash in case of mongo invalid operations (Olivier Poitrey).
- Fix: PATCH and PUT don't respect flask.abort() in a pre-update event. Closes #395 (Christopher Larsen).
- Fix: Validating keyschema rules would cause a TypeError since 0.4. Closes pyeve/cerberus#48.
- Fix: Crash if client projection is not a dict #390 (Olivier Poitrey).
- Fix: Server crash in case of invalid "where" syntax #386 (Olivier Poitrey).

Version 0.4

Released on 20 June, 2014.

- [new] You can now start the app without any resource defined and use `app.register_resource` later as needed (Petr Jašek).
- [new] Data layer is now usable outside request context, for example within a Celery task where there's no request context (Petr Jašek).
- [new][change] Add pagination info to get results whatever the HATEOAS status. Closes #355 (Olivier Poitrey).
- [new] Ensure all errors return a parseable body (JSON or XML). Closes #365 (Olivier Poitrey).
- [new] Apply sub-request route's params to the created document if matching the schema, e.g. a POST on `/people/1234.../invoices` will set the `contact_id` field to 1234... so created invoice is automatically associated with the parent resource (Olivier Poitrey).
- [new] Allow some more HTTP errors (403 and 404) to be thrown from db hooks (Olivier Poitrey).
- [new] `ALLOWED_READ_ROLES`. A list of allowed *roles* for resource endpoints with GET and OPTIONS methods (Olivier Poitrey).
- [new] `ALLOWED_WRITE_ROLES`. A list of allowed *roles* for resource endpoints with POST, PUT and DELETE methods (Olivier Poitrey).
- [new] `ALLOWED_ITEM_READ_ROLES`. A list of allowed *roles* for item endpoints with GET and OPTIONS methods (Olivier Poitrey).
- [new] `ALLOWED_ITEM_WRITE_ROLES`. A list of allowed *roles* for item endpoints with PUT, PATCH and DELETE methods (Olivier Poitrey).

- [new] 'dependencies' validation rule.
- [new] 'keyschema' validation rule.
- [new] 'regex' validation rule.
- [new] 'set' as a core data type.
- [new] 'min' and 'max' now apply to floats and numbers too.
- [new] File Storage. `EXTENDED_MEDIA_INFO` allows a list of meta fields (file properties) to forward from the file upload driver (Ben Demaree).
- [new] Python 3.4 is now supported.
- [new] Support for default values in documents with more than one level of data (Javier Gonel).
- [new] Ability to send entire document in write responses. `BANDWITH_SAVER` aka Coherence Mode (Josh Villbrandt).
- [new] `on_pre_<METHOD>` events expose the *lookup* dictionary which allows for setting up dynamic database lookups on both resource and item endpoints.
- [new] Return a 400 response on pymongo DuplicateKeyError, with exception message if debug mode is on (boosh).
- [new] PyPy officially supported and tested (Javier Gonel).
- [new] tox support (Javier Gonel).
- [new] Post database events (Javier Gonel). Addresses #272.
- [new] Versioned Documents (Josh Villbrandt). Closes #224.
- [new] Python trove classifiers added to setup.py.
- [new] Client projections are also honored at item endpoints.
- [new] validate that ID_FIELD is not set as a resource `auth_field`. Addresses #266.
- [new] `URL_PROTOCOL` defines the HTTP protocol used when building HATEOAS links. Defaults to `''` for relative paths (Junior Vidotti).
- [new] `on_delete_item` and `on_deleted_item` is raised on DELETE requests sent to document endpoints. Addresses #232.
- [new] `on_delete_resource` and `on_deleted_resource` is raised on DELETE requests sent to resource endpoints. Addresses #232.
- [new] `on_update` is raised on PATCH requests, when a document is about to be updated on the database. Addresses #232.
- [new] `on_replace` is raised on PUT requests, when a document is about to be replaced on the database. Addresses #232.
- [new] `auth` constructor argument accepts either a class instance or a callable. Closes #248.
- [change] Cerberus 0.7.2 is now required.
- [change] Jinja2 2.7.3 is now required.
- [change] Werkzeug 0.9.6 is now required.
- [change] simplejson 3.5.2 is now required.
- [change] itsdangerous 0.24 is now required. Addresses #378.
- [change] Events 0.2.1 is now required.

- [change] MarkupSafe 0.23 is now required.
- [change] For bulk and non-bulk inserts, response status now always either 201 when everything was ok or 400 when something went wrong. For bulk inserts, if at least one document doesn't validate, the whole request is rejected, and none of the documents are inserted into the database. Additionnaly, this commit adopts the same response format as collections: responses are always a dict with a `_status` field at its root and an eventual `_error` object if `_status` is `ERR` to comply with #366. Documents status are stored in the `_items` field (Olivier Poitrey).
- [change] Callbacks get whole json response on `on_fetched`. This allows for callbacks functions to alter the whole payload, even when HATEOAS is enabled and `_items` and `_links` metafields are present.
- [change] `on_insert` is not raised anymore on PUT requests (replaced by above mentioned `on_replace`).
- [change] `auth.request_auth_value` is no more. Yay. See below.
- [change] `auth.set_request_auth_value()` allows to set the `auth_field` value for the current request.
- [change] `auth.get_request_auth_value()` allows to retrieve the `auth_field` value for the current request.
- [change] `on_update(ed)` and `on_replace(ed)` callbacks now receive both the original document and the updates (Jaroslav Semančík).
- [change] Review event names (Javier Gonel).
- [fix] return 500 instead of 404 if CORS is enabled. Closes #381.
- [fix] Crash on GET requests on resource endpoints when ID_FIELD is missing on one or more documents. Closes #351.
- [fix] Cannot change a nullable objectid type field to contain null. Closes #341.
- [fix] HATEOAS links as business unit values even when regexes are configured for the endpoint.
- [fix] Documentation improvements (Jen Montes).
- [fix] KeyError exception was raised when field specified in schema as embeddable was missing in a particular document (Jaroslav Semančík).
- [fix] Tests on HEAD requests would very occasionally fail. See #316.
- [change] PyMongo 2.7.1 is now required.
- [fix] Automatic fields such as `DATE_CREATD` and `DATE_CREATED` are correctly handled in client projections (Josh Villbrandt). Closes #282.
- [fix] Make codebase compliant with latest PEP8/flake8 release (Javier Gonel).
- [fix] If you had a media field, and set datasource projection to 0 for that field, the media would not be deleted. Closes #284.
- [fix] tests cleanup (Javier Gonel).
- [fix] tests now run on any system without needing to set `ulimit` to a higher value (Javier Gonel).
- [fix] media files: don't try to delete a field that does not exist (Taylor Brown).
- [fix] Occasional KeyError while building `_media` helper dict. See #271 (Alexander Hendorf).
- [fix] `If-Modified-Since` misbehaviour when a datasource filter is set. Closes #258.
- [fix] Trouble serializing list of dicts. Closes #265 and #244.
- [fix] `HATEOAS` item links are now coherent actual endpoint URL even when natural immutable keys are used in URLs (Junior Vidotti). Closes #256.

- [fix] Replaced `ID_FIELD` by `item_lookup_field` on self link. item_lookup_field will default to `ID_FIELD` if blank.

Version 0.3

Released on 14 February, 2014.

- [fix] Serialization of sub-documents (Hannes Tiede). Closes #244.
- [new] `X_MAX_AGE` allows to configure CORS Access-Control-Max-Age (David Buchmann).
- [fix] `GET` with `If-Modified-Since` on list endpoint returns incorrect 304 if resource is empty. Closes #243.
- [change] `POST` will return `201 Created` if at least one document was accepted for insertion; `200 OK` otherwise (meaning the request was accepted and processed). It is still client's responsability to parse the response payload to check if any document did not pass validation. Addresses #201 #202 #215.
- [new] `number` data type. Allows both integers and floats as field values.
- [fix] Using primary keys other than _id. Closes #237.
- [fix] Add tests for `PUT` when User Restricted Resource Access is active.
- [fix] Auth field not set if resource level authentication is set. Fixes #231.
- [fix] RateLimit check was occasionally failing and returning a 429 (John Deng).
- [change] Jinja2 2.7.2 is now required.
- [new] media files (images, pdf, etc.) can be uploaded as `media` document fields. When a document is requested, eventual media files will be returned as Base64 strings. Upload is done via `POST`, `PUT` and `PATCH` using the `multipart/form-data` content-type. For optmized performance, by default files are stored in GridFS, however custom `MediaStorage` classes can be provided to support alternative storage systems. Clients and API maintainers can exploit the projections feature to include/exclude media fields from requests. For example, a request like `/url/<id>?projection={"image": 0}` will return the document without the image field. Also, while setting a resource `datasource` it is possible to explicitly exclude media fields from standard responses (clients will need to explicitly add them to the payload with `?projection={"image": 1}`).
- [new] `media` type for schema fields.
- [new] `media` application argument. Allows to specify a media storage class to be used to store media files. Defaults to `GridFSMediaStorage`.
- [new] `GridFSMediaStorage` class. Stores files into GridFS.
- [new] `MediaStorage` class provides a standardized API for storing files, along with a set of default behaviors that all other storage systems can inherit or override as necessary.
- [new] `file` data type support and validation for resource schema.
- [new] `multipart/form-data` content-type is now supported for requests.
- [fix] Field exclusion (`?projection={"fieldname": 0}`) now supported in client projections. Remember, mixing field inclusion and exclusion is still not supported by MongoDB.
- [fix] `URL_PREFIX` and `API_VERSION` are correctly reported in HATOEAS links.
- [fix] `DELETE` on sub-resources should only delete documents referenced by the parent. Closes #212.
- [fix] `DELETE` on a resource endpoint honors User-Restricted Resource Access. Closes #213.
- [new] `JSON` allows to enable/disable JSON responses. Defaults to `True` (JSON enabled).
- [new] `XML` allows to enable/disable XML responses. Defaults to `True` (XML enabled).

- [fix] XML properly honors `_LINKS` and `_ITEMS` settings.
- [fix] return all document fields when resource schema is empty.
- [new] pytest.ini for pytest support.
- [fix] All tests should now run with nose and pytest. Closes #209.
- [new] `query_objectid_as_string` resource setting. Defaults to `False`. Addresses #207.
- [new] `ETAG` allows to customize the etag field. Defaults to `_etag`.
- [change] `etag` is now `_etag` in all default response payloads (see above).
- [change] `STATUS` defaults to '_status'.
- [change] `ISSUES` defaults to '_issues'.
- [change] `DATE_CREATED` defaults to '_created'. Upgrade existing collections by running `db.<collection>.update({}, { $rename: { "created": "_created" } }, { multi: true })` in the mongo shell. If an index exists on the field, drop it and create a new one using the new field name.
- [change] `LAST_UPDATED` defaults to '_updated'. Upgrade existing collections by running `db.<collection>.update({}, { $rename: { "updated": "_updated" } }, { multi: true })` in the mongo shell. If an index exists on the field, drop it and create a new one usung the new field name.
- [change] Exclude `etag` from both response payload and headers if concurrency control is disabled (`IF_MATCH = False`). Closes #205.
- [fix] Custom `ID_FIELD` would fail on update/insert methods. Fixes #203 (Jaroslav Semančík).
- [change] GET: when If-Modified-Since header is present, either no documents (304) or all documents (200) are sent per the HTTP spec. Original behavior can be achieved with: `/resource?where={"updated":{"$gt":"if-modified-since-date"}}` (Josh Villbrandt).
- [change] Validation errors are now reported as a dictionary with offending fields as keys and issues descriptions as values.
- [change] Cerberus v0.6 is now required.

Version 0.2

Released on 30 November, 2013.

- [new] Sub-Resources. It is now possible to configure endpoints such as: `/companies/<company_id>/invoices`. Also, the corresponding item endpoints, such as `/companies/<company_id>/invoices/<invoice_id>`, are available. All CRUD operations on these endpoints are allowed. Closes 156.
- [new] `resource_title` allows to customize the endpoint title (HATEOAS).
- [new][dev] `extra` cursor property, when present, will be added to `GET` responses (with same key). This feature can be used by Eve extensions to inject proprietary data into the response stream (Petr Jašek).
- [new] `IF_MATCH` allows to disable checks for ETag matches on edit, replace and delete requests. If disabled, requests without an If-Match header will be honored without returning a 403 error. Defaults to True (enabled by default).
- [new] `LINKS` allows to customize the links field. Default to '_links'.
- [new] `ITEMS` allows to customize the items field. Default to '_items'.
- [new] `STATUS` allows to customize the status field. Default to 'status'.
- [new] `ISSUES` allows to customize the issues field. Default to 'issues'.

- [new] Handling custom ID fields tutorial.
- [new] A new `json_encoder` initialization argument is available. It allows to pass custom JSONEncoder or eve.io.BaseJSONEncoder to the Eve instance.
- [new] A new `url_converters` initialization argument is available. It allows to pass custom Flask url converters to the Eve constructor.
- [new] ID_FIELD fields can now be of arbitrary types, not only ObjectIds. Thanks to Kelvin Hammond for contributing to this one. Closes #136.
- [new] `pre_<method>` and `pre_<method>_<resource>` event hooks are now available. They are raised when a request is received and before processing it. The resource involved and the Flask request object are returned to the callback function (dccrazyboy).
- [new] `embedded_fields` activates default Embedded Resource Serialization on a list of selected document fields. Eventual embedding requests by clients will be processed along with default embedding. In order for default embedding to work, the field must be defined as embeddable, and embedding must be active for the resource (with help from Christoph Witzany).
- [new] `default_sort` option added to the `datasource` resource setting. It allows to set default sorting for the endpoint. Default sorting will be overriden by a client request that happens to include a `?sort` argument within the query string (with help from Christoph Witzany).
- [new] You can now choose to provide custom settings as a Python dictionary.
- [new] New method `Eve.register_resource()` for registering new resource after initialization of Eve object. This is needed for simpler initialization API of all ORM/ODM extensions (Stanislav Heller).
- [change] Rely on Flask endpoints to map urls to resources.
- [change] For better consistency with new `pre_<method>` hooks, `on_<method>` event hooks have been renamed to `on_post_<method>`.
- [change] Custom authentication classes can now be set at endpoint level. When set, an endpoint-level auth class will override the eventual global level auth class. Authentication docs have been updated (and greatly revised) accordingly. Closes #89.
- [change] JSON encoding is now handled at the DataLayer level allowing for specialized, granular, data-aware encoding. Also, since the JSON encoder is now a class attribute, extensions can replace the pre-defined data layer encoder with their own implementation. Closes #102.
- [fix] HMAC example and docs updated to align with new hmac in Python 2.7.3, which is only accepting bytes string. Closes #199.
- [fix] Properly escape leaf values in XML responses (Florian Rathgeber).
- [fix] A read-only field with a default value would trigger a validation error on POST and PUT methods.

Version 0.1.1

Released on October 31th, 2013.

- DELETE now uses the original document ID_FIELD when issuing the delete command to the underlying data layer (Xavi Cubillas).
- Embedded Resource Serialization also available at item endpoints (`/invoices/<id>/?embedded={'person':1}`),
- `collection` (used when setting up a data relation, see Embedded Resource Serialization) has been renamed to `resource` in order to avoid confusion between the Eve schema and underlying MongoDB collections.

- Nested endpoints. Endpoints with deep paths like `/contacts/overseas` can now function in conjuction with top-level endpoints (`/contacts`). Endpoints are completely independent: each can allow item lookups (`/contacts/<id>` and `contacts/overseas/<id>`) and different access methods. Previously, while you could have complex urls, you could not get nested endpoints to work properly.
- PyMongo 2.6.3 is now supported.
- item-id wrappers have been removed from POST/PATCH/PUT requests and responses. Requests for single document insertion/edition are now performed by just submitting the relevant document. Bulk insert requests are performed by submitting a list of documents. The response to bulk requests is a list itself in which every list item contains the state of the corresponding request document. Please note that this is a breaking change. Also be aware that when the request content-type is `x-www-form-urlencoded`, single document insert is performed. Closes #139.
- ObjectId are properly serialized on POST/PATCH/PUT methods.
- Queries on ObjectId and datetime values in nested documents.
- `auth.user_id` renamed to `auth.request_auth_value` for better consistency with the `auth_field` setting. Closes #132 (Ryan Shea).
- Same behavior as Flask, SERVER_NAME now defaults to None. It allows much easier development on distant machine that may changes IP (Ronan Delacroix).
- CORS support was not available for `additional_lookup` urls (Petr Jašek.)
- 'default' field values that could be assimilated to `None` (0, None, "") would be ignored.
- POST and PUT would fail with 400 if there was no auth class while `auth_field` was set for a resource.
- Fix order of string arguments in exception message in flaskapp.validate_schema() (Roy Smith).

Version 0.1

Released on September 30th, 2013.

- `PUT` method for completely replace a document while keeping the same unique identifier. Closes #96.
- Embedded Resource Serialization. If a document field is referencing a document in another resource, clients can request the referenced document to be embedded within the requested document (Bryan Cattle). Closes #68.
- "No trailing slash" URLs are now supported. Closes #118.
- HATEOAS is now optional and can be disabled both at global and resource level.
- `X-HTTP-Method-Override` supported for all HTTP Methods. Closes #95.
- HTTP method is now passed into `authenticate()` and `check_auth()` (Ken Carpenter). Closes #90 .
- Cleanup and hardening of User-Restricted Resource Access Edit (Bryan Cattle).
- Account Management tutorial updated to reflect the event hooks naming update introduced in v0.0.9.
- Some more Python 3 refactoring (Dong Wei Ming).
- Events 0.2.0 is now supported.
- PyMongo 2.6.2 is now supported.
- Cerberus 0.4.0 is now supported.
- Item `GET` on documents with non-existent 'created' field (because stored outside of API context) were not returning a default value for the field.

- Edits on documents with non-existent 'created' or 'updated' fields (because stored outside of the API context) were returning `412 Precondition Failed`. Closes #123.
- `on_insert` is raised when a `PUT` (replace action) is about to be performed. Closes #120.
- Installation on Windows with Python 3 was returning encoding errors.
- Fixed #99: malformed XML render when href includes forbidden URI/URL chars.
- Fixed a bug introduced with 0.0.9 and Python 3 support. Filters (`?where`) on datetime values were not working when running on Python 2.x.
- Fixed some typos and minor grammatical errors all across the documentation (Ken Carpenter, Jean Boussier, Kracekumar, Francisco Corrales Morales).

Version 0.0.9

Released on August 29, 2013

- PyMongo 2.6 is now supported.
- `FILTERS` boolean replaced by `ALLOWED_FILTERS` list which allows for explicit whitelisting of filter-enabled fields (Bryan Cattle). Closes #78.
- Custom user ids for User-Restricted Resource Access, allowing for more flexibility and token revocation with token-based authentication. Closes #73.
- `AUTH_USERNAME_FIELD` renamed to `AUTH_FIELD`.
- `auth_username_field` renamed to `auth_field`.
- BasicAuth and subclasses now support `user_id` property.
- Updated the event hooks naming system to be more robuts and consistent. Closes #80.
- To emphasize the fact that they are tied to a method, all `on_<method>` hooks now have `<method>` in uppercase.
- `on_getting` hook renamed to `on_fetch_resource`.
- `on_getting_<resource>` hook renamed to `on_fetch_resource_<resource>`
- `on_getting_item` hook renamed to `on_fetch_item`.
- `on_getting_item_<item_title>` hook renamed to `on_fetch_item_<item_title>`.
- `on_posting` hook renamed to `on_insert`.
- Datasource projections always include automatic fields (`ID_FIELD`, `LAST_UPDATED`, `DATE_CREATED`). Closes #85.
- Public HTTP methods now override *auth_username_field* Edit. Closes #70 (Bryan Cattle).
- Response date fields are now using GMT instead of UTC. Closes #83.
- Handle the case of 'additional_lookup' field being an integer. If this is the case you can omit the 'url' key, as it will be ignored, and the integer value correctly parsed.
- More informative HTTP error messages. Some more informative error messages have been added for HTTP 400/3/12 and 500 errors. The error messages only show if DEBUG==True (Bryan Cattle).
- `on_getting(resource, documents)` is now `on_getting_resource(resource, documents)`; `on_getting_<resource>(documents)` is now known as ``on_getting_resource_<resource>(documents)` (Ryan Shea).
- Added a new event hook: `on_getting_item_<title>(_id, document)` (Ryan Shea).

- Allow `auth_username_field` to be set to `ID_FIELD` (Bryan Cattle).
- Python 3.3 is now supported.
- Flask 0.10.1 is now supported.
- Werkzeug 0.9.4 is now supported.
- Copyright finally updated to 2013.

Version 0.0.8

Released on July 25th 2013.

- Only run RateLimiting tests if redis-py is installed and redis-server is running.
- CORS `Access-Control-Allow-Headers` header support (Garrin Kimmell).
- CORS `OPTIONS` support for resource and items endpoints (Garrin Kimmell).
- `float` is now available as a data-type in the schema definition ruleset.
- `nullable` field schema rule is now available. If `True` the field value can be set to null. Defaults to `False`.
- v0.3.0 of Cerberus is now a requirement.
- `on_getting`, `on_getting_<resource>` and `on_getting_item` event hooks. These events are raised when documents have just been read from the database and are about to be sent to the client. Registered callback functions can eventually manipulate the documents as needed. Please be aware that `last_modified` and `etag` headers will always be consistent with the state of the documents on the database (they won't be updated to reflect changes eventually applied by the callback functions). Closes #65.
- Documentation fix: `AUTH_USERFIELD_NAME` renamed to `AUTH_USERNAME_FIELD` (Julien Barbot).
- Responses to GET requests for resource endpoints now include a `last` item in the *_links* dictionary. The value is a link to the last page available. The item itself is only provided if pagination is enabled and the page being requested isn't the last one. Closes #62.
- It is now possible to set the MongoDB write concern level at both global (`MONGO_WRITE_CONCERN`) and endpoint (`mongo_write_concern`) levels. The value is a dictionary with all valid MongoDB write_concern settings (w, wtimeout, j and fsync) as keys. `{'w': 1}` is the default, which is also MongoDB's default setting.
- `TestMininal` class added to the test suite. This will allow to start the building of the tests for an application based on Eve, by subclassing the TestMinimal class (Daniele Pizzolli).

Version 0.0.7

Released on June 18th 2013.

- Pinned Werkzeug requirement to v0.8.3 to avoid issues with the latest release which breaks backward compatibility (actually a Flask 0.9 requirements issue, which backtracked to Eve).
- Support for Rate Limiting on all HTTP methods. Closes #58. Please note: to successfully execute the tests in 'eve.tests.methods.ratelimit.py`, a running redis server is needed.
- `utils.request_method` internal helper function added, which allowed for some nice code cleanup (DRY).
- Setting the default 'field' value would not happen if a 'data_relation' was nested deeper than the first schema level. Fixes #60.

- Support for EXTRA_RESPONSE_FIELDS. It is now possible to configure a list of additonal document fields that should be provided with POST responses. Normally only automatically handled fields (ID_FIELD, LAST_UPDATED, DATE_CREATED, etag) are included in POST payloads. EXTRA_RESPONSE_FIELDS is a global setting that will apply to all resource endpoint . Defaults to [], effectively disabling the feature. extra_response_fields is a local resource setting and will override EXTRA_RESPONSE_FIELDS when present.
- on_posting and on_posting_<resource> event hooks. on_posting and on_posting_<resource> events are raised when documents are about to be stored. Among other things this allows callback functions to arbitrarily update the documents being inserted. on_posting(resource, documents) is raised on every successful POST while on_posting_<resource>(documents) is only raised when <resource> is being updated. In both circumstances events will be raised only if at least one document passed validation and is going to be inserted.
- Flask native request.json is now used when decoding request payloads.
- *resource* argument added to Authorization classes. The check_auth() method of all classes in the eve.auth package (BasicAuth, HMACAuth, TokenAuth) now supports the *resource* argument. This allows subclasses to eventually build their custom authorization logic around the resource being accessed.
- MONGO_QUERY_BLACKLIST option added. Allows to blacklist mongo query operators that should not be allowed in resource queries (?where=). Defaults to ['$where', '$regex']. Mongo Javascript operators are disabled by default as they might be used as vectors for injection attacks. Javascript queries also tend to be slow and generally can be easily replaced with the (very rich) Mongo query dialect.
- MONGO_HOST defaults to 'localhost'.
- MONGO_PORT defaults to 27017.
- Support alternative hosts/ports for the test suite (Paul Doucet).

Version 0.0.6

Released on May 13th 2013.

- Content-Type header now properly parsed when additional arguments are included (Ondrej Slinták).
- Only fields defined in the resource schema are now returned from the database. Closes #52.
- Default SERVER_NAME is now set to 127.0.0.1:5000.
- auth_username_field is honored even when there is no query in the request (Thomas Sileo).
- Pagination links in XML payloads are now properly escaped. Fixes #49.
- HEAD requests supported. Closes #48.
- Event Hooks. Each time a GET, POST, PATCH, DELETE method has been executed, both global on_<method> and resource-level on_<method>_<resource> events will be raised. You can subscribe to these events with multiple callback functions. Callbacks will receive the original flask.request object and the response payload as arguments.
- Proper max_results handling in eve.utils.parse_request, refactored tests (Tomasz Jezierski).
- Projections. Projections are conditional queries where the client dictates which fields should be returned by the API (Nicolas Bazire).
- ALLOW_UNKNOWN option, and the corresponding allow_options local setting, allow for a less strict schema validation. Closes #34.
- ETags are now provided with POST responses. Closes #36.
- PATCH performance improvement: ETag is now computed in memory; performing an extra database lookup is not needed anymore.

- Bulk Inserts on the database. POST method heavily refactored to take advantage of MongoDB native support for Bulk Inserts. Please note: validation constraints are checked against the database, and not between the payload documents themselves. This causes an interesting corner case: in the event of a multiple documents payload where two or more documents carry the same value for a field where the `unique` constraint is set, the payload will validate successfully, as there are no duplicates in the database (yet). If this is an issue, the client can always send the documents once at a time for insertion, or validate locally before submitting the payload to the API.
- Responses to document GET requests now include the ETag in both the header and the payload. Closes #29.
- `methods` settings keyword renamed to `resource_methods` for coherence with the global `RESOURCE_METHODS` (Nicolas Carlier).

Version 0.0.5

Released on April 11th 2013.

- Fixed an issue that apparently caused the test suite to only run successfully on the dev box. Thanks Chronidev for reporting this.
- Referential integrity validation via the new `data_relation` schema keyword. Closes #25.
- Support for `Content-Type: application/json` for POST and PATCH methods. Closes #28.
- User-restricted resource access. Works in conjunction with Authentication. When enabled, users can only read/update/delete resource items created by themselves. Can be switched on and off at global level via the `AUTH_USERFIELD_NAME` keywork, or at single resource endpoints with the user_userfield_name keyword (the latter will override the former). The keyword contains the actual name of the field used to store the username of the user who created the resource item. Defaults to '', which disables the feature (Thomas Sileo).
- `PAGING_LIMIT` keyword setting renamed to `PAGINATION_LIMIT` for better coherency with the new `PAGINATION` keyword. This could break backward compatibility in some cases.
- `PAGING_DEFAULT` keyword settings renamed to `PAGINATION_DEFAULT` for better coherence with the new `PAGINATION` keyword. This could break backward compatibility in some cases.
- `ITEM_CACHE_CONTROL` removed as it seems unnecessary at the moment.
- Added an example on how to handle events to perform custom actions. Closes #23 and #22.
- `eve.validation_schema()` now collects offending items and returns all of them into the exception message. Closes #24.
- Filters (`?where=`), sorting (`?sort=`) and pagination (`?page=10`) can now be be disabled at both global and endpoint level. Closes #7.
- CORS (Cross-Origin Resource Sharing) support. The new `X-DOMAINS` keywords allows API maintainers to specify which domains are allowed to perform CORS requests. Allowed values are: None, a list of domains, or '*' for a wide-open API. Closes #1.
- HMAC (Hash Message Authentication Code) based Autentication.
- Token Based Authentication, a variation of Basic Authentication. Closes #20.
- Orphan function removed (`eve.methods.get.standard_links`).
- `DATE_CREATED` and `LAST_UPDATED` fields now show default values for documents created outside the API context. Fixes #18.

Version 0.0.4

Released on February 25th 2013.

- Consistent ETag computation between runs/instances. Closes #16.
- Support for Basic Authentication (RFC2617).
- Support for fine-tuning authentication with `PUBLIC_METHODS` and `PUBLIC_ITEM_METHODS`. By default, access is restricted to *all* endpoints, for *all* HTTP verbs (methods), effectively locking down the whole API.
- Supporto for role-based access control with `ALLOWED_ROLES` and `allowed_roles`.
- Support for all standard Flask initialization parameters.
- Support for default values in resource fields. The new `default` keyword can now be used when defining a field rule set. Please note: currently default values are supported only for main document fields. Default values for fields in embedded documents will be ignored.
- Multiple API endpoints can now target the same database collection. For example now you can set both `/admins/` and `/users/` to read and write from the same collection on the db, *people*. The new `datasource` setting allows to explicitly link API resources to database collections. It is a dictionary with two allowed keys: *source* and *filter*. *source* dictates the database collection consumed by the resource. *filter* is the underlying query, applied by the API when retrieving and validating data for the resource. Previously, the resource name would dictate the linked datasource (and of course you could not have two resources with the same name). This remains the default behaviour: if you omit the `datasource` setting for a resource, its name will be used to determine the database collection.
- It is now possibile to set predefined db filters for each resource. Predefined filters run on top of user queries (GET requests with `where` clauses) and standard conditional requests (`If-Modified-Since`, etc.) Please note that datasource filters are applied on GET, PATCH and DELETE requests. If your resource allows for POST requests (document insertions), then you will probably want to set the validation rules accordingly (in our example, 'username' should probably be a required field).
- JSON-Datetime dependency removed.
- Support for Cerberus v0.0.3 and later.
- Support for Flask-PyMongo v0.2.0 and later.
- Repeated XML requests to the same endpoint could occasionally return an Internal Server Error (Fixes #8).

Version 0.0.3

Released on January 22th 2013.

- XML rendering love. Lots of love.
- JSON links are always wrapped in a `_links` dictionary. Key values match the relation between the item being represented and the linked resource.
- Streamlined JSON responses. Superflous `response` root key has been removed from JSON payloads. GET requests to resource endpoints: items are now wrapped with an `_items` list. GET requests to item endpoints: item is now at root level, with no wrappers around it.
- Support for API versioning through the new API_VERSION configuration setting.
- Boolean values in request forms are now correctly parsed.
- Tests now run under Python 2.6.

Version 0.0.2

Released on November 27th 2012.

- Homepage/api entry point resource links fixed. They had bad 'href' tags which also caused XML validation issues when processing responses (especially when accessing the API via browser).
- Version number in 'Server' response headers.
- Added support for DELETE at resource endpoints. Expected behavior: will delete all items in the collection. Disabled by default.
- `eve.io.mongo.Validator` now supports `Validator` signature, allowing for further subclassing.

Version 0.0.1

Released on November 20th 2012.

- First public preview release.

Note: This documentation is under constant development. Please refer to the links on the sidebar for more information.

INDEX

E