
awokado

Release 0.7

May 21, 2021

1	Installation	3
2	Quickstart	5
2.1	Simple example	5
3	Reference	9
4	Relations	13
5	Diagram	19
6	Features	21
6.1	Filtering	21
6.1.1	syntax	21
6.1.2	available operators	21
6.1.3	examples	21
6.2	Sorting	22
6.2.1	syntax	22
6.2.2	examples	22
6.3	Includes	22
6.3.1	syntax	22
6.3.2	examples	22
6.4	Limit Offset (pagination)	22
6.4.1	syntax	22
6.4.2	examples	22
7	Documentation	23
7.1	Examples	23
8	Changelog	25
8.1	[0.7] - 2019-11-15	25
8.1.1	Changed	25
8.2	[0.6b2] - 2019-11-15	25
8.2.1	Fixes	25
8.3	[0.6b1] - 2019-11-13	25
8.3.1	Removed	25
8.4	[0.6b0] - 2019-11-13	26

	8.4.1	Added	26
	8.4.2	Fixes	26
	8.4.3	Changed	26
	8.4.4	Removed	26
8.5	[0.5b2]	- 2019-10-28	26
	8.5.1	Fixes	26
8.6	[0.5b1]	- 2019-10-28	26
	8.6.1	Fixes	26
8.7	[0.5b0]	- 2019-10-25	26
	8.7.1	Changed	26
8.8	[0.4b5]	- 2019-10-16	27
	8.8.1	Added	27
	8.8.2	Changed	27
8.9	[0.4b4]	- 2019-10-15	27
	8.9.1	Fixes	27
8.10	[0.4b3]	- 2019-09-24	27
	8.10.1	Fixes	27
8.11	[0.4b2]	- 2019-09-23	27
	8.11.1	Added	27
8.12	[0.4b1]	- 2019-09-18	27
	8.12.1	Added	27
	8.12.2	Changed	28
	8.12.3	Removed	28
8.13	[0.3b19]	- 2019-08-28	28
	8.13.1	Added	28
8.14	[0.3b17]	- 2019-07-18	28
	8.14.1	Added	28
8.15	[0.3b16]	- 2019-05-24	28
	8.15.1	Changed	28
8.16	[0.3b15]	- 2019-05-23	28
	8.16.1	Added	28
	8.16.2	Fixes	29
	8.16.3	Removed	29
8.17	[0.3b14]	- 2019-05-15	29
	8.17.1	Fixes	29
8.18	[0.3b13]	- 2019-04-02	29
	8.18.1	Changed	29
8.19	[0.3b12]	- 2019-03-14	29
	8.19.1	Added	29
	8.19.2	Deprecated	30
8.20	[0.3b11]	- 2019-03-13	30
	8.20.1	Added	30
8.21	[0.3b10]	- 2019-03-11	30
	8.21.1	Added	30
8.22	[0.3b7]	- 2019-03-05	30
	8.22.1	Added	30
	8.22.2	Deprecated	30
8.23	[0.3b5]	- 2019-03-04	30
	8.23.1	Added	30
	8.23.2	Fixes	31
8.24	[0.3b2]	- 2019-03-01	31
	8.24.1	Added	31
	8.24.2	Changed	31
	8.24.3	Deprecated	31

8.24.4 Removed	31
9 Indices and tables	33
Python Module Index	35
Index	37

Fast and flexible low-level API framework based on [Falcon](#) , [Marshmallow](#) and [SQLAlchemy Core](#) API is close to OpenAPI 3.0 specification.

Awokado uses [dynaconf](#) for loading it settings. You can find all available variables in *settings.toml* file.

CHAPTER 1

Installation

Use pipenv at the command line:

```
$ pipenv install awokado
```

or pip:

```
$ pip install awokado
```


CHAPTER 2

Quickstart

Install awokado before it's too late.

Awokado uses `dynaconf` for loading its settings. So store them in `settings.toml`, for example:

```
1 #settings.toml
2
3 [default]
4     DATABASE_PASSWORD='your_password'
5     DATABASE_HOST='localhost'
6     DATABASE_USER='your_user'
7     DATABASE_PORT=5432
8     DATABASE_DB='try_awokado'
```

2.1 Simple example

Awokado based on Falcon, so we use the REST architectural style. That means we're talking about resources. Resources are simply all the things in your API or application that can be accessed by a URL.

First of all, we need to create a model to further be connected with a resource.

This model will act as a link to a database entity. Read more about it [here](#).

At this point, the database should be already created.

```
1 #books.py
2
3 import falcon
4 import sqlalchemy as sa
5 from awokado.consts import CREATE, READ, UPDATE, DELETE
6 from awokado.db import DATABASE_URL
7 from awokado.middleware import HttpMiddleware
8 from awokado.resource import BaseResource
9 from marshmallow import fields
```

(continues on next page)

(continued from previous page)

```

10 from sqlalchemy import create_engine
11 from sqlalchemy.ext.declarative import declarative_base
12
13 Base = declarative_base()
14
15
16 class Book(Base):
17     __tablename__ = "books"
18
19     id = sa.Column(sa.Integer, primary_key=True, autoincrement=True)
20     description = sa.Column(sa.Text)
21     title = sa.Column(sa.Text)
22
23
24 e = create_engine(DATABASE_URL)
25 Base.metadata.create_all(e)

```

Resources are represented as classes inherited from awokado `BaseResource`, that gives an opportunity to use get, create, delete, update methods.

```

1 class BookResource(BaseResource):
2     class Meta:
3         model = Book
4         name = "book"
5         methods = (CREATE, READ, UPDATE, DELETE)
6
7     id = fields.Int(model_field=Book.id)
8     title = fields.String(model_field=Book.title, required=True)
9     description = fields.String(model_field=Book.description)

```

Add routes, so resources can handle requests:

```

1 api = falcon.API(middleware=[HttpMiddleware()])
2
3 api.add_route("/v1/book/", BookResource())
4 api.add_route("/v1/book/{resource_id}", BookResource())

```

The final file version should look like [this one](#).

Now we're ready to run the above example. You can use the `uwsgi` server.

```

1 pipenv install uwsgi
2 uwsgi --http :8000 --wsgi-file books.py --callable api

```

Test it using `curl` in another terminal.

Create entity using following `curl`:

```

1 curl localhost:8000/v1/book --data-binary '{"book":{"title":"some_title","description
↵":"some_description"}}' --compressed -v | python -m json.tool
2
3 {
4     "book": [
5         {
6             "description": "some_description",
7             "id": 1,
8             "title": "some_title"

```

(continues on next page)

(continued from previous page)

```
9         }
10     ]
11 }
```

And then, with read request see what you've got:

```
1 curl localhost:8000/v1/book | python -m json.tool
2
3 {
4     "meta": {
5         "total": 1
6     },
7     "payload": {
8         "book": [
9             {
10                 "description": "some_description",
11                 "id": 1,
12                 "title": "some_title"
13             }
14         ]
15     }
16 }
```


In class `Meta` we declare different resource's options. There is a possibility to write your own behavior for certain methods .

```
class awokado.resource.BaseResource
```

```
class Response (resource: BaseResource, is_list: bool = False)
```

Response class helps to collect your data and prepare it in a readable format for the Frontend (or another API Client)

You can override it in your resource to change response format:

```
class MyResponse (Response) :  
    PAYLOAD_KEYWORD = "data"  
  
class MyBaseResource (BaseResource) :  
    Response = MyResponse
```

Default serialization for list requests (`/v1/book/`):

```
{  
  "payload": {  
    "book": [  
      {  
        "name": "My Book",  
        "authors": [1, 2]  
      }  
    ]  
  },  
  "meta": {  
    "total": 1  
  }  
}
```

Default serialization for single object (`/v1/book/123`):

```
{
  "book": [
    {
      "name": "My Book",
      "authors": [1, 2]
    }
  ]
}
```

auth (**args, **kwargs*) → awokado.utils.AuthBundle

This method should return (user_id, token) tuple

create (*session: sqlalchemy.orm.session.Session, payload: dict, user_id: int*) → dict

Create method

You can override it to add your logic.

First of all, data is prepared for creating: Marshmallow load method for data structure deserialization and then preparing data for SQLAlchemy create a query.

Inserts data to the database (Uses bulky library if there is more than one entity to create). Saves many-to-many relationships.

Returns created resources with the help of read_handler method.

delete (*session: sqlalchemy.orm.session.Session, user_id: int, obj_ids: list*)

Simply deletes objects with passed identifiers

on_delete (*req: falcon.request.Request, resp: falcon.response.Response, resource_id: int = None*)

Falcon method. DELETE-request entry point.

Here is a database transaction opening. This is where authentication takes place (if auth class is pointed in *resource*) Then delete method is run.

on_get (*req: falcon.request.Request, resp: falcon.response.Response, resource_id: int = None*)

Falcon method. GET-request entry point.

Here is a database transaction opening. This is where authentication takes place (if auth class is pointed in *resource*) Then read_handler method is run. It's responsible for the whole read workflow.

on_patch (*req: falcon.request.Request, resp: falcon.response.Response, *args, **kwargs*)

Falcon method. PATCH-request entry point.

Here is a database transaction opening. This is where authentication takes place (if auth class is pointed in *resource*) Then update method is run.

on_post (*req: falcon.request.Request, resp: falcon.response.Response*)

Falcon method. POST-request entry point.

Here is a database transaction opening. This is where authentication takes place (if auth class is pointed in *resource*) Then create method is run.

update (*session: sqlalchemy.orm.session.Session, payload: dict, user_id: int, *args, **kwargs*) → dict

First of all, data is prepared for updating: Marshmallow load method for data structure deserialization and then preparing data for SQLAlchemy update query.

Updates data with bulk_update_mappings sqlalchemy method. Saves many-to-many relationships.

Returns updated resources with the help of read_handler method.

class awokado.meta.ResourceMeta

Parameters

- **name** – Resource name. Used for two resources connection by relation
- **model** – represents sqlalchemy model or cte
- **methods** – tuple of methods you want to allow
- **auth** – awokado *BaseAuth* class for embedding authentication logic
- **skip_doc** – set true if you don't need to add the resource to documentation
- **disable_total** – set false, if you don't need to know returning objects amount in read-requests
- **id_field** – you can specify your own primary key if it's different from the 'id' field. Used in reading requests (GET)
- **select_from** – provide data source here if your resource use another's model fields (for example `sa.outerjoin(FirstModel, SecondModel, FirstModel.id == SecondModel.first_model_id)`)

class awokado.auth.BaseAuth

CREATE = { 'ROLE NAME': Boolean value }

Example: 'ADMIN': True, 'GUEST': False

READ, UPDATE, DELETE

class awokado.response.Response

Response class helps to collect your data and prepare it in a readable format for the Frontend (or another API Client)

You can override it in your resource to change response format:

```
class MyResponse(Response):
    PAYLOAD_KEYWORD = "data"

class MyBaseResource(BaseResource):
    Response = MyResponse
```

Default serialization for list requests (/v1/book/):

```
{
  "payload": {
    "book": [
      {
        "name": "My Book",
        "authors": [1, 2]
      }
    ]
  },
  "meta": {
    "total": 1
  }
}
```

Default serialization for single object (/v1/book/123):

```
{
  "book": [
    {
      "name": "My Book",
      "authors": [1, 2]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
    }  
  ]  
}
```

```
META_KEYWORD = 'meta'
```

```
PAYLOAD_KEYWORD = 'payload'
```

```
TOTAL_KEYWORD = 'total'
```

```
serialize() → dict
```

```
set_parent_payload(parent_payload: Optional[List[T]] = None) → None
```

```
set_related_payload(related_payload: Optional[Dict[KT, VT]]) → None
```

```
set_total(total_objects_count: int)
```

CHAPTER 4

Relations

Here is a more complicated version of simple awokado usage. Awokado provides you with the possibility to easily build relations between entities.

Let's take the Authors-Books one-to-many relation, for example.

Firstly, we need models:

```
1 #models.py
2
3 import sqlalchemy as sa
4 from awokado.db import DATABASE_URL
5 from sqlalchemy import create_engine
6 from sqlalchemy.ext.declarative import declarative_base
7
8
9 Base = declarative_base()
10
11
12 class Book(BaseModel):
13     __tablename__ = "books"
14
15     id = sa.Column(sa.Integer, primary_key=True, autoincrement=True)
16     author_id = sa.Column(
17         sa.Integer,
18         sa.ForeignKey("authors.id", onupdate="CASCADE", ondelete="SET NULL"),
19         index=True,
20     )
21
22     description = sa.Column(sa.Text)
23     title = sa.Column(sa.Text)
24
25 class Author(BaseModel):
26     __tablename__ = "authors"
27
28     id = sa.Column(sa.Integer, primary_key=True, autoincrement=True)
```

(continues on next page)

(continued from previous page)

```

29     first_name = sa.Column(sa.Text, nullable=False)
30     last_name = sa.Column(sa.Text, nullable=False)
31
32 e = create_engine(DATABASE_URL)
33 Base.metadata.create_all(e)

```

Secondly, we write resources for each entity connected with their models.

Bind Book to Author using the ToOne awokado custom_field. Resource argument is the name field of Meta class in Author resource we're connecting to, model_field argument is the field in Book model where Author unique identifier is stored.

```

1  #resources.py
2
3  import sqlalchemy as sa
4  from awokado import custom_fields
5  from awokado.consts import CREATE, READ, UPDATE
6  from awokado.resource import BaseResource
7  from awokado.utils import ReadContext
8  from marshmallow import fields
9
10 import models as m
11
12 class BookResource(BaseResource):
13     class Meta:
14         model = m.Book
15         name = "book"
16         methods = (CREATE, READ, UPDATE)
17
18     id = fields.Int(model_field=m.Book.id)
19     title = fields.String(model_field=m.Book.title, required=True)
20     description = fields.String(model_field=m.Book.description)
21     author = custom_fields.ToOne(
22         resource="author", model_field=m.Book.author_id
23     )

```

The continuation of building the connection is in the Author resource. Here we define another end of connection by the ToMany field.

```

1  class AuthorResource(Resource):
2     class Meta:
3         model = m.Author
4         name = "author"
5         methods = (CREATE, READ, UPDATE)
6         select_from = sa.outerjoin(
7             m.Author, m.Book, m.Author.id == m.Book.author_id
8         )
9
10     id = fields.Int(model_field=m.Author.id)
11     books = custom_fields.ToMany(
12         fields.Int(),
13         resource="book",
14         model_field=m.Book.id,
15         description="Authors Books",
16     )
17     books_count = fields.Int(
18         dump_only=True, model_field=sa.func.count(m.Book.id)

```

(continues on next page)

(continued from previous page)

```

19     )
20     name = fields.String(
21         model_field=sa.func.concat(
22             m.Author.first_name, " ", m.Author.last_name
23         ),
24         dump_only=True,
25     )
26     last_name = fields.String(
27         model_field=m.Author.last_name, required=True, load_only=True
28     )
29     first_name = fields.String(
30         model_field=m.Author.first_name, required=True, load_only=True
31     )

```

So finally here are the methods where we add logic for getting connected entities.

Pay attention, that methods should be named with following template - “get_by_<name>_ids”, where “<name>” is Meta class name option in resource you connecting to.

```

1  #BookResource
2
3  def get_by_author_ids(
4      self, session, ctx: ReadContext, field: sa.Column = None
5  ):
6      authors = sa.func.array_remove(
7          sa.func.array_agg(m.Author.id), None
8      ).label("authors")
9      q = (
10         sa.select(
11             [
12                 m.Book.id.label("id"),
13                 m.Book.title.label("title"),
14                 m.Book.description.label("description"),
15                 authors,
16             ]
17         )
18         .select_from(
19             sa.outerjoin(m.Book, m.Author, m.Author.id == m.Book.author_id)
20         )
21         .where(m.Book.author_id.in_(ctx.obj_ids))
22         .group_by(m.Book.id)
23     )
24     result = session.execute(q).fetchall()
25     serialized_objs = self.dump(result, many=True)
26     return serialized_objs
27
28
29  #AuthorResource
30
31  def get_by_book_ids(
32      self, session, ctx: ReadContext, field: sa.Column = None
33  ):
34      books_count = self.fields.get("books_count").metadata["model_field"]
35      q = (
36         sa.select(
37             [
38                 m.Author.id.label("id"),

```

(continues on next page)

(continued from previous page)

```

39         self.fields.get("name")
40         .metadata["model_field"]
41         .label("name"),
42         books_count.label("books_count"),
43     ]
44 )
45 .select_from(
46     sa.outerjoin(m.Author, m.Book, m.Author.id == m.Book.author_id)
47 )
48 .where(m.Book.id.in_(ctx.obj_ids))
49 .group_by(m.Author.id)
50 )
51 result = session.execute(q).fetchall()
52 serialized_objs = self.dump(result, many=True)
53 return serialized_objs

```

Add routes, so resources can handle requests:

```

1 app = falcon.API()
2 api.add_route("/v1/author/", AuthorResource())
3 api.add_route("/v1/author/{resource_id}", AuthorResource())
4 api.add_route("/v1/book/", BookResource())
5 api.add_route("/v1/book/{resource_id}", BookResource())

```

Test it using curl in terminal.

Create entities using following curl:

```

1 curl localhost:8000/v1/author --data-binary '{"author":{"last_name": "B", "first_name
  ↳": "Sier"}}' --compressed -v | python -m json.tool
2
3 {
4     "author": [
5         {
6             "books": [],
7             "books_count": 0,
8             "id": 1,
9             "name": "Sier B"
10        }
11    ]
12 }
13
14 curl localhost:8000/v1/book --data-binary '{"book":{"title":"some_title","description
  ↳":"some_description", "author":"1"}}' --compressed -v | python -m json.tool
15
16 {
17     "book": [
18         {
19             "author": 1,
20             "description": "some_description",
21             "id": 1,
22             "title": "some_title"
23        }
24    ]
25 }

```

And then, with read request see what you've got:

```
1 curl localhost:8000/v1/author?include=books | python -m json.tool
```

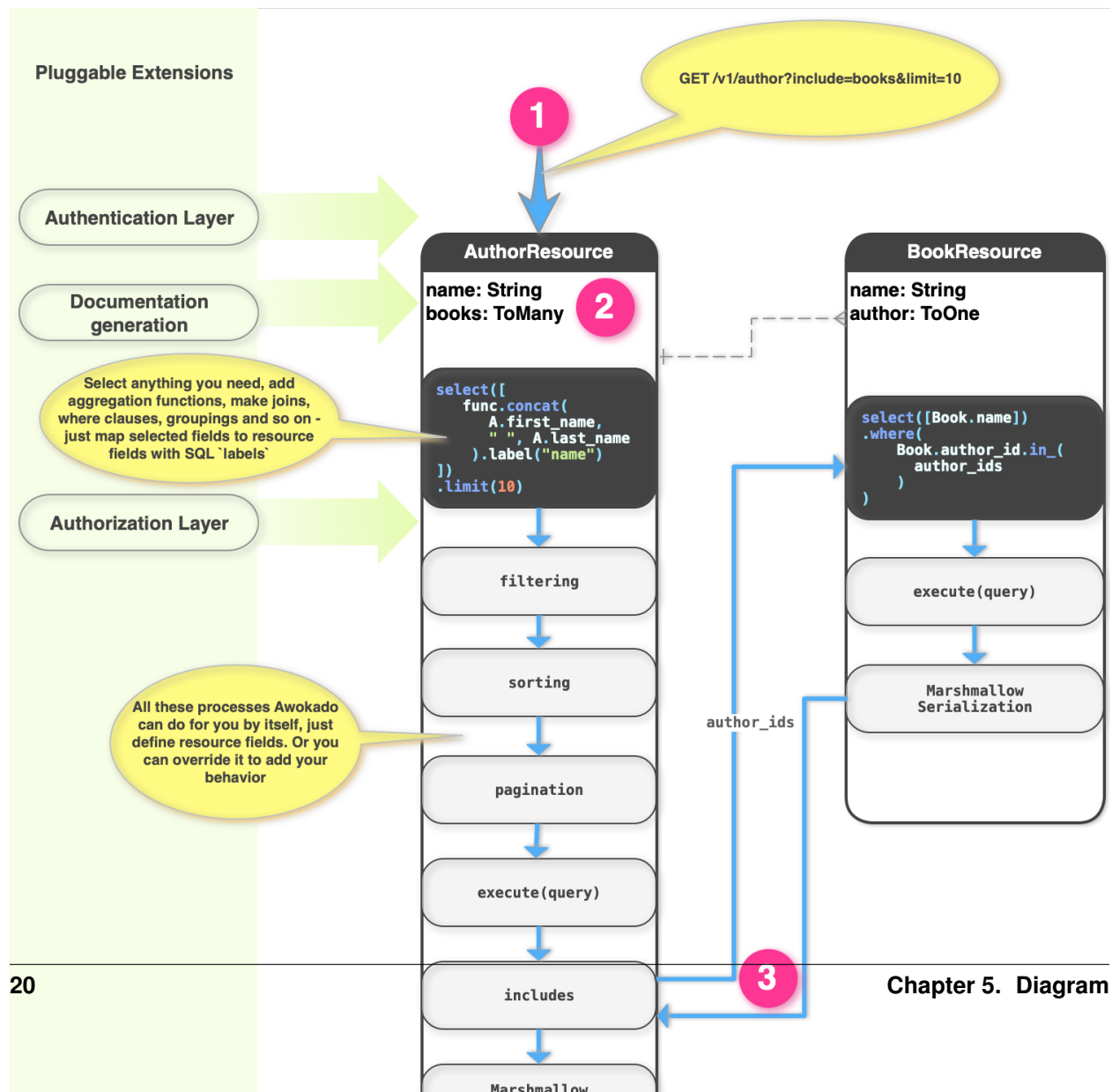
```
2 {
3   "meta": {
4     "total": 1
5   },
6   "payload": {
7     "author": [
8       {
9         "books": [
10           1
11         ],
12         "books_count": 1,
13         "id": 1,
14         "name": "Sier B"
15       }
16     ],
17     "book": [
18       {
19         "description": "some_description",
20         "id": 1,
21         "title": "some_title"
22       }
23     ]
24   }
25 }
```

```
26
27
28 curl localhost:8000/v1/book?include=author | python -m json.tool
```

```
29 {
30   "meta": {
31     "total": 1
32   },
33   "payload": {
34     "author": [
35       {
36         "books_count": 1,
37         "id": 1,
38         "name": "Sier B"
39       }
40     ],
41     "book": [
42       {
43         "author": 1,
44         "description": "some_description",
45         "id": 1,
46         "title": "some_title"
47       }
48     ]
49   }
50 }
51
```


CHAPTER 5

Diagram



6.1 Filtering

6.1.1 syntax

`resource_field_name[operator]=value`

6.1.2 available operators

- lte
- eq
- gte
- ilike
- in
- empty
- contains
- gt
- lt

6.1.3 examples

`/v1/user/?username[ilike]=Andy`

it's equal to SQL statement: `SELECT * FROM users WHERE username ILIKE '%Andy%';`

`/v1/user/?id[in]=1,2,3,4`

it's equal to SQL statement: `SELECT * FROM users WHERE id IN (1,2,3,4);`

6.2 Sorting

6.2.1 syntax

`sort=resource_field_name,-another_resource_field_name`

use `-` for descending order

6.2.2 examples

`/v1/user/?sort=name,-record_created`

6.3 Includes

6.3.1 syntax

`include=resource_relation_name`

6.3.2 examples

`/v1/author/?include=books`

`/v1/author/?include=books,stores`

6.4 Limit Offset (pagination)

6.4.1 syntax

`limit=integer&offset=integer`

6.4.2 examples

`/v1/user/?limit=10&offset=10`

`/v1/user/?offset=10`

`/v1/user/?limit=2000`

CHAPTER 7

Documentation

Awokado allows to generate documentation for a project using swagger(3rd version). To generate documentation you need to import add route with `SwaggerResource` after adding all other routes. Also, you can choose UI for swagger (SwaggerUI or Redoc) by adding route for resources `SwaggerUIResource` or `RedocViewResource`

```
class awokado.documentation.SwaggerResource
```

```
    __init__(api: falcon.api.API, project_name: str, api_hosts: Union[str, Sequence[str]] = '/',
             api_version: str = '1.0.0', description: str = "")
             Resource for '/swagger.yaml'
```

Parameters

- **api** – Your falcon.API instance
- **project_name** – Title for your documentation
- **api_hosts** – List of places where api can be
- **api_version** – String with number of version of you project
- **description** – Absolute path to template with description of your project

```
class awokado.documentation.SwaggerUIResource
```

```
class awokado.documentation.RedocViewResource
```

7.1 Examples

```
1  from awokado.documentation import generate_documentation
2  from dynaconf import settings
3  from api.routes import api
4
5  api.add_route(
6      "/swagger.yaml",
```

(continues on next page)

(continued from previous page)

```
7     ApiResource(  
8         api=api,  
9         project_name="Example Documentation",  
10        api_host=('http://example.com/api', '/'),  
11        api_version="4.2.0",  
12        description="Some api description and introducing words"  
13    )  
14 )  
15  
16 # You can choose one  
17 api.add_route("/doc", SwaggerUIResource("/swagger.yaml"))  
18 api.add_route("/redoc", RedocViewResource("/swagger.yaml"))
```

CHAPTER 8

Changelog

The format is based on [Keep a Changelog](#)

8.1 [0.7] - 2019-11-15

8.1.1 Changed

- Update sqlalchemy 1.3.10 -> 1.3.23
- Update marshmallow 3.2.2 -> 3.12.1
- Update apispec 3.1.0 -> 3.3.2

8.2 [0.6b2] - 2019-11-15

8.2.1 Fixes

- Fix Pipfile parser in setup.py

8.3 [0.6b1] - 2019-11-13

8.3.1 Removed

- Unused exceptions

8.4 [0.6b0] - 2019-11-13

8.4.1 Added

- `ApiViewer` resource (swagger ui)

8.4.2 Fixes

- Fixed many typing issues, code simplified

8.4.3 Changed

- Function `generate_documentation` replaced with `ApiResource` class
- Updated `marshmallow` and `apispec` dependencies to latest versions
- `Resource.Meta` now object of class `ResourceMeta`
- Documentation now don't create `swagger.yaml` file. Now it stored in memory

8.4.4 Removed

- `get_readme`, `empty_response` functions

8.5 [0.5b2] - 2019-10-28

8.5.1 Fixes

- Move args parsing to `FilterItem` dataclass
- Fix typing issues
- Fix code duplication in `validate_create_request`

8.6 [0.5b1] - 2019-10-28

8.6.1 Fixes

- Fix bug in `ReadContext` imports

8.7 [0.5b0] - 2019-10-25

8.7.1 Changed

- Argument `resp: falcon.Response` removed from functions `validate_create_request` and `validate_update_request`
- Read logic moved to `ReadContext` class

8.8 [0.4b5] - 2019-10-16

8.8.1 Added

- Class for database config

8.8.2 Changed

- Move `cached_property` to `deps`
- Now we use Python 3.7

8.9 [0.4b4] - 2019-10-15

8.9.1 Fixes

- Fix receiving field from relative model

8.10 [0.4b3] - 2019-09-24

8.10.1 Fixes

- `Resource.Meta.skip_doc = True` was generating broken `swagger.yml`

8.11 [0.4b2] - 2019-09-23

8.11.1 Added

- Added `lt` and `gt` operators in filtering.

8.12 [0.4b1] - 2019-09-18

8.12.1 Added

- `pipenv` implemented.
- `mypy` implemented.
- `awokado.response.Response` class added. [Read more](#)
- method `awokado.resource.BaseResource.read__serializing` now uses `awokado.response.Response` class to serialize data. Take a look at the [code](#) if you already override this method

8.12.2 Changed

- `awokado.utils.ReadContext` moved to `awokado.request.ReadContext`

8.12.3 Removed

- `awokado.utils.empty_response` function removed (use `awokado.response.Response` instead)

8.13 [0.3b19] - 2019-08-28

8.13.1 Added

- Added documentation.

8.14 [0.3b17] - 2019-07-18

8.14.1 Added

- Added ability to use any resource field as “id” field. Currently only for read logic. This field can be specified as `id_field` attribute of resource’s `Meta` class
- Added schema validation for POST requests with a friendly message about the exception

8.15 [0.3b16] - 2019-05-24

8.15.1 Changed

Updated falcon from 1.4.1 to 2.0.0 ([Falcon 2 Changelog](#)), which led to following changes:

- `application.req_options.auto_parse_qs_csv` param is now `False` by default, so you’ll need to manually set it to `True`
- `application.req_options.strip_url_path_trailing_slash` param is now `False` by default, so you’ll need to manually set it to `True`
- For direct data read from request `req.bounded_stream` is now used instead of `req.stream`.

8.16 [0.3b15] - 2019-05-23

8.16.1 Added

- Added pre-commit hook to run black formatting checks
- Added ability to specify full database url in settings

8.16.2 Fixes

- Fixed “load_only” fields appearing in read request results
- Fixed “awokado_debug” setting being always required in settings
- Fixed attribute “auth” being mandatory in resource.Meta
- Fixed method “auth” being mandatory to overwrite in resource
- Fixed method “audit_log” being mandatory to overwrite in resource

8.16.3 Removed

- Functions `set_bearer_header`, `get_bearer_payload` and `AWOKADO_AUTH_BEARER_SECRET` var are removed

8.17 [0.3b14] - 2019-05-15

8.17.1 Fixes

- Fixed documentation generation for bulk operations

8.18 [0.3b13] - 2019-04-02

8.18.1 Changed

No backward compatibility. Need to change custom `delete()` and `can_create()` methods.

- `delete` method supports bulk delete.
- add `payload` attribute to `can_create` method.

8.19 [0.3b12] - 2019-03-14

8.19.1 Added

- `select_from` attribute in class `Meta`, allows you to specify `sqlalchemy.select_from()` arguments.
Example:

```
class AuthorResource(Resource):
    class Meta:
        model = m.Author
        name = "author"
        methods = (CREATE, READ, UPDATE, BULK_UPDATE, DELETE)
        auth = None
        select_from = sa.outerjoin(
            m.Author, m.Book, m.Author.id == m.Book.author_id
        )
```

8.19.2 Deprecated

- `join` argument in the resource field

8.20 [0.3b11] - 2019-03-13

8.20.1 Added

- ability to make list of joins in resource field

```
book_titles = fields.List(
    fields.Str(),
    resource="author",
    model_field=sa.func.array_remove(sa.func.array_agg(m.Book.title), None),
    join=[
        OuterJoin(
            m.Tag, m.M2M_Book_Tag, m.Tag.id == m.M2M_Book_Tag.c.tag_id
        ),
        OuterJoin(
            m.M2M_Book_Tag, m.Book, m.M2M_Book_Tag.c.book_id == m.Book.id
        ),
    ],
)
```

8.21 [0.3b10] - 2019-03-11

8.21.1 Added

- Automated SQL generation for POST/PATCH requests

8.22 [0.3b7] - 2019-03-05

8.22.1 Added

- `bulk_create` method in base resource

8.22.2 Deprecated

- `create` method (is going to be replaced with `bulk_create`)

8.23 [0.3b5] - 2019-03-04

8.23.1 Added

- API simple workflow diagram

- `disable_total` attr for `Resource.Meta`. Set it to `True` to avoid adding total column: `sa.func.count().over()`. Useful for historical tables, where pagination based on date instead of limit / offset to not overload SQL database

8.23.2 Fixes

- Fixed `description` arg for `ToMany` and `ToOne` fields (was broken)

8.24 [0.3b2] - 2019-03-01

8.24.1 Added

- Documentation generation for API resources
- Automated SQL generation for `GET` requests (including `ToOne` and `ToMany` relation fields)
- `AWOKADO_DEBUG` handle traceback exception in API response

8.24.2 Changed

- all `Forbidden` exceptions now raise `HTTP_403` instead of `HTTP_401`

8.24.3 Deprecated

-

8.24.4 Removed

- `###` Fixed
-

CHAPTER 9

Indices and tables

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

a

`awokado.auth`, [11](#)

Symbols

`__init__()` (*awokado.documentation.SwaggerResource* method), 23

A

`auth()` (*awokado.resource.BaseResource* method), 10
`awokado.auth` (module), 11

B

`BaseAuth` (class in *awokado.auth*), 11
`BaseResource` (class in *awokado.resource*), 9
`BaseResource.Response` (class in *awokado.resource*), 9

C

`create()` (*awokado.resource.BaseResource* method), 10

D

`delete()` (*awokado.resource.BaseResource* method), 10

M

`META_KEYWORD` (*awokado.response.Response* attribute), 12

O

`on_delete()` (*awokado.resource.BaseResource* method), 10
`on_get()` (*awokado.resource.BaseResource* method), 10
`on_patch()` (*awokado.resource.BaseResource* method), 10
`on_post()` (*awokado.resource.BaseResource* method), 10

P

`PAYLOAD_KEYWORD` (*awokado.response.Response* attribute), 12

R

`RedocViewResource` (class in *awokado.documentation*), 23
`ResourceMeta` (class in *awokado.meta*), 10
`Response` (class in *awokado.response*), 11

S

`serialize()` (*awokado.response.Response* method), 12
`set_parent_payload()` (*awokado.response.Response* method), 12
`set_related_payload()` (*awokado.response.Response* method), 12
`set_total()` (*awokado.response.Response* method), 12
`SwaggerResource` (class in *awokado.documentation*), 23
`SwaggerUIResource` (class in *awokado.documentation*), 23

T

`TOTAL_KEYWORD` (*awokado.response.Response* attribute), 12

U

`update()` (*awokado.resource.BaseResource* method), 10