flask-rest-jsonapi Documentation

Release 0.1

miLibris

Mar 13, 2017

Contents

1 Contents **3**

1.1 Installation . . . 3
1.2 Resource . . . 3
1.3 Data layer . . . 6
1.4 Sorting . . . 10
1.5 Fields restriction . . . 10
1.6 Filtering . . . 11
1.7 Pagination . . . 11
1.8 Routing . . . 12
1.9 Tutorial . . . 14

2 Api reference **25**

Flask-REST-JSONAPI is a library to help you build REST apis. It is built around:

- jsonapi: a specification for building apis in json
- flask: a microframework for Python based on Werkzeug
- marshmallow-jsonapi: JSON API 1.0 formatting with marshmallow
- sqlalchemy: SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.
- mongodb: a free and open-source cross-platform document-oriented database program

I have created this library because I was looking for the best way to implement a REST api. The jsonapi specification is a very strong specification about interactions between the api and the caller and I think it is a very good one.

There is a lot of very good REST libraries based on Flask like Flask-RESTful or Flask-Restless but I would like to combine the flexibility of Flask-RESTful with the simplicity of Flask-Restless and the power of Marshmallow and SQLAlchemy around a strong and shareable communication protocol: jsonapi.

Moreover, most Flask frameworks only support SQLAlchemy so I would like to create an generic abstraction to communicate with any data provider: the data layer system. Current available data layers are:

- SQLAlchemy
- MongoDB

You can easily create and use your own data layer to communicate with the data provider of your choice. Read the data layer section to learn more.

Here is a quick example:

```
from flask import Flask
from flask_rest_jsonapi import Api, ResourceDetail

app = Flask(__name__)
api = Api(app)


class HelloWorld(ResourceDetail):
    def get(self):
        return "Hello world"

api.detail_route('index', '/', resource_cls=HelloWorld)

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

Save this file as api.py

Launch local server:

```
$ python api.py
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
```

Now you can try this:

```
$ curl "http://127.0.0.1:5000/" -H "Content-Type: application/vnd.api+json"\
  -H "Accept: application/vnd.api+json"
"Hello world"
```

Note: All code examples in this tutorial are based on a classic blog example with topic and post.

CHAPTER 1

Contents

Installation

```
$ pip install flask-rest-jsonapi
```

Resource

Flask-REST-JSONAPI provides 3 resource class helpers:

- ResourceList
- ResourceDetail
- ResourceRelationship

Note: If you forget to set one of the required attributes of a resource class, the library will raise an Exception to help you find which attribute is missing in which resource class.

ResourceList

This class provides a default implementation of GET and POST methods to:

- GET: retrieve a list of items with the GET request method
- POST: create an item with POST request method

You can rewrite those methods if the default behaviour is not enough for you.

If you want to use one of those default method implementations, you have to configure your resource class.

Class attributes:

- resource_type (str): name of the resource type
- schema (dict): schema information:
  - cls (Schema): a marshmallow schema class
  - get_kwargs (dict) *Optional*: additional kwargs for instance schema in get method
  - post_kwargs (dict) *Optional*: additional kwargs for instance schema in post method
- endpoint (dict): endpoint information:
  - name (str): name of the endpoint
  - include_view_kwargs (boolean) *Optional*: set it to True if you want to include view kwargs to the endpoint url build context
- Meta (class):
  - data_layer (dict): data layer information:
    - cls (BaseDataLayer): a data layer class like SqlalchemyDataLayer, MongoDataLayer or your custom data layer
  - get_decorators (list) *Optional*: a list of decorators to plug to the get method
  - post_decorators (list) *Optional*: a list of decorators to plug to the post method
  - disabled_methods (list) *Optional*: a list of request methods to disallow access to. Those methods will return a 405 (Method Not Allowed) status code

Example:

```
from flask_rest_jsonapi import ResourceList, SqlalchemyDataLayer

from your_project.models import Post
from your_project.schemas import PostSchema
from your_project.extensions import sql_db, oauth2

def get_base_query(self, **view_kwargs):
    query = self.session.query(Post)


class PostList(ResourceList):
    class Meta:
        data_layer = {'cls': SqlalchemyDataLayer,
                      'kwargs': {'model': Post, 'session': sql_db.session},
                      'get_base_query': get_base_query}

        get_decorators = [oauth2.require_oauth('post_list')]
        post_decorators = [oauth2.require_oauth('post_create')]

    resource_type = 'post'
    schema = {'cls': PostSchema,
              'get_kwargs': {'only': ('title', 'content', 'created')},
              'post_kwargs': {'only': ('title', 'content')}}
    endpoint = {'name': 'post_list',
                'include_view_kwargs': True}
```

ResourceDetail

This class provides a default implementation of GET, PATCH and DELETE methods to:

- GET: retrieve item details with the GET request method
- POST: update an item with the PATCH request method
- DELETE: delete an item with the DELETE request method

You can rewrite those methods if the default behaviour is not enough for you.

If you want to use one of those default method implementations, you have to configure your resource class.

Class attributes:

- resource_type (str): name of the resource type
- schema (dict): schema information:
  - cls (Schema): a Marshmallow schema class
  - get_kwargs (dict) *Optional*: additional kwargs for instance schema in get method
  - patch_kwargs (dict) *Optional*: additional kwargs for instance schema in patch method
- Meta (class):
  - data_layer (dict): data layer information:
    - cls (BaseDataLayer): a data layer class like SqlalchemyDataLayer, MongoDataLayer or your custom data layer
  - get_decorators (list) *Optional*: a list of decorators to plug to the get method
  - post_decorators (list) *Optional*: a list of decorators to plug to the post method
  - disabled_methods (list) *Optional*: a list of request methods to disallow acces to. Those methods will return a 405 (Method Not Allowed) status code

Example:

```
from flask_rest_jsonapi import ResourceList, SqlalchemyDataLayer

from your_project.models import Post
from your_project.schemas import PostSchema
from your_project.extensions import sql_db

class PostDetail(ResourceDetail):

    class Meta:
        data_layer = {'cls': SqlalchemyDataLayer,
                      'kwargs': {'session': sql_db.session,
                                 'model': Post,
                                 'id_field': 'post_id',
                                 'url_param_name': 'post_id'}}

        get_decorators = [oauth2.require_oauth('provider_detail')]
        patch_decorators = [oauth2.require_oauth('provider_update')]

        disabled_methods = ['DELETE']

    resource_type = 'provider'
    schema = {'cls': ProviderSchema,
              'get_kwargs': {'only': ('title', 'content', 'created', 'author')},
              'patch_kwargs': {'only': ('title', 'content')}}
```

Method rewrite

If you want to rewrite the default implementation of a resource method you can return a tuple instead of flask BaseReponse, like in Flask-RESTful.

Example:

```
from flask import Flask
from flask_rest_jsonapi import ResourceDetail

app = Flask(__name__)


class HelloWorld(ResourceDetail):
    def get(self):
        return "Hello world", 202, {'custom_header':'custom_header_value'}
```

Keep in mind that if you want to stay compliant with jsonapi specifications you have to return well formatted json responses and status code. For example if you rewrite the POST method to distribute the creation of an item you have to return a 202 (Accepted) status code.

Data layer

A data layer is a CRUD interface between resource methods and data providers or ORMs. The data layer class must implement the BaseDataLayer class. You can use one of the provided default classes:

- Sqlalchemy
- Mongodb

But you can also create a custom one that better fits your needs.

Usage example:

```
from flask_rest_jsonapi import ResourceList, SqlalchemyDataLayer

from your_project.models import Post
from your_project.schemas import PostSchema
from your_project.extensions import sql_db

def get_base_query(self, **view_kwargs):
    query = self.session.query(Post)


class PostList(ResourceList):
    class Meta:
        data_layer = {'cls': SqlalchemyDataLayer,
                      'kwargs': {'model': Post, 'session': sql_db.session},
                      'get_base_query': get_base_query}

    resource_type = 'post'
    schema = {'cls': PostSchema}
    endpoint = {'name': 'post_list'}
```

Sqlalchemy

A data layer around SQLAlchemy

ResourceList

Instance attributes:

- model (Model): sqlalchemy model
- session (Session): sqlalchemy session instance

Class attributes:

- get_base_query (callable): a callable to retrieve the base data in GET method
- before_create_instance (callable) *Optional*: additional operations before creating an instance in the POST method

Example:

```
import datetime

from sqlalchemy import NoResultFound
from flask_rest_jsonapi import ResourceList, SqlalchemyDataLayer

from your_project.models import Post
from your_project.schemas import PostSchema
from your_project.extensions import sql_db
from your_project.lib import get_topic


def get_base_query(self, **view_kwargs):
    query = self.session.query(Post)


def before_create_instance(self, data, **view_kwargs):
    """Make additional work before to create your instance:
        - make checks
        - retrieve a related object to plug to the instance
        - compute an instance attribut
        - or what ever you want.
    """
    try:
        topic = self.session.query(Topic).filter_by(topic_id=view_kwargs['topic_id']).
↪one()
    except NoReultFOund:
        abort(404)

    data['topic'] = topic
    data['created'] = datetime.datetime.utcnow()


class PostList(ResourceList):
    class Meta:
        data_layer = {'cls': SqlalchemyDataLayer,
                      'kwargs': {'model': Post, 'session': sql_db.session},
                      'get_base_query': get_base_query,
                      'before_create_instance': before_create_instance}
```

```
    resource_type = 'post'
    schema = {'cls': PostSchema}
    endpoint = {'name': 'post_list'}
```

ResourceDetail

Instance attributes:

- model (Model): sqlalchemy model
- session (Session): sqlalchemy session instance
- id_field (str): the model identifier attribute name
- url_param_name (str): the name of the URL param in the route to retrieve value from

Class attributs:

- before_update_instance (callable) *Optional*: additional operations to run before updating an instance in the patch method
- before_delete_instance (callable) *Optional*: additional operations to run before deleting an instance in the delete method

Example:

```
from sqlalchemy import NoResultFound
from flask_rest_jsonapi import ResourceList, SqlalchemyDataLayer

from your_project.models import Post
from your_project.schemas import PostSchema
from your_project.extensions import sql_db


def before_update_instance(self, item, data, **view_kwargs):
    """Make additional work before to update your instance:
        - make checks
        - compute an instance attribut
        - or what ever you want.
    """
    data['updated_at'] = datetime.datetime.utcnow()


def before_delete_instance(self, data, **view_kwargs):
    """Make additional work before to delete your instance:
        - make checks
        - or what ever you want.
    """


class PostDetail(ResourceDetail):

    class Meta:
        data_layer = {'cls': SqlalchemyDataLayer,
                      'kwargs': {'session': sql_db.session,
                                 'model': Post,
                                 'id_field': 'post_id',
                                 'url_param_name': 'post_id'},
```

```
                        'before_update_instance': before_update_instance,
                        'before_delete_instance': before_delete_instance}

    resource_type = 'post'
    schema = {'cls': PostSchema}
```

Available opertations

All available operations on SQLAlchemy model fields (depending on the field type) can be used for filtering. See the SQLAlchemy documentation to learn more.

Mongo

A data layer around MongoDB

ResourceList

Instance attributes:

- collection (str): the mongodb collection name
- mongo: the mongodb connector
- model (type): the type of the document

Class attributs:

- get_base_query (callable): a callable to retrieve the base data in get method

Example:

```
from flask_rest_jsonapi import ResourceList, MongoDataLayer

from your_project.models import Post
from your_project.schemas import PostSchema
from your_project.extensions import mongo


def get_base_query(self, **view_kwargs):
    """Get base data filter
    """
    return {'topic_id': view_kwargs['topic_id']}


class PostList(ResourceList):

    class Meta:
        data_layer = {'cls': MongoDataLayer,
                      'kwargs': {'collection': 'logging',
                                 'model': dict,
                                 'mongo': mongo},
                      'get_base_query': get_base_query}

    resource_type = 'post'
    schema = {'cls': PostSchema}
    endpoint = {'name': 'post_list'}
```

ResourceDetail

Instance attributes:

- collection (str): the mongodb collection name
- mongo: the mongodb connector
- model (type): the type of the document
- id_field (str): the model identifier attribute name
- url_param_name (str): the name of the URL param in the route to retrieve value from

Example:

```
from flask_rest_jsonapi import ResourceList, MongoDataLayer

from your_project.models import Post
from your_project.schemas import PostSchema
from your_project.extensions import mongo


class PostDetail(ResourceDetail):

    class Meta:
        data_layer = {'cls': MongoDataLayer,
                      'kwargs': {'collection': 'post',
                                 'mongo': mongo,
                                 'model': dict,
                                 'id_field': 'post_id',
                                 'url_param_name': 'post_id'}}

    resource_type = 'post'
    schema = {'cls': PostSchema}
```

Available opertations

All available operations on mongodb fields (depending on the field type) can be used for filtering. See the MongoDB documentation to learn more.

Sorting

You can sort results with the "sort" querystring URL parameter.

Example (not urlencoded for readability):

```
GET /topics/1/posts?sort=-created,title HTTP/1.1
Content-Type: application/vnd.api+json
Accept: application/vnd.api+json
```

Fields restriction

You can retrieve only requested fields with the querystring URL parameter "fields"

Example (not urlencoded for readability):

```
GET /topics/1/posts?fields[post]=title,content HTTP/1.1
Content-Type: application/vnd.api+json
Accept: application/vnd.api+jsona
```

Filtering

You can filter results with the querystring url parameter "filter"

Example (not urlencoded for readability):

```
GET /topics/1/posts?filter[post]=[{"field":"created","op":"gt","value":"2016-11-10"}]␣
↪HTTP/1.1
Content-Type: application/vnd.api+json
Accept: application/vnd.api+json
```

You can add multiple filters but "or" expressions are not implemented yet. I will create a filtering system like Flask-Restless as soon as possible.

Multiple filter example:

```
GET /topics/1/posts?filter[post]=[{"field":"created","op":"gt","value":"2016-11-10"},{
↪"field":"title","op":"like","value":"%test%"}] HTTP/1.1
Content-Type: application/vnd.api+json
Accept: application/vnd.api+json
```

Available operations depend on the data layer chosen. Read the "Available operations" section of your data layer documentation to learn more.

Pagination

You can control the pagination with the querystring url parameter "page"

Example (not urlencoded for readability):

```
GET /topics/1/posts?page[size]=2 HTTP/1.1
Content-Type: application/vnd.api+json
Accept: application/vnd.api+json

GET /topics/1/posts?page[size]=2&page[number]=3 HTTP/1.1
Content-Type: application/vnd.api+json
Accept: application/vnd.api+json
```

If you want to disable pagination just set page size to 0

Example (not urlencoded for readability):

```
GET /topics/1/posts?page[size]=0 HTTP/1.1
Content-Type: application/vnd.api+json
Accept: application/vnd.api+json
```

Routing

Example:

```
from flask import Flask
from flask_rest_jsonapi import Api

from your_project.resources import TopicList, TopicDetail

app = Flask(__name__)
api = Api(app)

api.list_route('topic_list', /topics', resource_cls=TopicList)
api.detail_route('topic_detail', '/topics/<int:topic_id>', resource_cls=TopicDetail)
```

This routing example will create this site map:

url	method	endpoint
/topics	GET,POST	topic_list
/topics/<int:topic_id>	GET,PATCH,DELETE	topic_detail

You can add multiple URLs for the same resource:

```
from flask import Flask
from flask_rest_jsonapi import Api

from your_project.resources import TopicList

app = Flask(__name__)
api = Api(app)

api.list_route('topic_list', /topics', '/topic_list', resource_cls=TopicList)
```

Blueprint

your_project.views.py

```
from flask import Blueprint
from flask_rest_jsonapi import Api

from your_project.resources import TopicList

rest_api_bp = Blueprint('rest_api', __name__)
api = Api(rest_api_bp)

api.list_route('topic_list', /topics', resource_cls=TopicList)
```

your_project.app.py

```
from flask import Flask
from your_project.views import api

app = Flask(__name__)
api.init_app(app)
```

Flask extension

your_project.extensions.py

```
from flask_rest_jsonapi import Api

api = Api()
```

your_project.views.py

```
from your_project.resources import TopicList
from your_project.extensions import api

api.list_route('topic_list', /topics', resource_cls=TopicList)
```

your_project.app.py

```
from flask import Flask
from your_project.extensions import api

app = Flask(__name__)
api.init_app(app)
```

Resource configuration

You can directly configure your resources from the routing system. But I don't recommand to do that. I think it is better to organize your project with a strong separation between resources definitions and routing.

Example:

```
   api.list_route('topic_list',
                  '/topics',
                  resource_type='topic',
                  schema=TopicSchema,
                  data_layer=SqlalchemyDataLayer,
                  data_layer_kwargs={'model': Topic, 'session': session},
                  data_layer_additional_functions={'get_base_query': topic_get_base_
↪query})

But I think it is better to write code like that:
```

```
def get_base_query(self, **view_kwargs):
    return self.session.query(Topic)


class TopicResourceList(ResourceList):

    class Meta:
        data_layer = {'cls': SqlalchemyDataLayer,
                      'kwargs': {'model': Topic, 'session': sql_db.session},
                      'get_base_query': get_base_query}

    resource_type = 'topic'
    schema = {'cls': TopicSchema}
    endpoint = {'name': 'topic_list'}

api.list_route('topic_list', '/topics', resource_cls=TopicResourceList)
```

Tutorial

In this tutorial, we will cover a simple blog example with topic, post and author entities.

Note: I don't include imports on this tutorial for readability but you can see them in examples/full_example.py.

Note: All requests and responses are well formatted for better readability.

Initialize flask application, API and database

```
app = Flask(__name__)
api = Api(app)
engine = create_engine('sqlite:////tmp/test.db')
Session = sessionmaker(bind=engine)
session = Session()
```

Define and initialize models

```
Base = declarative_base()


class Topic(Base):
    __tablename__ = 'topic'

    id = Column(Integer, primary_key=True)
    name = Column(String)


class Post(Base):
    __tablename__ = 'post'

    id = Column(Integer, primary_key=True)
    title = Column(String)
    content = Column(String)
    topic_id = Column(Integer, ForeignKey('topic.id'))
    author_id = Column(Integer, ForeignKey('author.id'))

    topic = relationship("Topic", backref="posts")
    author = relationship("Author", backref="posts")


class Author(Base):
    __tablename__ = 'author'

    id = Column(Integer, primary_key=True)
    name = Column(String)
```

```
Base.metadata.create_all(engine)
```

Define marshamllow-jsonapi schemas

```
class TopicSchema(Schema):

    class Meta:
        type_ = 'topic'
        self_view = 'topic_detail'
        self_view_kwargs = {'topic_id': '<id>'}
        self_view_many = 'topic_list'

    id = fields.Str(dump_only=True)
    name = fields.Str(required=True)

    posts = Relationship(related_view='post_list',
                         related_view_kwargs={'topic_id': '<id>'},
                         many=True,
                         type_='post')


class PostSchema(Schema):

    class Meta:
        type_ = 'post'
        self_view = 'post_detail'
        self_view_kwargs = {'post_id': '<id>'}

    id = fields.Str(dump_only=True)
    title = fields.Str(required=True)
    content = fields.Str()
    author_name = fields.Function(lambda obj: obj.author.name)
    author_id = fields.Int(required=True)

    topic = Relationship(related_view='topic_detail',
                         related_view_kwargs={'topic_id': '<topic.id>'},
                         type_='topic')

    author = Relationship(related_view='author_detail',
                          related_view_kwargs={'author_id': '<author.id>'},
                          type_='author')


class AuthorSchema(Schema):

    class Meta:
        type_ = 'author'
        self_view = 'author_detail'
        self_view_kwargs = {'author_id': '<id>'}
        self_view_many = 'author_list'

    id = fields.Str(dump_only=True)
    name = fields.Str(required=True)

    posts = Relationship(related_view='post_list',
```

```
                            related_view_kwargs={'author_id': '<id>'},
                            many=True,
                            type_='post')
```

Register resources and routes

```
def topic_get_base_query(self, **view_kwargs):
    return self.session.query(Topic)

api.list_route('topic_list',
               '/topics',
               resource_type='topic',
               schema=TopicSchema,
               data_layer=SqlalchemyDataLayer,
               data_layer_kwargs={'model': Topic, 'session': session},
               data_layer_additional_functions={'get_base_query': topic_get_base_
↪query})

api.detail_route('topic_detail',
                 '/topics/<int:topic_id>',
                 resource_type='topic',
                 schema=TopicSchema,
                 data_layer=SqlalchemyDataLayer,
                 data_layer_kwargs={'model': Topic, 'session': session, 'id_field':
↪'id', 'url_param_name': 'topic_id'})


def post_get_base_query(self, **view_kwargs):
    query = self.session.query(Post)

    if view_kwargs.get('topic_id'):
        query = query.join(Topic).filter_by(id=view_kwargs['topic_id'])
    elif view_kwargs.get('author_id'):
        query = query.join(Author).filter_by(id=view_kwargs['author_id'])

    return query


def post_before_create_instance(self, data, **view_kwargs):
    try:
        topic = self.session.query(Topic).filter_by(id=str(view_kwargs['topic_id'])).
↪one()
    except NoResultFound:
        return ErrorFormatter.format_error(['Topic not found']), 404

    data['topic'] = topic

api.list_route('post_list',
               '/topics/<int:topic_id>/posts',
               '/authors/<int:author_id>/posts',
               resource_type='post',
               schema=PostSchema,
               schema_get_kwargs={'exclude': ('author_id', )},
               schema_post_kwargs={'exclude': ('author_name', )},
               data_layer=SqlalchemyDataLayer,
               data_layer_kwargs={'model': Post, 'session': session},
```

```
                data_layer_additional_functions={'get_base_query': post_get_base_query,
                                                 'before_create_instance': post_before_
↪create_instance},
                endpoint_include_view_kwargs=True)

api.detail_route('post_detail',
                 '/posts/<int:post_id>',
                 resource_type='post',
                 schema=PostSchema,
                 schema_get_kwargs={'exclude': ('author_id', )},
                 data_layer=SqlalchemyDataLayer,
                 data_layer_kwargs={'model': Post, 'session': session, 'id_field': 'id
↪', 'url_param_name': 'post_id'})


def author_get_base_query(self, **view_kwargs):
    return self.session.query(Author)

api.list_route('author_list',
               '/authors',
               resource_type='author',
               schema=AuthorSchema,
               data_layer=SqlalchemyDataLayer,
               data_layer_kwargs={'model': Author, 'session': session},
               data_layer_additional_functions={'get_base_query': author_get_base_
↪query})

api.detail_route('author_detail',
                 '/authors/<int:author_id>',
                 resource_type='author',
                 schema=AuthorSchema,
                 data_layer=SqlalchemyDataLayer,
                 data_layer_kwargs={'model': Author,
                                    'session': session,
                                    'id_field': 'id',
                                    'url_param_name': 'author_id'})
```

If you want to separate resource configuration from routing, you can do something like that:

```
def get_base_query(self, **view_kwargs):
    return self.session.query(Topic)


class TopicResourceList(ResourceList):

    class Meta:
        data_layer = {'cls': SqlalchemyDataLayer,
                      'kwargs': {'model': Topic, 'session': sql_db.session},
                      'get_base_query': get_base_query}

    resource_type = 'topic'
    schema = {'cls': TopicSchema}
    endpoint = {'name': 'topic_list'}

api.list_route('topic_list', '/topics', resource_cls=TopicResourceList)
```

List topics

Request:

```
$ curl "http://127.0.0.1:5000/topics" -H "Content-Type: application/vnd.api+json"\
  -H "Accept: application/vnd.api+json"
```

Response:

```
{
    "data": [],
    "links": {
        "first": "/topics",
        "last": "/topics",
        "self": "/topics"
    }
}
```

Create topic

Request:

```
$ curl "http://127.0.0.1:5000/topics" -X POST\
  -H "Content-Type: application/vnd.api+json"\
  -H "Accept: application/vnd.api+json"\
  -d '{
    "data": {
      "type": "topic",
      "attributes": {
        "name": "topic 1"
      }
    }
  }'
```

Response:

```
{
    "data": {
        "attributes": {
            "name": "topic 1"
        },
        "id": "1",
        "links": {
            "self": "/topics/1"
        },
        "relationships": {
            "posts": {
                "links": {
                    "related": "/topics/1/posts"
                }
            }
        },
        "type": "topic"
    },
    "links": {
        "self": "/topics/1"
```

```
    }
}
```

Now you can list again topics:

Request:

```
$ curl "http://127.0.0.1:5000/topics" -H "Content-Type: application/vnd.api+json"\
  -H "Accept: application/vnd.api+json"
```

Response:

```
{
    "data": [
        {
            "attributes": {
                "name": "topic 1"
            },
            "id": "1",
            "links": {
                "self": "/topics/1"
            },
            "relationships": {
                "posts": {
                    "links": {
                        "related": "/topics/1/posts"
                    }
                }
            },
            "type": "topic"
        }
    ],
    "links": {
        "first": "/topics",
        "last": "/topics?page%5Bnumber%5D=1",
        "self": "/topics"
    }
}
```

Update topic

Request:

```
$ curl "http://127.0.0.1:5000/topics/1" -X PATCH\
  -H "Content-Type: application/vnd.api+json"\
  -H "Accept: application/vnd.api+json"\
  -d '{
    "data": {
      "type": "topic",
      "id": "1",
      "attributes": {
        "name": "topic 1 updated"
      }
    }
  }'
```

Response:

```
{
    "data": {
        "attributes": {
            "name": "topic 1 updated"
        },
        "id": "1",
        "links": {
            "self": "/topics/1"
        },
        "relationships": {
            "posts": {
                "links": {
                    "related": "/topics/1/posts"
                }
            }
        },
        "type": "topic"
    },
    "links": {
        "self": "/topics/1"
    }
}
```

Delete topic

Request:

```
$ curl "http://127.0.0.1:5000/topics/1" -X DELETE\
  -H "Content-Type: application/vnd.api+json"\
  -H "Accept: application/vnd.api+json"
```

Create author

Request:

```
$ curl "http://127.0.0.1:5000/authors" -X POST\
  -H "Content-Type: application/vnd.api+json"\
  -H "Accept: application/vnd.api+json"\
  -d '{
    "data": {
      "type": "author",
      "attributes": {
        "name": "John Smith"
      }
    }
  }'
```

Response:

```
{
    "data": {
        "attributes": {
            "name": "John Smith"
        },
        "id": "1",
```

```
        "links": {
            "self": "/authors/1"
        },
        "relationships": {
            "posts": {
                "links": {
                    "related": "/authors/1/posts"
                }
            }
        },
        "type": "author"
    },
    "links": {
        "self": "/authors/1"
    }
}
```

Create post with an author in a topic

Before creating a post, we have to create a topic (because we have deleted the only one previously)

Request:

```
$ curl "http://127.0.0.1:5000/topics" -X POST\
  -H "Content-Type: application/vnd.api+json"\
  -H "Accept: application/vnd.api+json"\
  -d '{
    "data": {
      "type": "topic",
      "attributes": {
        "name": "topic 1"
      }
    }
  }'
```

Response:

```
{
    "data": {
        "attributes": {
            "name": "topic 1"
        },
        "id": "1",
        "links": {
            "self": "/topics/1"
        },
        "relationships": {
            "posts": {
                "links": {
                    "related": "/topics/1/posts"
                }
            }
        },
        "type": "topic"
    },
    "links": {
        "self": "/topics/1"
```

```
    }
}
```

Now we have a new topic, so let's create a post for it

Request:

```
$ curl "http://127.0.0.1:5000/topics/1/posts" -X POST\
  -H "Content-Type: application/vnd.api+json"\
  -H "Accept: application/vnd.api+json"\
  -d '{
    "data": {
      "type": "post",
      "attributes": {
        "title": "post 1",
        "content": "content of the post 1",
        "author_id": "1"
      }
    }
  }'
```

Response:

```
{
    "data": {
        "attributes": {
            "author_id": 1,
            "content": "content of the post 1",
            "title": "post 1"
        },
        "id": "1",
        "links": {
            "self": "/posts/1"
        },
        "relationships": {
            "author": {
                "links": {
                    "related": "/authors/1"
                }
            },
            "topic": {
                "links": {
                    "related": "/topics/2"
                }
            }
        },
        "type": "post"
    },
    "links": {
        "self": "/posts/1"
    }
}
```

List posts of topic 1

Request:

```
$ curl "http://127.0.0.1:5000/topics/1/posts" -H "Content-Type: application/vnd.
↪api+json"\
  -H "Accept: application/vnd.api+json"
```

Response:

```
{
    "data": [
        {
            "attributes": {
                "author_name": "John Smith",
                "content": "content of the post 1",
                "title": "post 1"
            },
            "id": "1",
            "links": {
                "self": "/posts/1"
            },
            "relationships": {
                "author": {
                    "links": {
                        "related": "/authors/1"
                    }
                },
                "topic": {
                    "links": {
                        "related": "/topics/1"
                    }
                }
            },
            "type": "post"
        }
    ],
    "links": {
        "first": "/topics/1/posts",
        "last": "/topics/1/posts?page%5Bnumber%5D=1",
        "self": "/topics/1/posts"
    }
}
```

List posts of author 1 (John Smith)

Request:

```
$ curl "http://127.0.0.1:5000/authors/1/posts" -H "Content-Type: application/vnd.
↪api+json"\
  -H "Accept: application/vnd.api+json"
```

Response:

```
{
    "data": [
        {
            "attributes": {
                "author_name": "John Smith",
                "content": "content of the post 1",
```

```
                "title": "post 1"
            },
            "id": "1",
            "links": {
                "self": "/posts/1"
            },
            "relationships": {
                "author": {
                    "links": {
                        "related": "/authors/1"
                    }
                },
                "topic": {
                    "links": {
                        "related": "/topics/1"
                    }
                }
            },
            "type": "post"
        }
    ],
    "links": {
        "first": "/authors/1/posts",
        "last": "/authors/1/posts?page%5Bnumber%5D=1",
        "self": "/authors/1/posts"
    }
}
```

CHAPTER 2

Api reference

- genindex
- modindex