invenio-rest Documentation

Release 1.0.0

CERN

May 14, 2018

Contents

REST API module for Invenio.

Invenio-REST takes care of installing basic error handling on a Flask API application, as well as initializing Flask-CORS for Cross-Origin Resources Sharing (not enabled by default).

Further documentation is available on https://invenio-rest.readthedocs.io/

CHAPTER 1

User's Guide

This part of the documentation will show you how to get started in using Invenio-REST.

1.1 Installation

Invenio-REST is on PyPI so all you need is:

```
$ pip install invenio-rest
```

1.2 Configuration

Invenio REST configuration.

Please also see Flask-CORS for many more configuration options.

invenio_rest.config.**CORS_EXPOSE_HEADERS = [u'ETag', u'Link', u'X-RateLimit-Limit', u'X-Rate**
Expose the following headers.

> **Note:** Overwrites Flask-CORS configuration.

invenio_rest.config.**CORS_RESOURCES = u'*'**
Dictionary for configuring CORS for endpoints.

See Flask-CORS for further details.

> **Note:** Overwrites Flask-CORS configuration.

invenio_rest.config.**CORS_SEND_WILDCARD = True**
Sending wildcard CORS header.

Note: Overwrites Flask-CORS configuration.

invenio_rest.config.**REST_ENABLE_CORS = False**
Enable CORS configuration. (Default: `False`)

invenio_rest.config.**REST_MIMETYPE_QUERY_ARG_NAME = None**

Name of the query argument to specify the mimetype wanted for the output. Set it to None to disable.

Note: You can customize the query argument name by specifying it as a string:

```
REST_MIMETYPE_QUERY_ARG_NAME = 'format'
```

With this value, the url will be:

```
/api/record/<id>?format=<value>
```

You can set the accepted values passing a dictionary to the key *record_serializers_aliases*:

```
record_serializers_aliases={
   'json': 'application/json',
   'marc21': 'application/marcxml+xml'
}
```

1.3 Usage

REST API module for Invenio.

Invenio-REST takes care of installing basic error handling on a Flask API application, as well as initializing Flask-Limiter for rate limiting and Flask-CORS for Cross-Origin Resources Sharing (not enabled by default). It also acts as orchestrator to take actions depending on the request headers, such as dealing with ETag/If-Modified-Since cache header or Content-Type/Accept header to resolve the right content serializer.

1.3.1 Initialization

First create a Flask application:

```
>>> from flask import Flask
>>> app = Flask('myapp')
```

Next, initialize your extension:

```
>>> from invenio_rest import InvenioREST
>>> InvenioREST(app)
<invenio_rest.ext.InvenioREST ...>
```

1.3.2 Serializers

Let's create 2 serializers that will return our answer to the correct format. For instance, our server will be able to either answer in JSON or in XML:

```
>>> import xmltodict
>>> from flask import jsonify, make_response
>>> def json_v1_search(search_result):
...     return make_response(jsonify(search_result))
>>> def xml_v1_search(search_result):
...     return make_response(xmltodict.unparse((search_result,)))
```

1.3.3 Views

Now we create our view that will handle the requests and return the serialized response based on the request's headers or using the default media type. To do so, we need to create a class that inherits *ContentNegotiatedMethodView*. In the constructor, we register our two serializers, and we create a *get* method for the *GET* requests:

```
>>> from invenio_rest import ContentNegotiatedMethodView
>>> class RecordsListResource(ContentNegotiatedMethodView):
...     def __init__(self, **kwargs):
...         super(RecordsListResource, self).__init__(
...             method_serializers={
...                 'GET': {
...                     'application/json': json_v1_search,
...                     'application/xml': xml_v1_search,
...                 },
...             },
...             default_method_media_type={
...                 'GET': 'application/json',
...             },
...             default_media_type='application/json',
...             **kwargs)
...     def get(self, **kwargs):
...         return {"title": "Test"}
```

To finish, we need to create a blueprint that defines an endpoint (here */records*) and that registers our class:

```
>>> from flask import Blueprint
>>> blueprint = Blueprint(
...     'mymodule',
...     'myapp',
...     url_prefix='/records',
...     template_folder='templates',
...     static_folder='static',
... )
>>> records_view = RecordsListResource.as_view('records')
>>> blueprint.add_url_rule('/', view_func=records_view)
>>> app.register_blueprint(blueprint)
```

Now you can launch your server and request it on the */records* endpoint, as described in *Example application*.

1.3.4 Building REST APIs for Invenio

Following is a quick overview over which tools we are currently using for building REST APIs. Before implementing your REST API, do take a look at some of the existing REST APIs already implemented in Invenio to get some inspiration.

In Invenio we have decided not to use some of the existing Flask extensions for building REST APIs since mostly these extensions are not very flexible and there are many existing Python libraries that do a much better job at the individual tasks.

Flask application

Invenio's REST API is running in its own Flask application. This ensures that the REST API can run on machines independently of the UI application and also ensures that e.g. error handling, that it can be independently versioned, is much simpler compared to having a mixed REST API/UI application.

Views

Views for REST APIs are built using standard Flask blueprints. We use `MethodView` for HTTP method based dispatching, and in particular we use Invenio-REST's subclass *`ContentNegotiatedMethodView`* which takes care of selecting the right serializer based on HTTP content negotiation.

Versioning

Versioning of the REST API is primarily achieved through HTTP content negotiation. I.e. you define a new MIME type that your clients explicitly request (using *`ContentNegotiatedMethodView`*).

Serialization/Deserialization

Invenio-REST provides 2 ways to define how to serialize the response content:

- *Query argument*: if the request query contains a well configured parameter, for example *format* (see the config *REST_MIMETYPE_QUERY_ARG_NAME*), then the serializer associated to the value of that argument will be used. With a request like:

  ```
  /api/record/<id>?format=custom_json
  ```

  and a defined mapping like:

  ```
  record_serializers_aliases={
      'custom_json': 'application/json',
      'marc21': 'application/marcxml+xml'
  }
  ```

  the output will be serialized using the serializer that is associated to the mimetype *application/json*.

- *Headers*: if the query argument is missing or disabled, then the headers *Accept* or *Content-Type* are parsed to resolve the serializer to use. The expected value for the header is the mimetype:

  ```
  Accept: application/json
  ```

  again, the serializer associated to that mimetype will be used to format the output.

For serialization/deserialization we primarily use Marshmallow which takes care that REST API endpoints are supplied with proper argument types such as list of strings or integers, or ensuring that e.g. timestamps are ISO8601 formatted in UTC when serializing to JSON, and correctly deserialize timestamps into Python datetime objects.

Invenio-Records-REST is currently the most advanced example of using serializers with both JSON, XML and text output.

Request parameters parsing

Request parameters in the URL or JSON are most often handled with the library webargs.

Error handling

Invenio-REST provides some default exceptions which you can subclass, which when thrown, will render a proper REST API response for the error to the client (see e.g. *RESTException*).

Headers (security, CORS and rate limiting)

Invenio-App is responsible for installing Flask-Tailsman which sets many important security related headers as well as Flask-Limiter which provides rate limiting.

Invenio-REST is responsible for installing Flask-CORS which provides support for Cross-Origin Resource Sharing.

Invenio-OAuth2Server along with Invenio-Accounts is responsible for providing API authentication based on OAuth 2.0 as well as protecting against CRSF-attacks in the REST API.

1.4 Example application

First install Invenio-REST, setup the application and load fixture data by running:

```
$ pip install -e .[all]
$ cd examples
$ ./app-setup.sh
$ ./app-fixtures.sh
```

Next, start the development server:

```
$ export FLASK_APP=app.py FLASK_DEBUG=1
$ flask run
```

and use cURL to explore the simplistic REST API:

```
$ curl -v -XGET http://0.0.0.0:5000/records/
$ curl -v -XGET http://0.0.0.0:5000/records/ \\
    -H Accept:application/xml
```

The example app demonstrates:

- Use of `Accept` headers to change the serialization from JSON to XML via the *invenio_rest.views.ContentNegotiatedMethodView*.
- CORS headers (`Access-Control-Allow-Origin` and `Access-Control-Expose-Headers`).

To reset the example application run:

```
$ ./app-teardown.sh
```

CHAPTER 2

API Reference

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API Docs

REST API module for Invenio.

class invenio_rest.ext.**InvenioREST**(*app=None*)
Invenio-REST extension.

Extension initialization.

Parameters **app** – An instance of `flask.Flask`.

init_app(*app*)
Flask application initialization.

Initialize the Rate-Limiter, CORS and error handlers.

Parameters **app** – An instance of `flask.Flask`.

init_config(*app*)
Initialize configuration.

Note: Change Flask-CORS and Flask-Limiter defaults.

Parameters **app** – An instance of `flask.Flask`.

2.1.1 Decorators

Decorators for testing certain assertions.

invenio_rest.decorators.**require_content_types**(**allowed_content_types*)
Decorator to test if proper Content-Type is provided.

Parameters ***allowed_content_types** – List of allowed content types.

Raises ***invenio_rest.errors.InvalidContentType*** – It's rised if a content type not allowed is required.

2.1.2 Errors

Exceptions used in Invenio REST module.

class invenio_rest.errors.**FieldError**(*field*, *message*, *code=None*)
Represents a field level error.

Note: This is not an actual exception.

Init object.

Parameters

- **field** – Field name.
- **message** – The text message to show.
- **code** – The HTTP status to return. (Default: None)

to_dict()
Convert to dictionary.

Returns A dictionary with field, message and, if initialized, the HTTP status code.

exception invenio_rest.errors.**InvalidContentType**(*allowed_content_types=None*, ***kwargs*)
Error for when an invalid Content-Type is provided.

Initialize exception.

code = 415
HTTP Status code.

exception invenio_rest.errors.**RESTException**(*errors=None*, ***kwargs*)
HTTP Exception delivering JSON error responses.

Initialize RESTException.

get_body(*environ=None*)
Get the request body.

get_description(*environ=None*)
Get the description.

get_errors()
Get errors.

Returns A list containing a dictionary representing the errors.

get_headers(*environ=None*)
Get a list of headers.

exception invenio_rest.errors.**RESTValidationError**(*errors=None*, ***kwargs*)
A standard REST validation error.

Initialize RESTException.

code = 400
HTTP Status code.

description = 'Validation error.'
Error description.

exception invenio_rest.errors.**SameContentException**(*etag*, *last_modified=None*, ***kwargs*)
304 Same Content exception.

Exception thrown when request is GET or HEAD, ETag is If-None-Match and one or more of the ETag values match.

Constructor.

Parameters

- **etag** – matching etag
- **last_modified** – The last modefied date. (Default: `None`)

code = 304
HTTP Status code.

description = 'Same Content.'
Error description.

get_response(*environ=None*)
Get a list of headers.

2.1.3 Views

REST API module for Invenio.

class invenio_rest.views.**ContentNegotiatedMethodView**(*serializers=None*, *method_serializers=None*, *serializers_query_aliases=None*, *default_media_type=None*, *default_method_media_type=None*, **args*, ***kwargs*)
MethodView with content negotiation.

Dispatch HTTP requests as MethodView does and build responses using the registered serializers. It chooses the right serializer using the request's accept type. It also provides a helper method for handling ETags.

Register the serializing functions.

Serializing functions will receive all named and non named arguments provided to `make_response` or returned by request handling methods. Recommended prototype is: `serializer(data, code=200, headers=None)` and it should return `flask.Response` instances.

Serializing functions can also be overridden by setting `self.serializers`.

Parameters

- **serializers** – A mapping from mediatype to a serializer function.

- **method_serializers** – A mapping of HTTP method name (GET, PUT, PATCH, POST, DELETE) -> dict(mediatype -> serializer function). If set, it overrides the serializers dict.
- **serializers_query_aliases** – A mapping of values of the defined query arg (see *config.REST_MIMETYPE_QUERY_ARG_NAME*) to valid mimetypes: dict(alias -> mimetype).
- **default_media_type** – Default media type used if no accept type has been provided and global serializers are used for the request. Can be None if there is only one global serializer or None. This media type is used for method serializers too if `default_method_media_type` is not set.
- **default_method_media_type** – Default media type used if no accept type has been provided and a specific method serializers are used for the request. Can be `None` if the method has only one serializer or `None`.

check_etag(*etag*, *weak=False*)
Validate the given ETag with current request conditions.

Compare the given ETag to the ones in the request header If-Match and If-None-Match conditions.

The result is unspecified for requests having If-Match and If-None-Match being both set.

Parameters **etag** (*str*) – The ETag of the current resource. For PUT and PATCH it is the one before any modification of the resource. This ETag will be tested with the Accept header conditions. The given ETag should not be quoted.

Raises

- **werkzeug.exceptions.PreconditionFailed** – If the condition is not met.
- ***invenio_rest.errors.SameContentException*** – If the the request is GET or HEAD and the If-None-Match condition is not met.

check_if_modified_since(*dt*, *etag=None*)
Validate If-Modified-Since with current request conditions.

dispatch_request(**args*, ***kwargs*)
Dispatch current request.

Dispatch the current request using `flask.views.MethodView` *dispatch_request()* then, if the result is not already a `flask.Response`, search for the serializing function which matches the best the current request's Accept header and use it to build the `flask.Response`.

Return type `flask.Response`

Raises **werkzeug.exceptions.NotAcceptable** – If no media type matches current Accept header.

Returns The response returned by the request handler or created by the serializing function.

get_method_serializers(*http_method*)
Get request method serializers + default media type.

Grab serializers from `method_serializers` if defined, otherwise returns the default serializers. Uses GET serializers for HEAD requests if no HEAD serializers were specified.

The method also determines the default media type.

Parameters **http_method** – HTTP method as a string.

Returns Tuple of serializers and default media type.

make_response(**args*, ***kwargs*)
Create a Flask Response.

Dispatch the given arguments to the serializer best matching the current request's Accept header.

Returns The response created by the serializing function.

Return type `flask.Response`

Raises **werkzeug.exceptions.NotAcceptable** – If no media type matches current Accept header.

match_serializers(*serializers*, *default_media_type*)
Choose serializer for a given request based on query arg or headers.

Checks if query arg *format* (by default) is present and tries to match the serializer based on the arg value, by resolving the mimetype mapped to the arg value. Otherwise, chooses the serializer by retrieving the best quality *Accept* headers and matching its value (mimetype).

Parameters

- **serializers** – Dictionary of serializers.
- **default_media_type** – The default media type.

Returns Best matching serializer based on *format* query arg first, then client *Accept* headers or None if no matching serializer.

invenio_rest.views.**create_api_errorhandler**(***kwargs*)
Create an API error handler.

E.g. register a 404 error:

```
app.errorhandler(404)(create_api_errorhandler(
    status=404, message='Not Found'))
```

Parameters ****kwargs** – It contains the `'status'` and the `'message'` to describe the error.

CHAPTER 3

Additional Notes

Notes on how to contribute, legal information and changes are here for the interested.

3.1 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

3.1.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/inveniosoftware/invenio-rest/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Invenio-REST could always use more documentation, whether as part of the official Invenio-REST docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/inveniosoftware/invenio-rest/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

3.1.2 Get Started!

Ready to contribute? Here's how to set up *invenio-rest* for local development.

1. Fork the *inveniosoftware/invenio-rest* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/invenio-rest.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv invenio-rest
$ cd invenio-rest/
$ pip install -e .[all]
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass tests:

```
$ ./run-tests.sh
```

   The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -s
    -m "component: title without verbs"
    -m "* NEW Adds your new feature."
    -m "* FIX Fixes an existing issue."
    -m "* BETTER Improves and existing feature."
    -m "* Changes something that should not be visible in release notes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

3.1.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5. Check https://travis-ci.org/inveniosoftware/invenio-rest/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.0.0 (released 2018-03-23)

- Initial public release.

3.3 License

Note: In applying this license, CERN does not waive the privileges and immunities granted to it by virtue of its status as an Intergovernmental Organization or submit itself to any jurisdiction.

3.4 Contributors

- Alexander Ioannidis
- Alizee Pace
- Chiara Bigarella
- Harri Hirvonsalo
- Harris Tzovanakis

- Jacopo Notarstefano
- Jiri Kuncar
- Lars Holm Nielsen
- Leonardo Rossi
- Nicola Tarocco
- Nicolas Harraudeau
- Nikos Filippakis
- Rémi Ducceschi
- Sami Hiltunen
- Sebastian Witowski
- Tibor Simko

Python Module Index

i

Index

S

T