invenio-jsonschemas Documentation

Release 1.0.0

CERN

Mar 21, 2018

Contents

Invenio module for building and serving JSONSchemas.

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

CHAPTER 1

User's Guide

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

1.1 Installation

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

```
$ pip install invenio-jsonschemas
```

1.2 Configuration

Default configuration.

invenio_jsonschemas.config.**JSONSCHEMAS_ENDPOINT = '/schemas'**
Default schema endpoint.

invenio_jsonschemas.config.**JSONSCHEMAS_HOST = 'localhost'**
Default json schema host.

invenio_jsonschemas.config.**JSONSCHEMAS_LOADER_CLS = None**
Loader class used in `JSONRef` when replacing `$ref`.

invenio_jsonschemas.config.**JSONSCHEMAS_REGISTER_ENDPOINTS_API = True**
Register the endpoints on the API app.

invenio_jsonschemas.config.**JSONSCHEMAS_REGISTER_ENDPOINTS_UI = True**
Register the endpoints on the UI app.

invenio_jsonschemas.config.**JSONSCHEMAS_REPLACE_REFS = False**
Whether to resolve $ref before serving a schema.

invenio_jsonschemas.config.**JSONSCHEMAS_RESOLVER_CLS = 'invenio_jsonschemas.utils.resolve_s**
Resolver used to resolve the schema.

if *invenio_jsonschemas.config.JSONSCHEMAS_RESOLVE_SCHEMA* is `True` or there is `?resolved=1` parameter on the request the resolver will run over the schema. This can be used for custom schemas resolver.

invenio_jsonschemas.config.**JSONSCHEMAS_RESOLVE_SCHEMA = False**
Whether to resolve schema using the Resolver Class.

If is `True`, will replace $ref and run the *invenio_jsonschemas.config.JSONSCHEMAS_RESOLVER_CLS* class before serving a schema.

invenio_jsonschemas.config.**JSONSCHEMAS_URL_SCHEME = 'https'**
Default url scheme for schemas.

1.3 Usage

Invenio-JSONSchemas is a module for building and serving JSON Schemas.

Using this module one can:

- Define JSON Schemas and expose them under a `/schemas` endpoint.
- Validate data using locally defined and/or external JSON Schemas.
- Resolve complex schemas and expand their references (`$ref`) or `allOf` tags for usage with other libraries that do not support them.

1.3.1 JSON Schema basics

Using JSON Schemas is a popular way to define and make publicly available the internal structure of complex entities being used inside an application. Since Invenio is a digital library framework, dealing with entities that contain complex metadata, like for example bibliographic records, is common and if not handled properly can lead to data inconsistencies.

We will not attempt to explain in detail how JSON Schemas are defined and work, since there are much more thorough resources available in the official JSON Schema website. Having a basic knowledge of them though is recommended in order to understand what this module provides on top of them.

Here is a basic JSON Schema defining the structure of a bibliographic record with a `title`, an optional `description`, a list of `creators` and a `publication_year`:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "id": "https://myapp.org/schemas/record.json",
    "type": "object",
    "properties": {
        "title": { "type": "string" },
        "description": { "type": "string" },
        "creators": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "name": { "type": "string" }
                },
                "required": ["name"]
            }
        },
```

```
        "publication_year": { "type": "integer" }
    },
    "required": ["title", "creators", "publication_year"]
}
```

A valid record for this JSON Schema would be the following:

```
{
    "title": "This is a record title",
    "creators": [ { "name": "Doe, John"}, { "name": "Roe, Jane" } ],
    "publication_year": 2018
}
```

1.3.2 Initialization

First create a Flask application:

```
>>> from flask import Flask
>>> app = Flask(__name__)
```

Configuration

Before we initialize the `InvenioJSONSchemas` extension, we need to configure which is our application's host. This will help with automatically skipping additional HTTP requests when fetching locally defined JSON Schemas. More about how this works is described in *Composable schemas with JSONRef*.

```
>>> # If your website's host is e.g. "myapp.org"
>>> app.config['JSONSCHEMAS_HOST'] = 'myapp.org'
```

Last, but not least, let's initialize the extension:

```
>>> from invenio_jsonschemas import InvenioJSONSchemas
>>> ext = InvenioJSONSchemas(app)
```

Setuptools integration

The above steps didn't actually register any JSON Schemas. In order for your JSON Schemas to be registered you must specify in your package's `setup.py` an entry point item in the `invenio_jsonschemas.schemas` group, pointing to a Python module where the actual JSON Schema `.json` files are placed. Invenio-JSONSchemas then takes care of loading them automatically during application initialization.

By default the extension loads from entrypoint group name `invenio_jsonschemas.schemas` but you can change that as shown below:

```
ext = InvenioJSONSchemas(app, entry_point_group=<entrypoint_group_name>)
```

1.3.3 Registering JSON Schemas

Here is a directory structure containing two JSON Schemas, `biology/animal_record_schema.json` and `record_schema.json`, taken from this module's example application:

```
$ tree --dirsfirst invenio-jsonschemas/examples/samplepkg

invenio-jsonschemas/examples/samplepkg
├── samplepkg
│   ├── jsonschemas
│   │   ├── biology
│   │   │   └── animal_record_schema.json
│   │   ├── __init__.py
│   │   └── record_schema.json
│   └── __init__.py
└── setup.py
```

The first thing in order to use `invenio_jsonschemas` is to register your folder that holds your schemas. To do so you have to include the entrypoint to your package that points to your schema folder as shown below:

```
# invenio-jsonschemas/examples/samplepkg/setup.py
...
entry_points={
    'invenio_jsonschemas.schemas': [
        'samplepkg = samplepkg.jsonschemas'  # path to your schema folder
    ],
},
...
```

After registering your schemas folder the extension knows where to find your schemas and how to load them. The extension loads every schema that is under `{JSONSCHEMAS_HOST}/{JSONSCHEMAS_ENDPOINT}` by fetching it locally and not making a network request. This means that reads directly the file of your schema. Also, the same happens if you have inside of your schema a `$ref` field pointing to the same url format. You can see the function called when the schema url is requested here.

1.3.4 Exposing JSON Schemas

You can enable/disable the endpoint that serves your schemas during the initialization of `InvenioJsonSchemas` extension by passing the parameter `register_config_blueprint`. This parameter points to your configuration variable that controls the serving of the schemas. So, if you want to disable the schemas serving you can do it as shown below:

Also by default the schema endpoint will be prefixed by `/schemas`. If you want to change that you can change the `JSONSCHEMAS_ENDPOINT` configuration variable. For more available configuration options see the *Configuration*.

For the above example the two schemas would we available under:

- `https://myapp.org/schemas/record_schema.json`, and
- `https://myapp.org/schemas/biology/animal_record_schema.json`

1.3.5 Composable schemas with JSONRef

A JSON Schema can be a fully fleshed out schema, composed of only the "primitive" types provided in the specification, but usually this is impractical when there are sub-entities that are repeated throughout the schema. For that reason JSON Schema provides `$ref` fields which can point to internal or external schemas. See such an example below:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "id": "https://myapp.org/schemas/record.json",
    "definitions": {
```

```
        "location": {
            "type": "object",
            "properties": {
                "city": { "stype": "string" },
                "country": { "stype": "string" },
                "address": { "stype": "string" }
            }
        }
    },
    "type": "object",
    "properties": {
        "title": { "type": "string" },
        "creator": { "$ref": "https://foo.org/schemas/person.json" },
        "origin": { "$ref": "#/definitions/location" }
    }
}
```

Invenio-JSONSchemas provides the ability to serve the fully resolved schema or the compact version including one or many `$ref` fields. The way to tell the extension to serve the resolved schema is either by passing the querystring parameter `refs=1` when fetching a schema or by setting the `JSONSCHEMAS_REPLACE_REFS` configuration variable to `True`. Internally the module uses the JsonRef package for resolving the references in the schema.

If you make a request to `GET https://myapp.org/schemas/record.json?refs=1`, you will get something similar to:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "id": "https://myapp.org/schemas/record.json",
    "definitions": { ... },
    "type": "object",
    "properties": {
        "title": { "type": "string" },
        "creator": {
            // person.json schema will be fetched and expanded here
            ...
         },
        "origin": {
            "type": "object",
            "properties": {
                "city": { "stype": "string" },
                "country": { "stype": "string" },
                "address": { "stype": "string" }
            }
        }
    }
}
```

The module also expands the `allOf` tags when the `resolved=1` querystring parameter is passed or the `JSONSCHEMAS_RESOLVE_SCHEMA` configuration variable is set to `True`. A schema example that includes the `allOf` tag can be shown below:

```
...
"id": "https://myapp.org/schemas/record.json",
"allOf": [
    { "properties": { "title": { "type": "string" } } },
    { "properties": { "status": { "enum": [ "published", "draft" ] } } }
]
...
```

If you make a request to `GET https://myapp.org/schemas/record.json?resolved=1` you would get a response in the following format:

```
...
// The "allOf" items have been merged in a single object
"properties": {
    "title": { "type": "string" },
    "status": { "enum": [ "published", "draft" ] }
}
...
```

Note on storing absolute URLs

As discussed in this issue, it is not recommended to store and expose absolute URLs in the `$ref`, as they can change in the future. One should instead try to use DOI/EPIC or other kind of identifiers with the certitude that they will never change, to avoid broken references.

1.3.6 Using with Invenio-Records

Invenio-JSONSchemas includes an `invenio_records.jsonresolver` entry point item which registers a JSONResolver plugin for Invenio-Records. This basically means that records that are being validated against schemas that include `$ref` s to locally defined schemas won't do an HTTP request to fetch these schemas, and resolve them locally. You can read more about record validation in the documentation of Invenio-Records.

1.4 Example application

Run example development server:

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

Open the schema from web:

```
$ curl http://localhost:5000/schemas/record_schema.json
$ curl http://localhost:5000/schemas/biology/animal_record_schema.json
```

Teardown the application:

```
$ ./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

Invenio module for building and serving JSONSchemas.

class invenio_jsonschemas.ext.**InvenioJSONSchemas**(*app=None*, ***kwargs*)
Invenio-JSONSchemas extension.

Register blueprint serving registered schemas and can be used as an api to register those schemas.

Note: JSON schemas are served as static files. Thus their "id" and "$ref" fields might not match the Flask application's host and port.

Extension initialization.

Parameters **app** – The Flask application. (Default: None)

init_app(*app*, *entry_point_group=None*, *register_blueprint=True*, *register_config_blueprint=None*)
Flask application initialization.

Parameters

- **app** – The Flask application.
- **entry_point_group** – The group entry point to load extensions. (Default: invenio_jsonschemas.schemas)
- **register_blueprint** – Register the blueprints.
- **register_config_blueprint** – Register blueprint for the specific app from a config variable.

init_config(*app*)
Initialize configuration.

class invenio_jsonschemas.ext.**InvenioJSONSchemasAPI**(*app=None*, ***kwargs*)
Invenio-JSONSchemas extension for API.

Extension initialization.

Parameters **app** – The Flask application. (Default: None)

init_app(*app*)
Flask application initialization.

Parameters **app** – The Flask application.

class invenio_jsonschemas.ext.**InvenioJSONSchemasState**(*app*)
InvenioJSONSchemas state and api.

Constructor.

Parameters **app** – application registering this state

get_schema(**args*, ***kwds*)
Retrieve a schema.

Parameters

- **path** – schema's relative path.
- **with_refs** – replace $refs in the schema.
- **resolved** – resolve schema using the resolver *invenio_jsonschemas.config.JSONSCHEMAS_RESOLVER_CLS*

Raises ***invenio_jsonschemas.errors.JSONSchemaNotFound*** – If no schema was found in the specified path.

Returns The schema in a dictionary form.

get_schema_dir(*path*)
Retrieve the directory containing the given schema.

Parameters **path** – Schema path, relative to the directory where it was registered.

Raises ***invenio_jsonschemas.errors.JSONSchemaNotFound*** – If no schema was found in the specified path.

Returns The schema directory.

get_schema_path(*path*)
Compute the schema's absolute path from a schema relative path.

Parameters **path** – relative path of the schema.

Raises ***invenio_jsonschemas.errors.JSONSchemaNotFound*** – If no schema was found in the specified path.

Returns The absolute path.

list_schemas()
List all JSON-schema names.

Returns list of schema names.

Return type list

loader_cls
Loader class used in *JsonRef.replace_refs*.

path_to_url(*path*)
Build URL from a path.

Parameters **path** – relative path of the schema.

Returns The schema complete URL or `None` if not found.

register_schema(*directory*, *path*)
Register a json-schema.

Parameters

- **directory** – root directory path.
- **path** – schema path, relative to the root directory.

register_schemas_dir(*directory*)
Recursively register all json-schemas in a directory.

Parameters **directory** – directory path.

resolver_cls
Loader to resolve the schema.

url_to_path(*url*)
Convert schema URL to path.

Parameters **url** – The schema URL.

Returns The schema path or `None` if the schema can't be resolved.

class invenio_jsonschemas.ext.**InvenioJSONSchemasUI**(*app=None*, ***kwargs*)
Invenio-JSONSchemas extension for UI.

Extension initialization.

Parameters **app** – The Flask application. (Default: `None`)

init_app(*app*)
Flask application initialization.

Parameters **app** – The Flask application.

2.1.1 Errors

Invenio-JSONSchemas errors.

exception invenio_jsonschemas.errors.**JSONSchemaDuplicate**(*schema*, *first_dir*, *second_dir*, **args*, ***kwargs*)
Exception raised when multiple schemas match the same path.

Constructor.

Parameters

- **schema** – duplicate schema path.
- **first_dir** – first directory where the schema was found.
- **second_dir** – second directory where the schema was found.

exception invenio_jsonschemas.errors.**JSONSchemaError**
Base class for errors in Invenio-JSONSchemas module.

exception invenio_jsonschemas.errors.**JSONSchemaNotFound**(*schema*, **args*, ***kwargs*)
Exception raised when a requested JSONSchema is not found.

Constructor.

Parameters **schema** – path of the requested schema which was not found.

2.1.2 JSON resolver

JSON resolver for JSON schemas.

invenio_jsonschemas.jsonresolver.**jsonresolver_loader**(*url_map*)
JSON resolver plugin that loads the schema endpoint.

Injected into Invenio-Records JSON resolver.

2.1.3 Views

Invenio module for building and serving JSONSchemas.

invenio_jsonschemas.views.**create_blueprint**(*state*)
Create blueprint serving JSON schemas.

Parameters **state** – *invenio_jsonschemas.ext.InvenioJSONSchemasState* instance used to retrieve the schemas.

2.1.4 Utils

Invenio JSONSchemas utils.

invenio_jsonschemas.utils.**resolve_schema**(*schema*)
Transform JSON schemas "allOf".

This is the default schema resolver.

This function was created because some javascript JSON Schema libraries don't support "allOf". We recommend to use this function only in this specific case.

This function is transforming the JSON Schema by removing "allOf" keywords. It recursively merges the subschemas as dictionaries. The process is completely custom and works only for simple JSON Schemas which use basic types (object, string, number, ...). Optional structures like "schema dependencies" or "oneOf" keywords are not supported.

Parameters **schema** (*dict*) – the schema to resolve.

Returns the resolved schema

Note: The schema should have the `$ref` already resolved before running this method.

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-jsonschemas/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-JSONSchemas could always use more documentation, whether as part of the official Invenio-JSONSchemas 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-jsonschemas/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-jsonschemas* for local development.

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

```
$ git clone git@github.com:your_name_here/invenio-jsonschemas.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-jsonschemas
$ cd invenio-jsonschemas/
$ 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 and 3.5. Check https://travis-ci.org/inveniosoftware/invenio-jsonschemas/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

3.4 Contributors

- Alexander Ioannidis
- Alizee Pace
- Chiara Bigarella
- Chris Aslanoglou
- Diego Rodriguez
- Harris Tzovanakis

- Jacopo Notarstefano
- Javier Martin Montull
- Jiri Kuncar
- Krzysztof Nowak
- Lars Holm Nielsen
- Leonardo Rossi
- Marco Neumann
- Nicolas Harraudeau
- Paulina Lach
- Robin Schroer
- Sami Hiltunen
- Sebastian Witowski
- Tibor Simko

Python Module Index

i

Index

U