Watson - Serialize

Release 1.0.1

Jan 15, 2018

Contents

Process and route HTTP Request messages.

CHAPTER 1

Build Status

CHAPTER 2

Installation

```
pip install watson-serialize
```

CHAPTER 3

Testing

Watson can be tested with py.test. Simply activate your virtualenv and run `python setup.py test`.

CHAPTER 4

Contributing

If you would like to contribute to Watson, please feel free to issue a pull request via Github with the associated tests for your code. Your name will be added to the AUTHORS file under contributors.

CHAPTER 5

Table of Contents

5.1 Usage

Serializing objects is an important way of ensuring that you can send data between your application easily and effectively. watson-serialize enables you to return a list of objects or a single object from a controller and have it automatically converted in a dictionary/list that is able to be processed by JSONEncoder.

watson-serialize has been inspired by Stormpath's excellent REST API design.

Tip: When designing your API, check out https://stormpath.com/blog/fundamentals-rest-api-design.

5.1.1 Defining your models

Models contain a class named Meta, which will store information about attributes that are to be serialized, strategies for handing complex types, and whether or not null values will be exposed.

Let's start by creating the definition of the model:

Note: This document assumes that you're also using watson-db for creating your models although any ORM that you use will work, so long as the models are defined declaratively.

```
from wason.db import models
from sqlalchemy import Column, Integer, String

class Model(models.Model):

    class Meta(object):
        attributes = (
            'id',
            'name')
```

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

The Meta class within your model can be defined in this way:

```
class Meta(object):
    attributes = []
    strategies = {}
    expand = True
    route = None
```

Defining attributes to be serialized

Only values within the attributes attribute of the Meta class will be allowed to be serialized. You can set these in the following way:

```
attributes = ('attribute1', 'attribute2')
```

Note: The first attribute in the list of attributes will be treated as the 'identifying' attribute, which is used when generating the metadata for the object.

Complex types

Handling complex types (such as date objects) within a model is quite straight forward. Within the Meta class, you can set properties on the *strategies* attribute. It follows this format:

```
def method_to_handle_processing(value):
    return value.attribute  # process the value here

strategies = {
    'attribute_name': method_to_handle_processing
}
```

Usually a simple lambda function is all that is needed, which is particularly useful for types like enums.

```
strategies = {
    'enum_attribute': lambda x: x.value
}
```

Automatically expanding nested models

By default, any nested models will expand all of the attributes within them. If you'd prefer to keep these to a minimum and force your users to explore your API to retrieve full data sets, you can set *expand* to *False* on the Meta object.

```
class Meta(object):
    expand = False
```

This will prevent the entire model from being returned, and will only return the identifying attribute.

Object metadata

Every serialized object will also include metadata to make it discoverable via your REST API.

```
{
    # existing object attributes

    'meta': {
        'href': '/path/to/resource'
    }
}
```

5.1.2 Serializing paginated results and lists

Often you won't want to return every single object via a REST API, you'll want to paginate them in some way. Returning a *watson.db.utils.Pagination* object to be serialized will automatically add additional metadata to the results.

```
{
    'items': [
        {
            # existing object attributes

            'meta': {
                'href': '/path/to/resource'
            }
        },
        # more items
    ],
    'meta': {
        'limit': N,
        'page': N,
        'total': N,
        'href': '/path/to/resources?page=N'
    }
}
```

5.1.3 Utilizing watson-serialize in your controllers

Of course, you could code all of what you've read above into your Watson controllers by using the lower level API, but that's why we have a simple *serialize* decorator available to get you up and running quickly.

Note: The following code assumes you're using watson-db and the associated repositories and Pagination. If you're not, just look at the decorators being used.

```
from watson.db import utils
from watson.framework import controllers
from watson.framework.views.decorators import view
from watson.serialize.decorators import serialize

class Controller(controllers.Rest):

    repository = SomeInjectedRepository()
```

```
    @view(format='json')
    @serialize
    def GET(self, id=None):
        if not id:
            return utils.Pagination(self.repository.query)
        return self.repository.get(id=id)
```

When the controllers *GET* action is called, the response will be serialized, and then JSON encoded for the end user.

Raising exceptions for your users

When you raise an exception during API access, you'll want to make sure that it is meaningful and useful for your end users.

```
from watson.serialize import errors

class MyRestError(errors.Base):
    pass

# somewhere within your controller
raise MyRestError(
    code=00,
    status_code=400,
    message='Something broke',
    developer_message='Invalid index supplied')
```

When the response is sent to the end user, they will receive a 400 error, with the following result:

```
{
    code: 40000,
    message: 'Something broke',
    developer_message: 'Invalid index supplied'
}
```

As you can see, code and status_code get combined to give you a unique code specifically related to that error.

5.1.4 How an end user will interact with your API

Assuming you've got your REST API hosted at *http://127.0.0.1*, and the models are exposed at */v1/models* the user would make the request in the following way:

```
curl "http://127.0.0.1/v1/models?include=name" \
    -H 'Content-Type: application/json; charset=utf-8'
```

The above request would return all of the models, and the *name* attribute for them.

The user could also exclude values instead of including specific ones:

```
curl "http://127.0.0.1/v1/models?exclude=name" \
    -H 'Content-Type: application/json; charset=utf-8'
```

This would return every attribute on the models except for the name attribute.

All of the query strings are just comma separated values, for example ‘?include=attribute,attribute2.

If you have chosen to prevent nested models from being expanded by default, the user will need to request the relevant attributes:

```
curl "http://127.0.0.1/v1/models?include=nested(*)" \
    -H 'Content-Type: application/json; charset=utf-8'
```

This would return every attribute on all models within the *nested* attribute.

```
curl "http://127.0.0.1/v1/models?include=nested(name)" \
    -H 'Content-Type: application/json; charset=utf-8'
```

This would return the *name* attribute on all models within the *nested* attribute.

It's up to you to decide how you'll structure the API itself, as well as the authentication methods for it. There's some great authentication providers available in *watson-auth* which can be seemlessly integrated into your API.

5.2 Reference Library

5.2.1 watson.serialize.decorators

watson.serialize.decorators.**serialize**(*func=None*, *router=None*)

Serialize an iterable object into a format suitable for encoding.

A user can add additional query string parameters to the URL in order to include/exclude/expand certain fields that are contained in the models being exposed.

Parameters **router** (*watson.routing.routers.Base*) – The router to be used, defaults to the router retrieved from the container

Returns A list/dictionary of values suitable for encoding

Usage:

5.2.2 watson.serialize.errors

exception watson.serialize.errors.**Base**(*code*, *message='Unknown Error'*, *status_code=406*, *developer_message=None*)

An extendible class for responding to exceptions caused within your application.

Usage:

__init__(*code*, *message='Unknown Error'*, *status_code=406*, *developer_message=None*)

Initialize the error.

Parameters

- **code** (*int/string*) – A unique exception code for the error, gets combined with the status_code
- **status_code** (*int*) – The HTTP status code associated with the response (see https://httpstatuses.com/)
- **message** (*string*) – A human friendly exception message

- **developer_message** (*string*) – A more complex exception message aimed at deveopers

5.2.3 watson.serialize.serializers

class watson.serialize.serializers.**Instance**(*router*)

Serialize an iterable object into a format suitable for json encoding.

include_name

boolean – Whether or not to include null values in the output

strategies

type

mixed – The class name of the object being serialized

Returns A list of objects that are suitable to be json encoded

Usage:

identifier

Unique identifier for the object being serialized.

Defaults to the first

Returns The first value from the attributes Meta class.

Return type string

Python Module Index

W

Index

Symbols

B

I

S

T

W