cosmicray Documentation

Release 0.0.5

Samir Omerovic

Mar 16, 2018

Contents

CHAPTER 1

Cosmicray

1.1 Develop a client for any http API and document its quirks and features

Cosmicray is a simple and high level http client API development framework. It provides the basic building blocks for defining enpoints, handling a request response and automatically converting the result into to Models.

Motivation:

- Ease of use
- Configureability and customization on every level
- Namespace different backends (one client to rule them all)
- Separate route definitions / response handling from models or "business logic"
- Ability to validate requests before making them
- Authenticate each request as needed
- Ability to associate routes to models

Warning: Cosmicray is under development

1.1.1 Install

```
$ pip install cosmicray
```

1.1.2 Quick start

Create App

```
>>> from cosmicray import Cosmicray
>>> api = Cosmicray('myapp', domain='http://mydomain.com')
```

Define routes and response handlers

Using the app we created, we can now add routes for it and define a response handler for each one. The response handler is simply a regular function that accepts a single argument of type *requests.Response* and returns the processed result.

```
>>> @api.route('/v1/dogs/{id}', ['GET', 'POST', 'PUT', 'DELETE'])
>>> def dogs(response):
...     return response.json()
```

- The decorator *api.route* creates an instance of *cosmicray.Route* named *dogs* and stores the given function internally as the response handler.
- Instances of *cosmicray.Route* are callable and accept parameters:
  - *model_cls*: Optional: Class that implements *_make(cls, response)* classmethod.
  - ***kwargs*: Keyword arguments.
    - *urlargs*: Mapping for url formatting arguments
    - *headers*: Mapping for headers
    - *params*: Mapping for query parameters
    - *data*, *json*, *files*: Request body
    - *authenticator*: Authenticator callback
    - *&rest*: Requests keyword arguments
- When and instance of *cosmicray.Route* is called, it returns a *Request* object and with this you can:
  - Use functions defined for each http method (ex: *get()*, *post()*, *put()*, *delete()*)
  - Override any parameters passed in (ex: *params*, *headers*, etc.) with setters
  - Automatically validates given parameters against the defined parameters on the *Route*
  - Authenticates the request, if the app was configured with an authenticator
  - After the response is handled by the response handler, the result is automatically mapped to the model class, if one was provided

How to make requests

```
>>> dogs().get()
>>> dogs(urlargs={id: 12345}).get()
>>> dogs(json={'name': 'Manu'}).post()
>>> dogs(urlargs={'id': 12345}, json={'age': 4}).put()
>>> dogs(urlargs={'id': 12345}).delete()
```

To specify request parameters

```
>>> dogs(params={'breed': 'husky'},
...      headers={'Content-Type': 'application/json'}).get()
```

Authenticating requests

Often you'll need to authenticate requests to access private resource and Cosmicray has a built-in mechanism to perform this step.

```
>>> def authenticator(request):
...     if not request.is_request_for(login):
...         auth = login(json={'username': 'me', 'password': 'mysecret'}).post()
...         return request.set_headers({'X-AUTH-TOKEN': auth['token']})
...     return request
...
>>> @api.route('/oauth', ['POST'])
... def login(response):
...     """Get an auth token for the given credentials"""
...     return response.json()
...
>>> @api.route('/private/resource', ['GET'])
... def private_resource(response):
...     """Must be authenticated to access this"""
...     return response.json()
...
>>> api.configure(authenticator=authenticator)
>>> # Now the private resourse will be automatically updated to include auth headers
>>> private_resource.get()
```

1.1.3 Models

Basics

- Cosmicray ships with a built-in Model class
- This base class is bound to a specific route handler and defines all the fields that would get mapped to a response or be part as the payload for *post* and *put* requests
- It automatically uses its defined fields as url parameters and as request body
- Provides functions to make http calls (ex: *get*, *create*, *update*, *delete*)
- You can override default behavior, such as create/update paylods

```
>>> from cosmicray.model import Model
>>> class Dog(Model):
...     __route__ = dogs
...     __slots__ = [
...         'id',
...         'name',
...         'breed',
...         'age'
...     ]
>>> manu = Dog(name='Manu', age=4).create()
>>> manu.breed = 'Husky'
```

```
>>> manu.update()
>>> manu.delete()
>>> manu = Dog(id=12345).get()
>>> alldogs = Dog().get()
```

Relationships with other models/routes

```
>>> from cosmicray.model import relationhip, Model, ModelParam
>>> class Cat(cosmicray.model.Model):
...     __route__ = cats
...     __slots__ = [
...         'id',
...         'name',
...         'age'
...     ]
...     friends = relationhip('Friend', urlargs={'id': ModelParam('id')})
```

If you don't want to use *cosmicray.Model* as your base, you can define your own OR even use just use *collections.namedtuple* as the model.

```
>>> class MyModel(object):
...     @classmethod
...     def _make(cls, response):
...         obj = cls()
...         ... do stuff with the response
...         return obj
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search