Resto Documentation

Release planning

Rafael Pivato

June 07, 2014

Contents

When you start creating a new REST-like service all that is required are a couple of functions. They process input data and return results usually in JSON type. All data your client need is there. It fit just right and that stance goes on until you realize you have a couple more functions than you thought initially.

After some hundreds extra lines than you would really like. And the worst thing is, maintaining its documentation is as hard as controlling and extending your API in a structured manner. You can avoid that making some brain storm before starting your first functions, but in the end, Python's flexibility will let you create unstructured code easily, removing the gap between your interface and your application logic.

That's where Resto comes into place. Its purpose is to help keeping application logic strong, allowing it to grow structured while making room for a REST interface that will be just that. In the end an application built with Resto will have a concise and clear data model. As a bonus, if all guidelines are applied the application will also be really framework agnostic.

This is specially awesome for Start-ups and new projects, where available time for planning is really too short, and you can not predict what will be your next feature. If you do not have room for rethinking things over and over again, start fast and do it once with Resto.

CHAPTER 1

How Resto will solve that

The excess of flexibility allied with the lack of resources for designing or predicting whole interface, even more growth possibilities, is what will be reduced in order increase maintainability. This is somewhat based on the iron triangle idea.

Another approach which Resto is based on is to not reinvent the wheel while using Python as the language. It is noticeable that existing frameworks and mapping tools require considerable amount of meta information to set things up. Resto still wants to provide such power in the future, but that's not how you'll start.

Its inner will is to allow declaring an interface using Python OOP mechanism with minimal interface requirements. This means that defining a class, as you would in Python, is all you need to have a REST interface defined.

CHAPTER 2

Roadmap

The project will target three main releases. First one will be 0.1 and should allow you to transfer a representational state of your service objects and operations using a simple structuring. That will allow creating components like the following:

```
http://example.com/teacher/612
http://example.com/course/214
http://example.com/student/345
http://example.com/teacher?course=214
http://example.com/student?course=214
http://example.com/course?teacher=612
```

You can see filters above that will allow you to constrain data being transferred. It allows getting teachers for a course, students for the same course and all courses for a given teacher.

In a second moment, when version 0.2 goes out, Resto should have the ability to provide compound components that depend on each other, allowing you to get, for instance, all courses for a given teacher:

```
/teacher/612/course
/course/345/student
```

If you think above is the same than using filters then this detail will explain things better. The different lies on the fact that for this second one, Resto will truly understand that there is a relation between a teacher and a course, whether at first version all we had was a filter based on an attribute.

2.1 Composite Endpoints

The last of very first three main versions will be major release 2.1 and should allow us to provide composite resources. This will be helpful for adapting your interface to specific clients. If you never read about composite endpoints it will be easy understanding its usage with front end UI clients like mobile applications. Such kind of clients need information from different resources in order to build one single screen. Instead of forcing them to make multiple round-trips for each resource, a composite may be defined in order deliver all required information in one step.

CHAPTER 3

Discuss

This is not a new idea yet certainly requires some thinking about it before getting a lot of code done. At least requires some exercises as code gets written and tests show one or another concept is wrong. There is no discussion group on the subject because there is really no group around it at this moment. If you feel likely to contribute then email the author and maybe some discussion can start.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*