Madame Documentation

Release 0.1.2

Asdine El Hrychy

January 08, 2013

CONTENTS

Release v0.1.2. (*Installation*)

Welcome to Madame's documentation. Madame is a RESTful API for MongoDB built on Flask. It is highly customizable and easy to use and to configure.

```
from madame import Madame

app = Madame()
app.run()
```

Madame depends on two main libraries: the Flask microframework and the Validictory data validator. These libraries are not documented here. If you want to dive into their documentation, check out the following links:

- Flask documentation
- Validictory documentation

CHAPTER

ONE

USER'S GUIDE

1.1 Introduction

Madame is a RESTful API for MongoDB built on Flask. It tends to be highly customizable and easy to use and configure.

1.1.1 Features

- **REST-compliant**
- **HATEOAS**
- **Dynamic collections**
- **JSON Schemas separated from code**
- **JSON Validation**
- **Cache**
- **Versioning**
- **Read-only by default**

1.1.2 License

CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.1.3 Authors

Develoment Lead

- Asdine El Hrychy <asdine.elhrychy@gmail.com>

1.2 Installation

This part of the documentation covers the installation of Madame.

1.2.1 Pip and easy_install

You can install Madame using `pip`

```
$ pip install madame
```

or using easy_install:

```
$ easy_install madame
```

1.2.2 Source Code

You can also get the code from Github

```
$ git clone git://github.com/asdine/madame.git
```

and install it:

```
$ cd madame
$ python setup.py install
```

1.3 Usage

Using Madame is very simple.

1.3.1 Prerequisites

Madame needs MongoDB to work properly. To do so, install it from MongoDB's website.

1.3.2 Basic usage

First, import the module:

```
>>> from madame import Madame
```

Then instantiate the Madame class:

```
>>> app = Madame()
```

Finally, run it:

```
>>> app.run()
```

Without configuration, your application is read-only and with no schemas. You would probably want to enable some collections. Go to the *Tutorial* section to see how easy it is to configure Madame applications.

1.4 Tutorial

This part of the documentation will help you learn how to use Madame by example.

In this tutorial, we will create a simple address book called MadameBook.

1.4.1 The goal

When you setup a new Madame server, you need to think about what methods you want to be allowed for users and what methods you don't.

For our address book, we want to :

- Get the collection list
- Get the contact list
- Add a new contact
- Get the contact information

But we don't want to be able to:

- Delete the whole contact list
- Delete a contact
- Modify a contact
- Add new collections
- Delete collections

1.4.2 Step 1 - The folders

Create a folder called `MadameBook` on your computer.

Download and install MongoDB anywhere.

1.4.3 Step 2 - The configuration

Create a file called `config.py` at the root of your folder, this file will be automatically loaded by Madame.

If you have installed MongoDB on localhost and haven't changed the port, you can skip these lines.

Otherwise, add theses lines in your config file:

```
MONGO_HOST = <IP>
MONGO_PORT = <PORT>
```

Change `<IP>` and `<PORT>` by the actual ip address and port of the server where MongoDB is installed.

1.4.4 Step 3 - The Database Schema

Now, we need to describe our data.

Madame uses JSON format to do so, and validictory to validate the data.

Create a file called `schemas.json` at the root of your folder.:

```
{
    "contacts": {
          "title"      : "Contacts",
          "description" : "My contact list",
          "schema"     : {
              "type" : "object",
              "properties" : {
                  "firstname" : {
                      "type"  : "string",
                      "minLength" : 1,
                      "maxLength" : 10
                  },
                  "lastname"  : {
                      "type"  : "string",
                      "minLength" : 1,
                      "maxLength" : 15,
                      "required"  : true
                  },
                  "phone-number" : {
                      "type"  : "string",
                      "format": "phone"
                  }
              }
          }
    }
}
```

The first line describes the name of your domain.

It will be used in the url:

```
GET http://localhost:5000/contacts/
```

The value of `"schemas"` follows validictory format. You can add as much schemas as you want inside this file.

Now, we need to tell Madame about our schema file.

Add this line in your configuration file:

```
SCHEMA_FILE = "schemas.json"
```

1.4.5 Step 4 - The methods allowed

Madame is read-only by default. It means that only the `GET` methods are allowed for the various urls.

Here's what we want to be allowed:

```
GET       /
GET       /contacts/
POST      /contacts/
GET       /contacts/id/
```

Here's what is allowed by default:

```
GET       /
GET       /contacts/
GET       /contacts/id/
```

To add the `POST` method on our collection, we need to tell it to Madame. Add the following line in your configuration file:

```
COLLECTION_POST = True
```

1.4.6 Step 5 - The server

Now that our configuration is set, we need to launch the server.

Create a file called `server.py`.

```
from madame import Madame

app = Madame()

app.run()
```

Now, launch it:

```
$ python server.py
```

1.4.7 Examples

To use your beautiful server, you can use Requests, `curl`, `Ajax`, or anything you want.

Here is an example with `curl` :

```
$ curl http://localhost:5000/
{
  "title": "Content",
  "description": "List of collections",
  "links": [
      {
          "href": "http://localhost:5000/",
          "description": "You are here.",
          "rel": "self",
          "title": "root"
      },
      {
          "href": "http://localhost:5000/contacts/",
          "description": "My contact list",
          "rel": "child",
          "title": "Contacts"
      }
  ]
}
```

Let's add a new contact :

```
$ curl -d '{ "lastname" : "baggins", "firstname" : "bilbo", "phone-number" : "555-666" }' -H "Content
{
  "title": "Document created",
  "links": [
      {
          "href": "http://localhost:5000/contacts/",
          "description": "You are here.",
          "rel": "self",
          "title": "Contacts"
      },
      {
          "href": "http://localhost:5000/contacts/50e737860ef3c42120601fae",
          "rel": "item"
      }
  ]
}
```

Then let's see the contact information :

```
$ curl http://localhost:5000/contacts/50e737860ef3c42120601fae
{
  "_id": "50e737860ef3c42120601fae",
  "firstname": "bilbo",
  "lastname": "baggins",
  "phone-number": "555-666",
  "created": "Fri Jan  4 20:11:50 2013",
  "updated": "Fri Jan  4 20:11:50 2013",
  "etag": "4a98c403225da6cab1b8f5557492a3c370666fe0"
}
```

1.5 Configuration Handling

Madame doesn't necessarily need a configuration file, but it can be very helpful to create one if you want to change your server's behaviour.

1.5.1 The configuration file

As Flask, Madame can handle configuration in two ways.

You can choose between one of those two, or use both for one to be more global and the other more specific.

From an environment variable

Create your file wherever you want on your server. Then, you need to create an environment variable called `MADAME_SETTINGS` and set its value with the path of your file.

```
$ touch /path/to/your/configfile.py
$ export MADAME_SETTINGS=/path/to/your/configfile.py
```

From the root of your application

Create the file at the root of your application. The file name has to be `config.py`.

```
$ cd /path/to/your/application
$ touch config.py
```

1.5.2 Builtin configuration values

You can use all of the Flask configuration values.

Here are some useful values:

Option	Description	Default value
DEBUG	enable/disable debug mode.	`True`
MONGO_HOST	The host name or IP address of your MongoDB server.	`"localhost"`
MONGO_PORT	The port number of your MongoDB server.	`27017`
MONGO_USERNAME	The user name for authentication.	`None`
MONGO_PASSWORD	The password for authentication.	`None`

Madame has also its own values:

Option	Description	Default value
SCHEMA_FILE	describes the name of the schema file.	`None`
ROOT_GET	enable/disable GET method on root.	`True`
ROOT_POST	enable/disable POST method on root.	`False`
ROOT_DELETE	enable/disable DELETE method on root.	`False`
COLLECTION_GET	enable/disable GET method on collections.	`True`
COLLECTION_POST	enable/disable POST method on collections.	`False`
COLLECTION_PUT	enable/disable PUT method on collections.	`False`
COLLECTION_PATCH	enable/disable PATCH method on collections.	`False`
COLLECTION_DELETE	enable/disable DELETE method on collections.	`False`
ITEM_GET	enable/disable GET method on items.	`True`
ITEM_PUT	enable/disable PUT method on items.	`False`
ITEM_PATCH	enable/disable PATCH method on items.	`False`
ITEM_DELETE	enable/disable DELETE method on items.	`False`
ROOT_TITLE	sets the root title that is displayed in responses.	`"Content"`
ROOT_DESCRIPTION	sets the root title that is displayed in responses.	`"List of collections"`

1.6 Schemas

Madame doesn't use any python ORM to describe data, it uses JSON schemas to describe each collections. The idea was to separate schemas from code to enable modifications during runtime. Schemas are considered as part of data, except the fact that they are not saved in the database.

Madame uses Validictory to validate data with the schemas provided.

There are two ways to add schemas:

- Decribe them in a file
- Add them with a *POST* method on the root url or your API during runtime.

1.6.1 The schema file

Creating a schema file allows you to set schemas available when your start the application.

These schemas can be disabled with a DELETE method during runtime if you set the configuration value ROOT_DELETE to True in the configuration file. See *Configuration Handling* section for more details.

To tell Madame about your schema file, add this line in your configuration file :

```
SCHEMA_FILE = <PATH>
```

Replace <PATH> with the path of your schema file.

The schema wrapper

Schemas are encapsulated in a wrapper that contains additional informations about your collections.

Here is the minimal content that needs to be in your schema file :

```
{
  <COLLECTION_URL> : {
      "title" : <COLLECTION_TITLE>,
      "description" : <COLLECTION_DESCRIPTION>,
      "schema" : {} # the validictory schema
  },

  <COLLECTION_URL> : ...
}
```

<COLLECTION_URL>	sets the url of your collection *example* : for "members" -> http://localhost:5000/members/
<COLLECTION_TITLE>	used in the response to describe the given collection
<COLLECTION_DESCRIPTION>	used in the response to describe the given collection

To get more informations about how to write schema, see the Validictory documentation, it is very simple and doesn't need to be documented here.

1.6.2 Adding schemas with POST

If the configuration value ROOT_POST is enabled, you can add schemas with a POST method. Here is an example with Requests :

```
>>> import request, json

>>> schema = { ... }
>>> headers = {'content-type': 'application/json'}
>>> r = requests.post('http://localhost:5000/',
... data=json.dumps(schema), headers=headers)
>>> print r.status_code
201
```

1.6.3 Schema example

In the schema file :

```
{
  # Pet collection
  # url : /pets/
  "pets" : {
```

```
      "title" : "Pets",
      "description" : "My pets",
      "schema" : {
          "type" : "object",
          "properties" : {
              "name" : {
                  "type"  : "string",
                  "minLength" : 1,
                  "maxLength" : 15,
                  "required"  : true
              },
              "age" : {
                  "type"  : "number"
              }
          }
      }
  },

  # Book collection
  # url : /books/
  "books" : {
      "title" : "Books"
      "description" : "My book list",
      "schema" : {
          "type" : "object",
          "properties" : {
              "title"     : {"type" : "string", "required" : true},
              "author"    : {"type" : "string", "required" : true}
          }
      }
  }
}
```

With the `POST` method :

```
>>> import request, json

>>> schema = {
... "type" : "object",
... "properties" : {
...     "title"     : {"type" : "string", "required" : true},
...     "author"    : {"type" : "string", "required" : true}
... }}
>>> headers = {'content-type': 'application/json'}
>>> r = requests.post('http://localhost:5000/',
... data=json.dumps(schema), headers=headers)
>>> print r.status_code
201
```