mustachebox Documentation

Release 0.1

Yohann Gabory

February 19, 2013

CONTENTS

Contents:

CHAPTER

ONE

INSTALLATION

1.1 Requirements

mustache box has no particular dependencies. As a django-app it need a working version of Django >= 1.3.

But in order to use the example backend, particulary the page listing all the graphs you can use, you have to install the docutils.

so just clone this repository :

1.2 Cloning the repository

```
hg clone https://boblefrag@bitbucket.org/boblefrag/mustachebox
```

1.3 Install Steps

1.3.1 Adding the app to the INSTALLED_APPS

```
# settings.py

 INSTALLED_APPS = (
...
'mustachebox',
)
```

1.3.2 Defining a GRAPH_BACKEND

As mustachebox rely on a backend, you must define your own. It's realy easy however. For a starting point, you can use the monitor_backend. This backend exist for testing and example purpose.

```
GRAPH_BACKEND="mustachebox.backends.example_backend"
```

1.3.3 Add the mustachebox urls to your project

include mustachbox to your urls :

```
url(r'^graphs/', include('mustachebox.urls')),
```

1.4 Testing

To start playing with MustacheBox, you can visit :

(http://localhost:8000/grapher/all/)

That list all the graphs the example backend can give you.

You can learn how to use mustachebox and create your own graph with following *Tutorial*

CHAPTER

TWO

TUTORIAL

In this tutorial I will show you how to create your own backend and how to render graph using this backend.

2.1 Create the Backend

The backend file can be located anywhere in your project. For example, the example backend of mustachebox is located in

`mustachebox/backends/example_backend`

But this is only an example and you can create your own backend in your own apps if you prefer.

For this exemple, you can create a file named "tutorial_backend" in the backends directory of mustachbox applications.

`mustachebox/backends/tutorial_backend`

2.1.1 Backend Format

The format of the backend you create is realy simple. mustachebox will look for a class named Backend. This class must inherit "mustachebox.backends.BaseBackend":

```
from mustachebox.backends import BaseBackend

class Backend(BaseBackend):
```

2.2 Define your first method

You can write any method you want, doing any work you need but it must return serializable data. For exemple, you cannot return python objects or datetime objects.

Your method will be given a kwarg parameter. You can use this parameter to change the data this method return.

for this tutorial, we will use pre-generated data to generate a time serie. This time serie will be render as is by the method because mustachebox will convert it to json befor rendering the template.

```
def test_method(self, **kwargs):
    """
    render a simple time serie suitable for javascript graphs :
        {
            2004: ['2004', 7160, 546],
```

```
            2005: ['2005', 5654, 5435],
            2006: ['2006', 7656, 6545],
            2007: ['2007', 5435, 6545],
            'label': ['year', 'sales', 'expenses']
        }
    """
    response = {
            2004: ['2004', 7160, 546],
            2005: ['2005', 5654, 5435],
            2006: ['2006', 7656, 6545],
            2007: ['2007', 5435, 6545],
            'label': ['year', 'sales', 'expenses']
        }
    return response
```

You can now test the method in the shell

```
python manage.py shell
>>> from mustachebox.backends.tutorial_backend import Backend
>>> Backend(name="test_method").data
'{2004: ['2004', 7160, 546], 2005: ['2005', 5654, 5435], 2006:
['2006', 7656, 6545], 2007: ['2007', 5435, 6545], 'label': ['year', 'sales', 'expenses']}'
```

2.3 Rendering

In order to render your graph, you must define a template. This can be done in your method. For this tutorial we will use the pre-existing area template. This template is suitable for rendering area graphs. So add this at the begining of your method

```
self.template = "area"
```

2.3.1 Testing

You can now easily test your new method. Just change the backend in your settings :

```
GRAPH_BACKEND="mustachebox.backends.tutorial_backend"
```

and hit your graph url:

(http://localhost:8000/graphs/test_method/)

2.4 Add the templatetag

Your graph can already be rendered as a templatetag. Anywhere in your application you can write :

```
{% load graph %}
{% graph test_method %}
```

This is because you use the area template and this template exist in the form of a complete template and in the form of a templatetag template.

2.4.1 Getting the json

Of course you can retreive the json form of your data. For this you can write anywhere in your application an ajax call tu your url :

```
$.ajax({
    type:"GET",
    url: 'http://localhost:8000/graphs/test_method/',
    success: function(response){
        do_something_with_response(response)
    }
})
```

CHAPTER

THREE

GRAPHS API

The graphics you can use with mustachebox are designed to be easy to use and easy to customize.

When you create your backend, you can reuse the javascripts files mustachebox provides.

Each javascript files take a particular format of data to render your graph.

3.1 Google Chart

Those graph uses the google chart tools.

https://developers.google.com/chart/

3.1.1 Time series charts

Area, BarChart and ColumnChart all use the same format of data.

Data formating

The time series chart take a list of integer for each series:

```
[23, 45, 56, 67, 78]
```

you must also provide a list of label

```
['sales', 'expenses', 'counts', 'words counts', 'total']
```

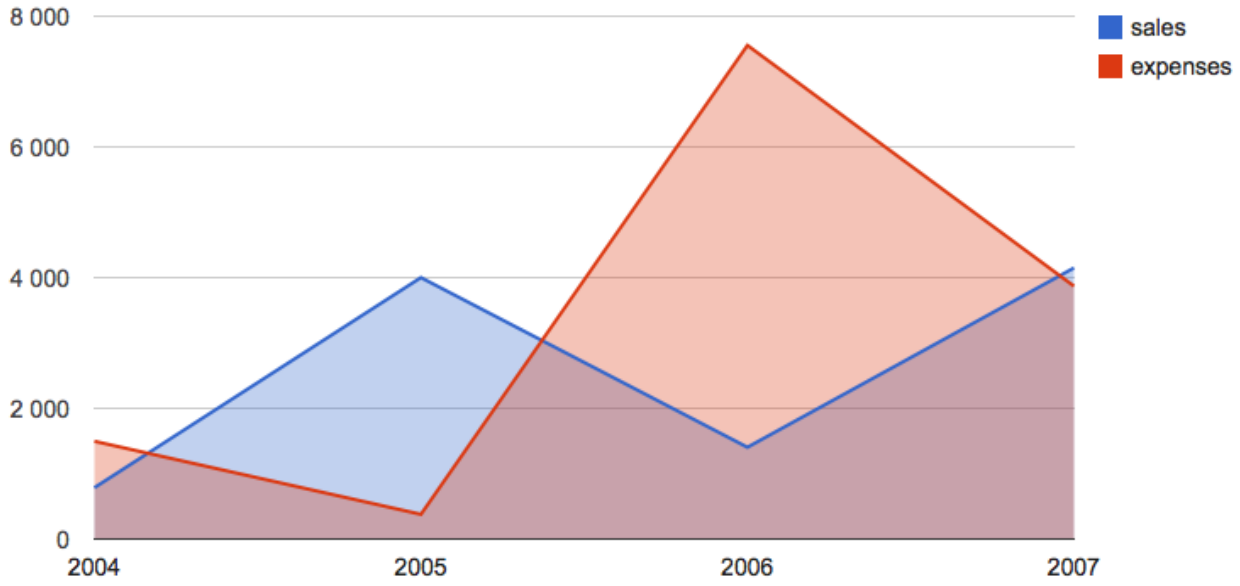
Finaly, you must encapsulate your data in a dictionnary

```
{
 2004: ['2004', 7160, 546],
 2005: ['2005', 5654, 5435],
 2006: ['2006', 7656, 6545],
 2007: ['2007', 5435, 6545],
 'label': ['year', 'sales', 'expenses']
}
```

You can use a time_serie function to parse your data and use this function in your method. For exemple

```
def time_serie(**kwargs):
    """
    Render a time serie
    """
    resp = {2004: ['2004'],
            2005: ['2005'],
            2006: ['2006'],
            2007: ['2007']}

    for i in range(2):
        for k, v in resp.iteritems():
            v.append(int((random.random() * random.random()) * 10000))
        resp['label'] = ['year', 'sales', 'expenses']
    return resp
```

3.1.2 Area



Data formating

See Time series charts

Rendering

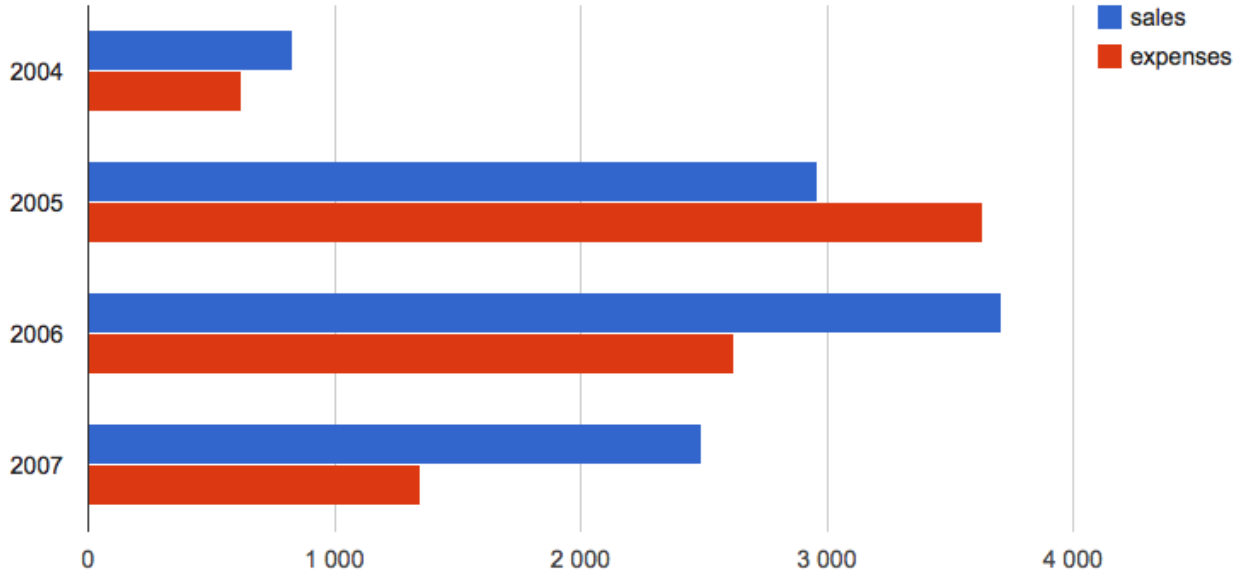
To render your chart you have to call the 'area' template

```
self.template = "area"
```

Complete Example

```
def render_area(self, **kwargs):
    self.template = 'area'
    return time_serie(**kwargs)
```

3.1.3 Bar Chart



Data formating

See Time series charts

Rendering

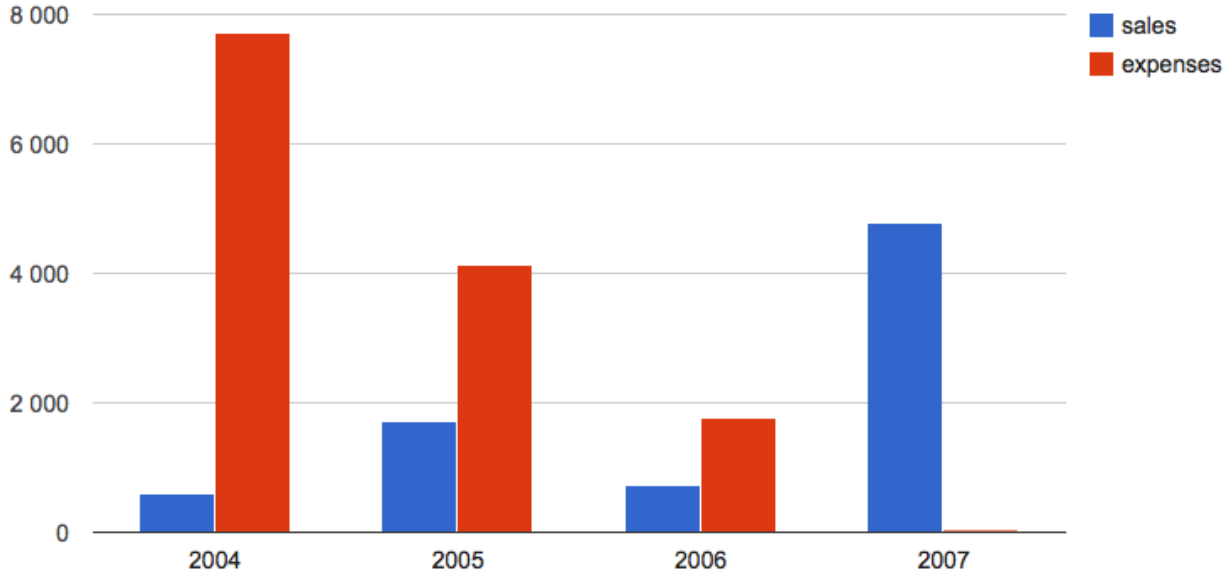
To render your chart you have to call the 'barchart' template

```
self.template = "barchart"
```

Complete Example

```
def render_barchart(self, **kwargs):
    self.template = 'barchart'
    return time_serie(**kwargs)
```

3.1.4 Column Chart



Data formating

See Time series charts

Rendering

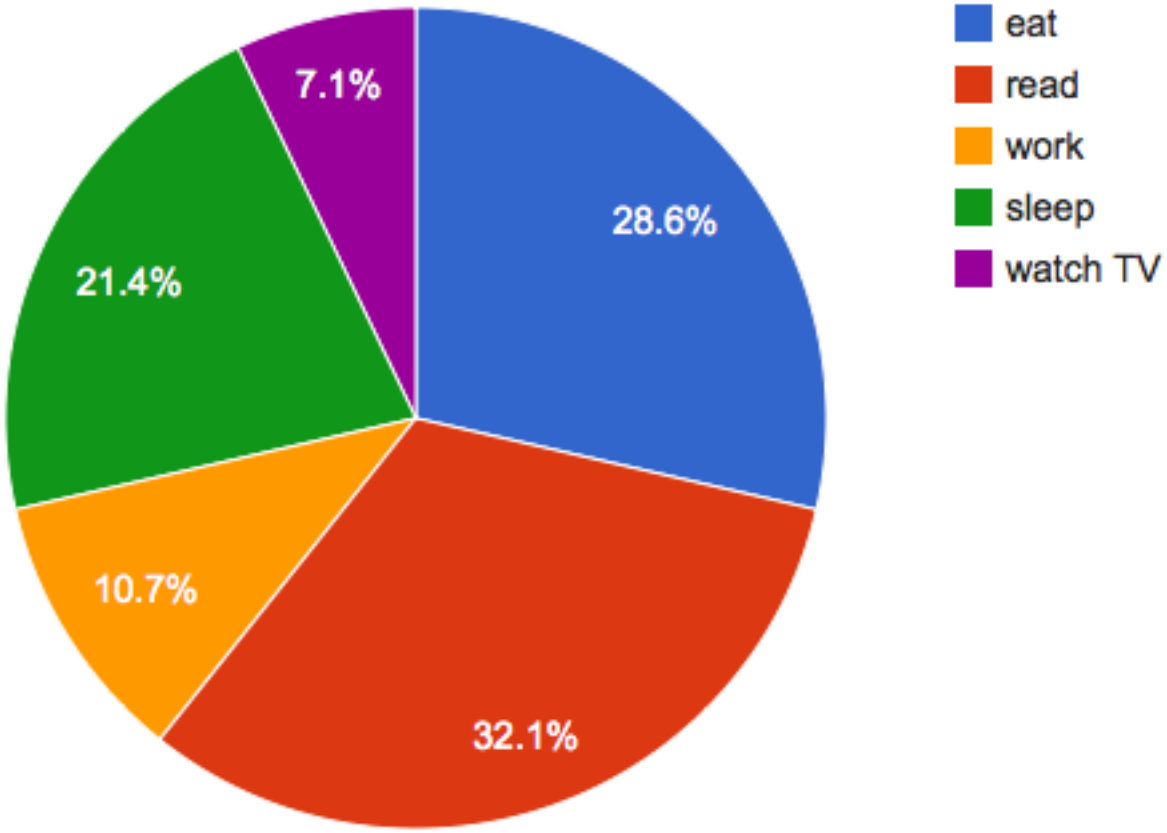
To render your chart you have to call the 'columnchart' template

```
self.template = "columnchart"
```

Complete Example

```
def render_columnchart(self, **kwargs):
    self.template = 'columnchart'
    return time_serie(**kwargs)
```

3.1.5 Pie Chart



Data formating

The data formating is a dictionnary containing :

- a list of label
- a list of dataset containing a string and an integer

The resulting dictionnary must look like this

```
{'label': ['name', 'count'],
 'records': [
                ['a name', 10],
                ...
            ]
}
```

The keys 'label' and 'records' are **mandatory**

Rendering

To render your chart you have to call the 'pie_chart' template

```
self.template = "pie_chart"
```

Complete Example

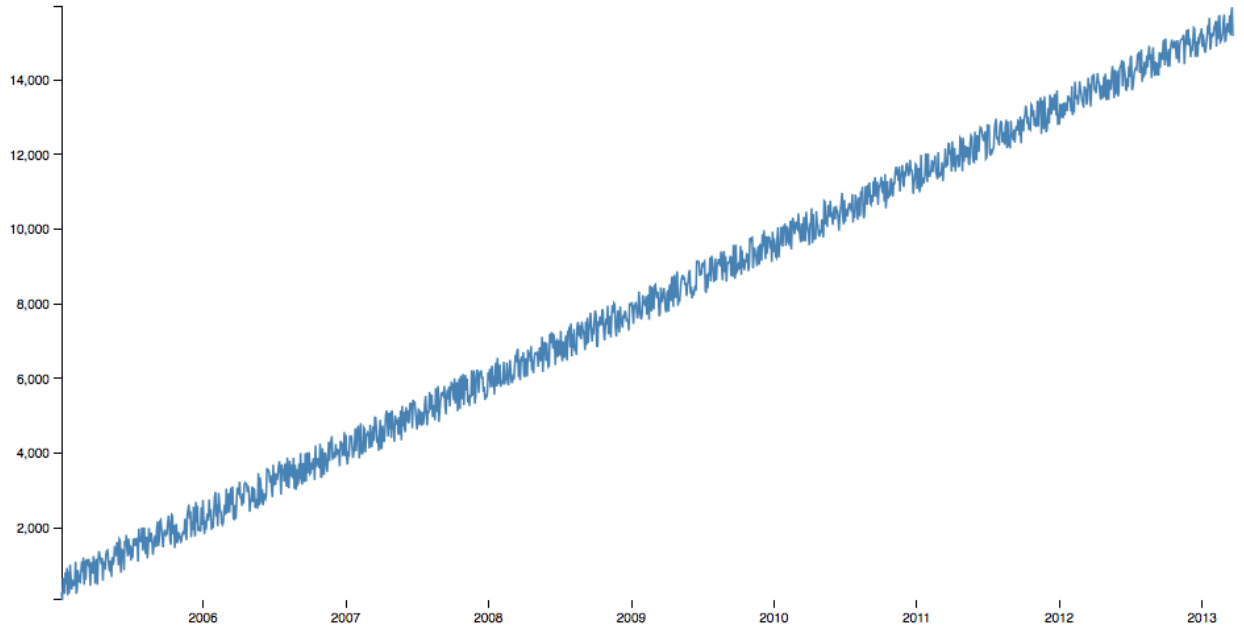
```
def pie_chart(self):
    self.template = "pie_chart"

    label = ['name', 'count']
    activities = []
    words = ['eat', 'read', 'work', 'sleep', 'watch TV']
    for elem in words:
        activities.append([elem, int(random.random() * 10)])

    return {'label': label, 'records': activities}
```

3.2 D3.js

Those graph uses the D3 js Toolkit.

http://d3js.org/

3.2.1 Line Chart



Data formating

The data must be a list of dict.

Each dict constain 2 keys :

- date : a timestamp multiplied by 1000 (python timestamps are in seconds and javascript timestamps are in milliseconds)
- value : an integer or a float

Rendering

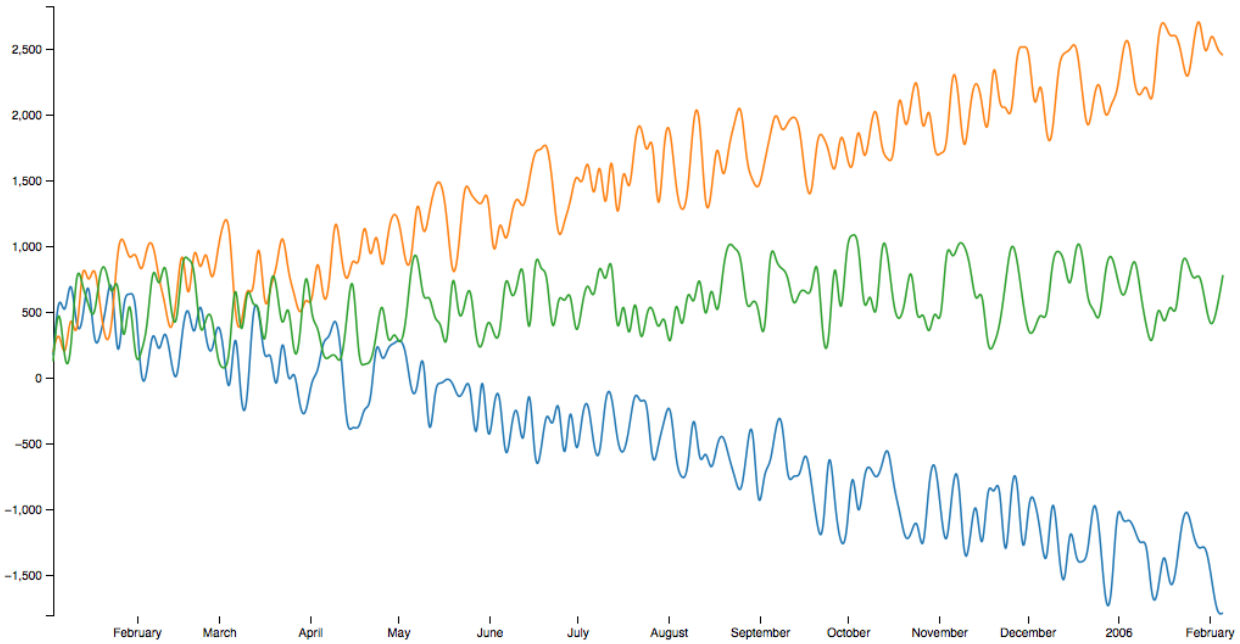
To render your chart you have to call the 'line_chart' template

```
self.template = "line_chart"
```

Complete Example

```
def line_chart(self):

    response = []
    self.template = "line_chart"
    date = datetime.datetime(2005, 1, 1, 0, 0, 0)
    for i in range(1500):
        date += datetime.timedelta(days=2)
        response.append({
            'date': time.mktime(date.timetuple()) * 1000,
            'value': (i * 10) + int(random.random() * 1000)})
    return response
```

3.2.2 Multi Line Chart



Data formating

The data must be a list of dict containing 3 key :

- date : a timestamp multiplied by 1000 (python timestamps are in seconds and javascript timestamps are in milliseconds)
- value : an integer or a float
- serie : the name of the serie the data is belonging to.

Rendering

To render your chart you have to call the 'multiline_chart' template

```
self.template = "multiline_chart"
```

Complete Example

```
def multiserie_linechart(self, **kwargs):
    self.template = "multiline_chart"
    response = []
    for a in range(3):  # we render a 3 series chart
        date = datetime.datetime(2005, 1, 1, 0, 0, 0)
        for i in range(200):
            if a % 3 == 0:
                mult = -10
            elif a % 2 == 0:
                mult = 1
            else:
                mult = 10
            date += datetime.timedelta(days=2)
            response.append({
                'date': time.mktime(date.timetuple()) * 1000,
                'value': (i * mult) + int(random.random() * 1000),
                'serie': a})
    return response
```

CHAPTER

FOUR

BACKENDS

CHAPTER

FIVE

TEMPLATE TAGS

CHAPTER

SIX

JSON OUTPUT

CHAPTER

SEVEN

DESCRIPTION

Mustache Box is a set of utilities to help you presenting graph of various data in a Django Project. Data can come from whatever source you want :

- Distant API
- Couchdb databases
- Your Django models
- ...

7.1 3 parts architecture

To make a graph you will need 3 parts :

- a method : the method lie in a Backend you create (or you can use predefine ones). Each method is responsible for gathering data (the way you want) and parse it to give a formated version of the data to the template.
- A template: the template is a classic HTML file made to hold your graph. You can use some predefine template or use your custom one
- a javascript file : the javascript file take your data and render a graph using the external dependencies you need (jquery, d3js, raphaël, etc...)

7.2 Features

Of course there is serval features you gain with this architecture :

- url auto-discovering : You do not have to design an url for each method. Mustache Box take the pain away for you and design urls for you. Of course, if you don't like this feature, you can design your own urls for each graphs.
- request/ajax response : each url responds either to classic request (rendering a template with the data) and to ajax returning only the formated json suitable for javasccript call. It has never be so easy to create long-pooling call.
- reusability: mustachebow is made in a reusable fashion. You can mix together your methods, javasccript files and html to present different data with the same template and/or javascript, or present the same data in different manners easily
- templatetags: when you create a graph you can use it in a templatetag too. It's really easy to do. You just have to write another template to hold your graph in the templatetag.

This software is released under GPLv3

CHAPTER

EIGHT

INDICES AND TABLES

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