django-dashing Documentation

Release 0.1

Mauricio Reyes

November 23, 2014

Contents

django-dashing is a customisable, modular dashboard application framework for Django to visualize interesting data about your project. Inspired in the exceptionally handsome dashboard framework Dashing

CHAPTER 1

Prerequisites

- Django 1.5.+

CHAPTER 2

Key concepts

- Use premade widgets, or fully create your own with css, html, and javascript.
- Use the API to push data to your dashboards.
- Drag & Drop interface for re-arranging your widgets.

CHAPTER 3

Installation

1. Install latest stable version from PyPi:

```
$ pip install django-dashing
```

2. Add *dashing* to `INSTALLED_APPS` of the your projects.

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

3. Include the polls URLconf in your project urls.py like this:

```
from dashing.utils import router
...

url(r'^dashboard/', include(router.urls)),
```

4. Start the development server and visit http://127.0.0.1:8000/dashboard/ to view the dummy dashboard.

CHAPTER 4

Getting Started

To create a custom dashboard you need create a `dashing-config.js` file in the static directory and optionally a custom `dashing/dashboard.html` template file.

4.1 Config File

You need put the `dashing-config.js` in the static directory (you can change the patch and name if you wrote a template file and start to create widgets for your project.

The dashing config file should start with the creation of a new dashboard `var dashboard = new Dashboard();` and start to place widgets with the following syntax `dashboard.addWidget(<name_of_widget>, <type_of_widget>, <options>);` where *name_of_widget* is the name that describe the objective of the widget (should be unique) *type_of_widget* is a valid widget type (Clock, Graph, List, Number) and options depends of each widget.

4.2 Template File

You can create a *dashboard.html* file to specify which widgets you can use or add your custom widgets, also add stylesheets and extra scripts. You will place inside the template directory in `dashing/dashboard.html`

4.3 Python Widget Classes

Optionally this app provide an useful set of classes to return the expected data for the default widgets, you can create a *widgets.py* file and inherit of these classes or create your own widgets inherit from `dashing.widgets.Widget`.

```
class Widget(JSONResponseMixin, View):
    def get(self, request, *args, **kwargs):
        context = self.get_context()
        return HttpResponse(json.dumps(context), content_type="application/json")

    def render_to_response(self, context, **response_kwargs):
        return self.render_to_json_response(context, **response_kwargs)
```

A custom widget can look like this:

```
class CustomWidget(NumberWidget):
    title = 'My Custom Widget'
    value = 25

    def get_more_info(self):
        more_info = 'Random additional info'
        return more_info
```

To register the url to serve this widget you must use the register method from `dashing.utils.router`, then in *urls.py* file put

```
from dashing.utils import router

router.register(CustomWidget, 'custom_widget')
```

Now we can access to CustomWidget from '/dashboard/widgets/custom_widget' if '/dashboard/' is the root of our dashboard

CHAPTER 5

Dashboards

5.1 Single Dashboard

To initialize a single dashboard you need create a Dashboard object and pass valid options as shown below:

```
var dashboard = new Dashboard(options);
```

Where the *options* are a json object with the following specifications

Options

name ***(optional)*** The name of widget. *(default:* `undefined` *)*

viewportWidth ***(optional)*** Width of viewport where expected that the dashboard was displayed. *(default:* `$(window).width()` *)*

viewportHeight ***(optional)*** Height of viewport where expected that the dashboard was displayed. *(default:* `$(window).height()` *)*

widgetMargins ***(optional)*** Margin between each widget. *(default:* `[5, 5]` *)*

widgetBaseDimensions ***(optional)*** Default width and height of each widget in the dashboard. *(default:* `[370, 340]` *)*

5.2 Multiple Dashboards

To initialize a multiple dashboards you need create a DashboardSet object and pass valid options as shown below:

```
var dashboardSet = new DashboardSet();
```

DashboardSet methods

addDashboard To add a new Dashboard:

```
dashboardSet.addDashboard(name, options)
```

Where *name* is a string with the name of dashboard and *options* is a json object with the same format of the options of the *Dashboard* object.

getDashboard To get a Dashboard from the DashboardSet object:

```
dashboardSet.getDashboard(name)
```

Swap between dashboards

To swap between dashboards need to press the *ctrl* key to display the menu.

CHAPTER 6

Widgets

To place widgets in your dashboard you need create a javascript file, where you call each widget that you need to place with the correct options, each widget provide two events that you can call in any javascript file to update the widget data or render the widget with the data that has.

For example if you create a number widget

```
var dashboard = new Dashboard();

...

dashboard.addWidget('example_widget', 'Number', {
    getData: function () {
        this.data = {
            title: 'Current Valuation',
            more_info: 'In billions',
            updated_at: 'Last updated at 14:10',
            change_rate: '64%',
            value: '$35'
        };
        dashboard.publish('example_widget/render');
    }
});
```

Then you can publish in any moment the events `dashboard.publish('example_widget/render')` to update the DOM of the widget and `dashboard.publish('example_widget/getData')` to get new data of the widget.

Note that in this example the *getData* method will be executed each 1000 milliseconds because is the default value of *interval* option in a *Number* widget.

6.1 Clock Widget

This widget can display an specific day an hour.

Options

row Number of rows occupied by the widget. *(default: 1)*

col Number of columns occupied by the widget. *(default: 1)*

render Function responsible of modify the DOM elements of the widget.

data JSON object that represent the date and time in format

```
{
    time: 'hh:mm:ss',
    date: 'Month Day DD sYYYY'
}
```

getData Function responsible to update *data* value, this function is executed each time interval specified in *interval* variable. You can rewrite this function to get data from an external source. This function should call render event to update the widget. *(default: return the browser time in a valid JSON format)*

getWidget Return the DOM element that represent the widget.

interval Actualization interval of widget data on milliseconds. *(default: 500)*

6.2 Graph Widget

This widget can display a value with an associate graph as background.

Options

row Number of rows occupied by the widget. *(default: 1)*

col Number of columns occupied by the widget. *(default: 2)*

render Function responsible of modify the DOM elements of the widget.

renderGraph Function responsible of draw the graph in the widget using Rickshaw library.

data JSON object that represent the date and time in format

```
{
    data: [
            {x: /x0/, y: /y0/},
            {x: /x1/, y: /y1/}
            ...
          ],
    value: /string/
    title: /string/,
    more_info: /string/
}
```

getData Function responsible to update *data* value, this function is executed each time interval specified in *interval* variable. You can rewrite this function to get data from an external source. This function should call render event to update the widget. *(default: empty function)*

getWidget Return the DOM element that represent the widget.

interval Actualization interval of widget data on milliseconds. *(default: 1000)*

GraphWidget Class

To use import from `dashing.widgets.GraphWidget`.

```
class GraphWidget(Widget):
    title = ''
    more_info = ''
    value = ''
    data = []

    def get_title(self):
        return self.title
```

```
    def get_more_info(self):
        return self.more_info

    def get_value(self):
        return self.value

    def get_data(self):
        return self.data

    def get_context(self):
        return {
            'title': self.get_title(),
            'more_info': self.get_more_info(),
            'value': self.get_value(),
            'data': self.get_data(),
        }
```

6.3 List Widget

This widget can display a list of elements with an associate value.

Options

row Number of rows occupied by the widget. *(default: 2)*

col Number of columns occupied by the widget. *(default: 1)*

render Function responsible of modify the DOM elements of the widget.

data JSON object that represent the date and time in format

```
{
    data: [
            {/key0/: /value0/},
            {/key1/: /value1/}
            ...
          ],
    title: /string/,
    more_info: /string/,
    updated_at: /string/
}
```

getData Function responsible to update *data* value, this function is executed each time interval specified in *interval* variable. You can rewrite this function to get data from an external source. This function should call render event to update the widget. *(default: empty function)*

getWidget Return the DOM element that represent the widget.

interval Actualization interval of widget data on milliseconds. *(default: 10000)*

ListWidget Class

To use import from `dashing.widgets.ListWidget`.

```
class ListWidget(Widget):
    title = ''
    more_info = ''
    updated_at = ''
    data = []
```

```
    def get_title(self):
        return self.title

    def get_more_info(self):
        return self.more_info

    def get_updated_at(self):
        return self.updated_at

    def get_data(self):
        return self.data

    def get_context(self):
        return {
            'title': self.get_title(),
            'more_info': self.get_more_info(),
            'updated_at': self.get_updated_at(),
            'data': self.get_data(),
        }
```

6.4 Number Widget

This widget can display a value with another interesting information.

Options

row Number of rows occupied by the widget. *(default: 1)*

col Number of columns occupied by the widget. *(default: 1)*

render Function responsible of modify the DOM elements of the widget.

data JSON object that represent the date and time in format

```
{
    value: /string/,
    title: /string/,
    change_rate: /string/,
    more_info: /string/,
    updated_at: /string/
}
```

getData Function responsible to update *data* value, this function is executed each time interval specified in *interval* variable. You can rewrite this function to get data from an external source. This function should call render event to update the widget. *(default: empty function)*

getWidget Return the DOM element that represent the widget.

interval Actualization interval of widget data on milliseconds. *(default: 1000)*

NumberWidget Class

To use import from `dashing.widgets.NumberWidget`.

```
class NumberWidget(Widget):
    title = ''
    more_info = ''
    updated_at = ''
    change_rate = ''
```

```
    value = ''

    def get_title(self):
        return self.title

    def get_more_info(self):
        return self.more_info

    def get_updated_at(self):
        return self.updated_at

    def get_change_rate(self):
        return self.change_rate

    def get_value(self):
        return self.value

    def get_context(self):
        return {
            'title': self.get_title(),
            'more_info': self.get_more_info(),
            'updated_at': self.get_updated_at(),
            'change_rate': self.get_change_rate(),
            'value': self.get_value(),
        }
```