Flask-Administration Documentation

Release 0.1.42

Bradford Toney

Apr 23, 2017

Contents

CHAPTER 1

Project Goal

The idea behind the project is to build a dashboard for adminstration in a way that it's both useful for business and its useful for small companies, It's not meant to be a do everything dashboard right now. It's currently in an unusable state but I hope some of the main functionality comes around soon.

CHAPTER 2

Contents:

Installation

Requirements

You'll need Flask 0.8

For a more complete listing look at the requirements.pip

Installing flask-administration

There are several ways to install flask-administration:

- Automatically, via a package manager.
- Manually, by downloading a copy of the release package and installing it yourself.
- Manually, by performing a Git checkout of the latest code.

To install the stable version with pip run:

```
pip install flask-administration
```

Using easy_install:

```
easy_install -Z flask-administration
```

The -Z flag makes sure the package is installed without zipping it, zipping it would mangle the Flask packages

To install the lastest development version run the following commands:

```
git clone git://github.com/bluemoon/flask-administration.git
cd flask-administration
sudo python setup.py install
```

Basic Configuration

Examples

Usage

Basic Usage

The simplest example is adding it only as a blueprint:

```
from flask.ext.administration import index
app.register_blueprint(index.admin)
```

A basic example with event driver would look like this:

```
from flask import Flask
from flask.ext.administration.blueprints import admin, events

def create_app():
    app = Flask(__name__)
    app.register_blueprint(event_driver.event_blueprint)
    app.register_blueprint(index.admin, url_prefix='/admin')
    return app

if __name__ == '__main__':
    app = create_app()
    app.run(debug=True)
```

With login protection

Generating an event key

Go to the metrics panel

CHAPTER 3

API/Reference Docs

HTTP Log API

This will describe the logging endpoints for flask

HTTP Event API

Parameters

- **_k** : Key, keeps different API services unique and easier to sort
- **_n** : Name of event you want to record, this has a limit of 255 characters
- **_p** : The identity of the person doing the event, this will likely be the users email address or possibly a unique id, has a limit of 255 characters
- **_t** : Timestamp of the event in the unix epoch format, used to keep events unique. This should only be used if you want to create an event in the past.
- **_d** : Data encoded in base64 format, this should only be used with the /events endpoint

JSON Format

This is the format we will use for JSON, this is the example for a singular event:

```
{
  '_n' : 'event name',
  '_p' : 'person'
}
```

Or for multiple events we may use a list of dictionaries:

```
[
  {
    '_n' : 'event name',
    '_p' : 'person',
    'customProperty' : 'CustomDataHere'
  },
  {
    '_n' : 'second event name'
  }
]
```

Response

As this is intended to be a very lightweight event system, the results are simply sent back in the text/plain mimetype. All endpoints follow this response.

- **1** - Data was logged
- **0** - Error, data was not logged

Endpoint(GET): /e

Parameters:

required: **_n**

Any other properties you'd like:

such as in this example, gender=male:

```
/e?_p=bob@bob.com&_n=Signup&gender=male&_k=KEY
```

This request translates out to have the values,

- _p - bob@bob.com
- _n - Signup
- _k - KEY
- gender - male

Sending data as base64:

If you choose to send the data in this manner it must adhere to the JSON format outlined above and then be encoded in base64.:

```
/e?_d=eyJldmVudCI6ICJnYW1lIiwgInBy<... truncated>&_k=KEY
```

Endpoint(POST): /e

Accepts encoded forms and JSON with the key used in a manner such that the URL below is the URL that should be POSTed to.:

```
/e?_k=KEY
```

Endpoint(GET): /events

List all of the events:

```
/events
```

List with page limits of 25 per page on page 3 (un-implemented):

```
/events?limit=25&page=3
```

List between two times:

```
/events?before=1231292609.140918&after=1331292609.140918
```

Listing with a specific event name:

```
/events?name=YourEventName
```

Flask-Administration Modules

__init__

flask_administration.__init__.**__version__**()

flask_administration.__init__.**_get_admin_extension_dir**()
Returns the directory path of this admin extension. This is necessary for setting the static_folder and templates_folder arguments when creating the blueprint.

flask_administration.__init__.**create_adminitizr_blueprint**(*view_decorator=<function view_decorator>*, *template_folder=None*, *static_folder=None*)

flask_administration.__init__.**view_decorator**(*f*)

metrics

utils

ajax

main

dashboard

Javascript

dashboard

The dashboard is built on top of backbone.js

class `models.`**Gauge**`()`

metrics

class **events**(*key*, *endpoint*)

Arguments

- **key** (*string*) – Key provided by the admin panel
- **endpoint** (*string*) – The endpoint of the blueprint for example if the url to **/e** is **example.com/events/e** then your endpoint should be **example.com/events**

Dashboard Design

Designing an endpoint for the javascript to build the dashboard from

Enpoint JSON

The endpoint will a starting point for the javascript to be generated from:

```
{
  "dashboard" : [
    {
      "type": "bar",
      "autoupdate" : "30",
      "title" : "Bar Metric",
      "size": "3x5"
    },
  ],
    }
```

The available **types** for widgets are going to be roughly:

- bar - bar chart

- pie - pie chart
- count - a counter
- sum - a counter that is summing
- average - an averaging metric

Backend Design

The backend design will be made up by several parts, there will be the **gauge** which provides the specific data source and type, the **gauge cluster** which is made up of gauges, and the dashboard which will be made up of a cluster or several clusters.

Development Journal

3/19

Things that a dashboard would have.

- Twitter/Sentinment analysis.
- Timeline(facebook style)
- Server Status
- Metrics, lots of metrics, cohort, funnel etc.
- Integration with external services (stripe, github)

3/22

I have chosen to start building the dashboard with javascript, well it's really coffeescript but i digress. I've run into a small issue with backbone.js, the issue being that when i dynamically create an instance of a class it does not seem to run the event delegation, I am not sure if this is an issue with dynamically creating the instance but from what i've googled it should not be.

3/26

I tend to be having some problems with backbone because of the way I am doing it, but i think that this may be normal. I may look at just using Backbone.Models so that i can fetch the json from the server in a simple manner and not have to deal with getting and handling the request, Even still I am questioning whether or not I even need to use backbone, while it seems nice it may be something that is just useful and not essential.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

Symbols

C

E

F

M

V