trumpet Documentation

Release 0.2.6

Joseph Rawson

Aug 16, 2018

Contents

1 trumpet **3**
- 1.1 News . . . 3
- 1.2 Goals and Progress . . . 3
- 1.3 Old . . . 4
- 1.4 Features . . . 4
- 1.5 Credits . . . 4

2 Installation **5**
- 2.1 Stable release . . . 5
- 2.2 From sources . . . 5

3 Usage **7**

4 Contributing **9**
- 4.1 Types of Contributions . . . 9
- 4.2 Get Started! . . . 10
- 4.3 Pull Request Guidelines . . . 11
- 4.4 Tips . . . 11

5 Credits **13**
- 5.1 Development Lead . . . 13
- 5.2 Contributors . . . 13

6 History **15**
- 6.1 0.2.0 (2017-12-07) . . . 15

7 Developing trumpet without vagrant **17**
- 7.1 Setup . . . 17

8 Trumpet Static Resources **21**
- 8.1 CSS Framework . . . 21
- 8.2 Basic Javascript Libraries . . . 22

9 Older Web Views Documentation **23**
- 9.1 Webviews . . . 23

10 Indices and tables **25**

Contents:

CHAPTER 1

trumpet

Build a website with pyramid

- Free software: UNLICENSED
- Documentation: https://trumpet.readthedocs.io.

1.1 News

Trumpet is getting a bit closer to the original intended goal of being a set of building blocks and tools to help build a pyramid web application. Management of static resources has moved completely away from python. Compass is still being used to help manage the stylesheets. Webpack is being used to handle the javascript, as well as some css, fonts, etc. Currently, cookiecutter is being used to test generating project skeletons, replacing the previous scaffold.

The general concept is to have support for creating web applications with different hosting requirements. A creative use of cookiecutter templates may provide the ability to generate a pyramid site, a static application/site, or even a tree of static assets that can be used in many projects.

1.2 Goals and Progress

- user management
  - login/logout
  - administer users via REST
  - reflective sqlalchemy code, db should provide minimum user/password tables
- db support
  - common sqlalchemy code for all databases **complete**
  - request object with attached sessionmaker **completed by upstream scaffold**
- session management **obsolete?**

    - minimal use of cookies **completed by using JSON Web Tokens**
    - use access_token as parameter to all requests requiring authentication
    - policies for session management *now token policies*
        * sessions per user (configure number of sessions a user can have)
        * sessions per device (register devices to user?)
        * session duration
        * session timeout/expiration
- view classes
    - basic view classes to be used by all views
        * common methods *WIP*
        * app settings available *still debating usefulness, JSONAPI may be better*
    - base user aware view class
        * base class for requests that need auth
    - base cornice resource
    - base static resource
    - base page resource *this is almost good enough*
        * send the html page that runs the app **complete**
        * use template to fill the head with links and meta info
        * handle permissions for access to app *send auth_token as query param?*
- server side validation **still needed**
    - use colander to build schemas for validation (or JSONSchema?)
- integrate with job servers for long running jobs

1.3 Old

Remnants of the old README can be found [here](https://github.com/umeboshi2/trumpet/blob/master/docs/misc.md).

1.4 Features

- TODO
    - remember vobject and icalendar to make .vcf files, etc...

1.5 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install trumpet, run this command in your terminal:

```
$ pip install trumpet
```

This is the preferred method to install trumpet, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for trumpet can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/umeboshi2/trumpet
```

Or download the tarball:

```
$ curl  -OL https://github.com/umeboshi2/trumpet/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use trumpet in a project:

```
import trumpet
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/umeboshi2/trumpet/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

trumpet could always use more documentation, whether as part of the official trumpet docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/umeboshi2/trumpet/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *trumpet* for local development.

1. Fork the *trumpet* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/trumpet.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv trumpet
$ cd trumpet/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 trumpet tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/umeboshi2/trumpet/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_trumpet
```

CHAPTER 5

Credits

5.1 Development Lead

- Joseph Rawson <joseph.rawson.works@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.2.0 (2017-12-07)

- First release on PyPI.

CHAPTER 7

Developing trumpet without vagrant

7.1 Setup

I like to use virtualenvwrapper

```
sudo apt-get install virtualenvwrapper
mkvirtualenv trumpet
workon trumpet
pip install requests
```

Development packages are needed to install some of the python packages:

```
.. code:: sh
```

sudo apt-get install libpq-dev python-dev libjpeg62-dev libpng12-dev libfreetype6-dev liblcms1-dev python-requests libxml2-dev libxslt1-dev libssl-dev

Next we have to download build and prepare the static resources.

7.1.1 Setup Compass

Make sure rubygems is on your system:

```
sudo apt-get install rubygems
```

Setup local gem environment:

```
mkdir -p ~/local/gems
```

Add to ~/.bashrc:

```
#setup gems if directory exists
if [ -d ~/local/gems ]; then
    export GEM_HOME=~/local/gems
    export PATH=~/local/gems/bin:$PATH
fi
```

Source the bashrc or spawn another shell and install the gems:

```
gem install sass -v 3.2.18
gem install compass -v 0.12.2
gem install susy -v 1.0.9
gem install sassy-buttons -v 0.2.6
gem install bootstrap-sass -v 3.0.2.1
gem install compass-ui -v 0.0.5
```

7.1.2 Setup NodeJS

FIXME: Need better instructions for nodeenv.

Get nodejs for virtualenv

```
workon trumpet
pip install nodeenv
nodeenv -p
```

The last statement will download the latest stable version of nodejs and build it in the python virtual environment so that both virtual environments can be integrated together.

Install global nodejs packages

Then, install these packages globally:

```
npm install -g coffee-script
npm install -g grunt-cli
npm install -g bower
```

7.1.3 Get packages for grunt

In the project directory, get the packages for grunt:

```
npm install
```

7.1.4 Get bower components

Then install the bower packages:

```
bower install
```

Bower packages can contain whole git repositories, which can be excessive when deploying a python package of static resources. I have written a script that helps to deploy only what is needed from the bower components. The script is not very smart, but handles any bower package that points to a single file, or list of files very well.

run grunt

```
grunt
```

make package

```
python setup.py (develop/install/sdist)
```

CHAPTER 8

Trumpet Static Resources

8.1 CSS Framework

- Compass:
  Compass is the tool I use to generate my CSS resources. The CSS specification has no definitios for variables, forcing many web developers to make class names such as "green" and then add CSS code like this:

```
.green {
background-color: green;
}
```

But what the developer really needs is something more along this idea:

```
.warn {
background-color: $warning-background;
}
```

... which helps to simplify the structure of the CSS and remove some of the bad hacks that are used to workaround the deficiencies of the CSS specification.

- Susy(Unused): Susy is a grid layout system that will allow for responsive webpages. I am not using this anymore, as bootstrap is currently handling the responsive grid layout, but Susy is superior to bootstrap and since I am also using bootstrap-sass, I feel that I can eventually reimplement the bootstrap grid layout in Susy. UPDATE: I decided to use the bootstrap grid system for the time being.
- Sassy Buttons: This is a collection of mixins and defaults that help a developer make custom buttons very easily.
- Bootstrap for Sass: This wonderful package allows me to refrain from using the css that is provided with bootstrap and quickly make a custom version that I can integrate more closely with other objects on the page. Having bootstrap in this form allows me to adjust how bootstrap operates and allows me to only choose the parts I need (Currently everything is included).
- FontAwesome: Instead of just using the basic css, I have chosen to use the fontawesome-sass distribution. This provides scalable vector icons to websites.

- Compass UI: This compass plugin provides the ability to generate jQueryUI themes with a minimum of effort. I have spent hours on the themeroller before trying to create a custom theme that would match the general colors that I use on a web page. With this plugin, all I have to do is set the variables to correspond to the color variables that I use elsewhere on the page and I instantly get themed widgets that don't look like they came from another site.

8.2 Basic Javascript Libraries

- Requirejs: Required.
- jQuery: jQuery is a very good for selecting and maninpulating elements in the DOM.
- jQuery User Interface: jQueryUI is used for the fullcalendar widget, as well as for dialog boxes and other user interface elements that aren't used through boostrap. The corresponding styles are maintained with compass.
- Bootstrap v3: Bootstrap is a CSS/Javascript framework used to help make responsive websites. Bootstrap was selected to be used in order to serve to mobile devices. The CSS is handled through compass with bootstrap-sass.
- Underscore.js: Underscore is a library full of useful utilities, and like jqueryui, is depended upon by other javascript libraries I use.
- Backbone.js: Backbone is an excellent library that provides an api to make very rich views tied to models that are seamlessly synchronized with the server via a REST interface.
- FullCalendar: FullCalendar is a very good library that provides an interactive calendar where events can be retrieved dynamically and grouped, colored, or otherwised styled in many ways. The calendar provides monthly, weekly, and daily view models to interact with.
- Ace Editor: The ACE editor is a good text editor that is very useful for editing html, css, java/coffee scripts, and other formats that aren't being used yet.
- CoffeeScript: I am currently experimenting executing coffeescript on the client using the browser to compile the code. While compilation is generally quick on the browser, the size of the compiler (196KB, and already minified) encourages me to consider implementing server side compilation.
- Teacup: "Teacup is templates in CoffeeScript." – nuff said http://en.wikipedia.org/wiki/Domain-specific_language

CHAPTER 9

Older Web Views Documentation

9.1 Webviews

A Webview is a name created to denote a single page application. Trumpet is being geared to become a library to help create websites where most/all pages are apps. I have managed to shrink the html page served by pyramid to a very small head with one script tag loading requirejs and pointing to the loader for the app. I would love to use another name for this, but naming is probably the hardest part of development.

9.1.1 Page Layouts

Before switching to the SPA paradigm of thought, every page was rendered with a template that depended on the presence of a layout model. The layout model was simply an object with attributes that was applied to a mako template. The type of each attribute is either text or mako template. One of my original goals was to store the main layout template in the database, as well as css and javascript, to allow the end user/admin to customize the site without updating the code.

9.1.2 Server vs. Client

Mako templates are very powerful, allowing the author to wield the full power of python when rendering the template. In fact, the templates are versatile enough to bypass writing code in the view callable and put all the logic in the template, although this is generally not the wisest thing to do. The largest problem with using the mako templates is that the code is executed server side, preventing me from being able to protect the service from mistakes or malice. I thought about using a more restrictive template system, but soon realized that the inherent problem was the server side rendering, and the server templates would either have to be too limited to be of more than cosmetic use, or flexible enough to bypass the policy of the server.

This is where client side templating comes to the rescue. With client side templates, it is far more difficult to endanger the service that is being provided. I am expecting the worst of the problems to be dysfunctional pages, although the admin pages that edit the templates should always work. I also see a very slim (I hope!) possibility that bad templates could cause a denial of service, but I don't expect this to be a problem that occurs often.

Being less familiar with javascript than with python, I had to search and compare templating styles. I started with underscore templates, but found them to be very limiting. I then started using EJS templates, and found them to be very similar to mako templates, although not as versatile. Nevertheless, I settled on using EJS templates for trumpet.

After believing I was happy with using EJS templates, I was looking around the stackoverflow forums to find a solution that I was having with CoffeeScript and learned about teacup. Teacup is a domain specific language that works very well as a templating system. With teacup, not only do you have the full flexibility of javascript when rendering the template, you are also using coffeescript to define the template, which is far more elegant than anything else I have seen on either the python or javascript sides. I did like how concise jade is, but I feel that teacup will be a better fit for trumpet.

9.1.3 REST

The general idea behind REST isn't really hard to understand. It was having to unlearn the way by which much of the web had already been operating for years. I spent many years knowing nothing of PUT and DELETE, but only being familiar with GET and POST, which I naively treated (loosely) as read/write methods. Now that I look back (and not very far either) I was often using GET to perform both deletions of single objects, as well as attaching relations together.

After learning REST, my ability to write arbitrary url's to perform a function has been severely hampered, and this is a good thing. I now have only four verbs that I am able to use, and I am completely restricted from putting verbs in the url, or even identifying the url as an action. These restrictions help keep the web services well structured and coherent. In fact, a good REST API decouples the server from the browser, allowing a larger variety of clients to have access to the services.

9.1.4 Static Resources

Managing static resources can become very messy the more involved a project becomes in using them. The very large variety of javascript libraries and css frameworks available can be overwhelming. Making sure that everything fits together and works can be an arduous task. Tracking upstream dependencies is probably a bit more difficult for a python/pyramid programmer than it is for a person using rails or nodejs. I had been (and I am currently still) using fanstatic to help manage these resources. There are quite a few prepackaged libraries depending on fanstatic available on the Python Package Index. These packages don't seem to be in much use, and after updating quite a few of them myself, I decided to wean myself away from fanstatic. I am currently investigating webassets, which seems to be a far more robust and capable asset manager.

Moreover, and more especially with css, it can become very time consuming to modify two or more upstream css resources to match the general style of your page.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search