coffeestats$_{p r o j e c t D o c u m e n t a t i o n}$

Release 0.1.26

ChangeToMyName

Oct 06, 2018

Contents

1 Acknowledgements **3**

2 License **5**

3 Welcome to coffeestats's documentation! **7**
- 3.1 Development . . . 7
- 3.2 Tests . . . 10
- 3.3 Deployment . . . 11
- 3.4 REST API version 1.0 . . . 11
- 3.5 Code documentation . . . 13
- 3.6 Credits . . . 17

4 Indices and tables **19**

Python Module Index **21**

HTTP Routing Table **23**

This is the Django port of coffeestats. The port was started because of the PHP uglyness. Coffeestats is the software running at https://coffeestats.org/. Coffeestats allows registered users to track their caffeine usage (currently coffee and mate). The site provides nice charts with aggregated data of the current user's as well as other user's caffeine consumption.

CHAPTER 1

Acknowledgements

Thanks to all contributors to this project.

CHAPTER 2

License

Coffeestats is licensed under the terms of the MIT license:

```
The MIT License (MIT)

Copyright (c) 2013-2016 Florian Baumann, Jan Dittberner, Holger Winter,
                        Jeremias Arnstadt

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
the Software, and to permit persons to whom the Software is furnished to do so,
subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

CHAPTER 3

Welcome to coffeestats's documentation!

Contents:

3.1 Development

3.1.1 Getting the coffeestats source

To start development on coffeestats you have several options for getting a working environment. Everything starts with a git clone:

```
git clone https://github.com/coffeestats/coffeestats-django.git
```

3.1.2 Development environment setup

We recommend using Vagrant to have a completely isolated working environment. You can also use virtualenv if you don't want the overhead of a full virtual machine.

If you do not use Vagrant you are on your own when it comes to database setup and definition of environment variables.

Vagrant

To use Vagrant you can just run:

```
vagrant up
```

from within your git working copy. Just wait a few minutes (depending on the speed of your network connection and system performance) and you will have a running coffeestats instance available at http://localhost:8080/.

You can then just work with the files in your working copy. If you want to perform service restarts or any other system administration in your coffeestats virtual machine you can use:

```
vagrant ssh
```

A fresh Vagrant VM has everything setup and all dependendencies installed in a virtualenv in `~vagrant/coffeestats-venv/`. If you need to update the dependencies you can use:

```
sudo salt-call state.highstate
```

The Salt invocation will take care of restarting uwsgi and nginx if needed.

Virtualenv

If you want to avoid the overhead of a virtual machine you can also use virtualenv to setup your development environment.

You will need a PostgreSQL database and have to take care of setting the necessary environment variables for the Django settings yourself. Look at the Salt state descriptions to get an idea what has to be done.

Virtualenv Only

First, make sure you are using virtualenv. Once that's installed, create your virtualenv:

```
virtualenv ~/coffeestats-venv
cd coffeestats && add2virtualenv `pwd`
```

Use the following command to work with the virtual environment later:

```
. ~/coffeestats-venv/bin/activate
```

Virtualenv with virtualenvwrapper

In Linux and Mac OSX, you can install virtualenvwrapper which will take care of managing your virtual environments and adding the project path to the *site-directory* for you:

```
mkvirtualenv coffeestats-dev
cd coffeestats && add2virtualenv `pwd`
```

To work with the virtual environment later use:

```
workon coffeestats-dev
```

Installation of dependencies

If you use a virtual environment you have to install the necessary dependencies. You need Python and PostgreSQL development headers installed before the installation of the development dependencies will work. On Debian based systems you can use apt to install both:

```
sudo apt-get update
sudo apt-get install libpq-dev python-dev
```

Development dependencies are defined in `requirements/local.txt`. Use the following command to install the dependencies in your currently activated environment:

```
pip install -r requirements/local.txt
```

3.1.3 Development session

If you are using Vagrant as recommended you can start a development session by opening a terminal and an editor session inside of your coffeestats clone. In the terminal session you run:

```
vagrant up
# ... wait for the VM to start
vagrant ssh
# ... should now be logged in to your vagrant VM
. csdev.sh
. coffeestats-venv/bin/activate
cd /vagrant/coffeestats
```

If you are not familiar with Django you should start with the Django tutorial.

3.1.4 Directory structure

. base directory with .gitignore, .travis.yml, CONTRIBUTORS.txt, LICENSE.txt, README.txt, Vagrantfile

coffeestats base directory for the project code and other project files

assets directory for static files to be served by a web server. This directory is populated by **manage.py collectstatic**

caffeine directory containing the caffeine app. This app contains the main model classes, code for generating statistics as well as the views used to display the web user interface

caffeine_api_v1 directory containing the *REST API v1.0* implementation

coffeestats directory containing the configuration code for coffeestats like the main URL configuration, settings for different environments (local, test, production) and the WSGI application entry point

core directory containing code to be used by multiple Django apps

functional_tests directory containing functional tests based on Selenium

static directory containing subdirectories with static assets for coffeestats

css Sass sources as well as generated and hand-written CSS

common common styling like fonts, colors, icons and mediaqueries

components Sass components / pageareas which will be imported and compiled in the caffeine.scss

fonts font files

images icons and other image files

js JavaScript libraries and a common scripts.js (app specific JavaScript code is kept in `static/<appname>/js` subdirectories of the corresponding apps)

templates directory containing the HTML and email text templates

docs directory containing the Sphinx documentation source

requirements directory containing pip requirements files

salt directory containing the Salt states and pillars that are used to provision the Vagrant VM

3.1.5 CSS generation with Sass

We use Sass to generate our Cascading Style Sheets (CSS) file. Sass is a CSS generator feeded by a CSS like language. On Debian systems you can install Sass by running:

```
sudo apt-get install ruby-sass
```

On other systems with a Ruby Gems installation you can run:

```
gem install sass
```

During development you can continuosly run **sass** to generate the `coffeestats/static/css/caffeine.css`:

```
cd coffeestats/static
sass --watch css/caffeine.scss:css/caffeine.css
```

You can also run **sass** before committing your changes on `coffeestats/static/css/caffeine.scss` manually:

```
cd coffeestats/static
sass css/caffeine.scss:css/caffeine.css
```

Warning: Please be aware that all changes in `css/caffeine.css` you make manually will be overwritten the next time somebody runs Sass. You should always modify `css/caffeine.scss` instead.

SASS files which look like this: _filename.scss are for imports in other sass files. Sass won't generate own css files of them.

3.2 Tests

Coffeestats comes with a full suite of unit and functional tests. The unit tests are available in each app's tests module. To run the test suite you need the test dependencies installed (both `requirements/test.txt` and `requirements/local.txt` include the list of necessary Python modules).

When all test requirements are met you can run all tests using:

```
cd coffeestats
coverage run --branch manage.py test
```

You can get a coverage report with:

```
coverage report -m
```

Note: The functional tests in the `coffeestats/functional_tests` directory need a Firefox and a graphical display. If you want to run the tests in a headless environment you can use xvfb. This approach is also used on Travis CI

3.2.1 Continous Integration

The coffeestats test suite is run on Travis CI after every push to the master branch of the main github repository, code coverage is reported to the Coveralls service after successful builds.

3.3 Deployment

3.3.1 Salt states

The live deployment for https://coffeestats.org/ is done using Salt states. The setup is similar to the setup described in salt/roots/salt/coffeestats.

3.3.2 Manual deployment

You have to setup a WSGI capable web server. We recommend to use uwsgi and nginx. You should use virtualenv to isolate the application code and its dependencies from the rest of your system.

Requirements

The following preconditions have to be fulfilled for a manual deployment:

- Python 2.7.x
- PostgreSQL >= 9.1
- a WSGI capable web server

Database setup

We use Django's ORM and you can simply setup your database using:

```
python manage.py syncdb --migrate
```

3.4 REST API version 1.0

Coffeestats provides a small REST API to be used by third party applications. The API is described with some example curl calls below.

3.4.1 Base URI

The API is hosted at /api/v1/ and provides several resources that are described in detail *below*.

```
curl https://coffeestats.org/api/v1/$Resource
```

3.4.2 Authentication

The username and the user's on-the-run token are used for API call authentication. You can see both used as GET parameters for the bookmarkable on-the-run link on you profile page.

```
curl -X POST -d "u=user&t=yourtokenhere" https://coffeestats.org/api/v1/$Resource
```

This is an incomplete example. See *below* for detailed resource descriptions.

3.4.3 Resources

random-users

POST /api/v1/random-users

Query a set of random users

Form Parameters

- **u** – user name
- **t** – on-the-run token
- **count** – optional number of users

Response Headers

- Content-Type – *text/json*

Status Codes

- 200 OK – all is ok, body contains a list of users
- 403 Forbidden – authentication required

curl example

```
curl -X POST -d "u=user&t=yourtokenhere" https://coffeestats.org/api/v1/random-users␣
↪|python -mjson.tool
[
  {
    "coffees": "42",
    "location": "baz",
    "mate": "0",
    "name": "foobar",
    "profile": "https://coffeestats.org/profile?u=foobar",
    "username": "foobar"
  },
  ...
]
```

add-drink

POST /api/v1/add-drink

Submit the consumption of a drink (mate or coffee)

Form Parameters

- **u** – user name
- **t** – on-the-run token

- **beverage** – `mate` or `coffee`
- **time** – timestamp in a format with ISO 8601 date and time i.e. 2014-02-24 19:46:30

Response Headers

- Content-Type – *text/json*

curl example

```
curl -X POST -d "u=user&t=yourtokenhere&beverage=mate&time=2014-02-24 19:46:30"␣
↪https://coffeestats.org/api/v1/add-drink |python -mjson.tool
{
  "success": true
}
```

3.5 Code documentation

3.5.1 Caffeine app

caffeine.admin

Django admin classes for the caffeine app.

class caffeine.admin.**UserCreationForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

A form for creating new users. Includes all the required fields, plus a repeated password.

clean_password2()

Check that the two password entries match.

save(*commit=True*)

Save the provided password in hashed format.

class caffeine.admin.**UserChangeForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)

A form for updating users. Includes all the fields on the user, but replaces the password field with admin's password hash display field.

class caffeine.admin.**CaffeineUserAdmin**(*model*, *admin_site*)

Custom admin page for users.

add_form

alias of *UserCreationForm*

form

alias of *UserChangeForm*

caffeine.authbackend

Custom authentication backend for coffeestats.

class caffeine.authbackend.**LegacyCoffeestatsAuth**
Authentication backend for passwords generated by the original coffeestats PHP implementation.

caffeine.forms

Forms for coffeestats.

class caffeine.forms.**CoffeestatsRegistrationForm**(**args*, ***kwargs*)
This is the form for registering new users.

clean_username()
Validate that the username is alphanumeric and is not already in use.

class caffeine.forms.**SettingsForm**(*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*, *use_required_attribute=None*, *renderer=None*)
This is the form for changing a user's settings.

clean()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

clean_email()
Validate that the supplied email address is unique for the site.

class caffeine.forms.**SelectTimeZoneForm**(**args*, ***kwargs*)
This is the form for selecting a user's time zone.

class caffeine.forms.**SubmitCaffeineForm**(*user*, *ctype*, **args*, ***kwargs*)
This is the form for new caffeine submissions.

clean()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

caffeine.middleware

class caffeine.middleware.**EnforceTimezoneMiddleware**(*get_response*)
Middleware to enforce that users have a time zone set.

caffeine.models

class caffeine.models.**User**(**args*, ***kwargs*)
User model.

exception DoesNotExist

exception MultipleObjectsReturned

has_usable_password()
Checks whether the current user has either an old password hash or a valid new password hash.

set_password(*password*)
Sets the password and creates the authentication token if it is not set.

class caffeine.models.**CaffeineManager**

Manager class for Caffeine.

daily_caffeine()

Return series of daily coffees and mate in current month for all users.

Returns result dictionary

daily_caffeine_for_user(*user*)

Return series of daily coffees and mate in current month for user profile.

Parameters **user** (User) – user instance

Returns result dictionary

get_csv_data(*drinktype*, *user*)

Get user records for a specific drink type in CSV format.

Parameters

- **drinktype** (*str*) – drink type
- **user** (User) – user instance

Returns list of records in CSV format

hourly_caffeine()

Return series of hourly coffees and mate on current day for all users.

Returns result dictionary

hourly_caffeine_for_user(*user*)

Return series of hourly coffees and mate on current day for user profile.

Parameters **user** (User) – user instance

Returns result dictionary

hourly_caffeine_for_user_overall(*user*)

Return a series of hourly caffeinated drinks for the whole timespan of a user's membership.

Parameters **user** (User) – user instance

Returns result dictionary

hourly_caffeine_overall()

Return a series of hourly caffeinated drinks for the whole lifetime of the site.

Returns result dictionary

latest_caffeine_for_user(*user*, *count=10*)

Return the latest caffeine entries for the given user.

Parameters

- **user** (User) – user instance
- **count** (*int*) – number of entries

Returns list of Caffeine instances

monthly_caffeine_for_user(*user*)

Return a series of monthly coffees and mate in the current month for user profile.

Parameters **user** (User) – user instance

Returns result dictionary

monthly_caffeine_overall()
Return a series of monthly coffees and mate in the current month for all users.

Returns result dictionary

total_caffeine()
Return total caffeine for all users.

Returns result dictionary

total_caffeine_for_user(*user*)
Return total caffeine for user profile.

Parameters **user** (User) – user instance

Returns result dictionary

weekdaily_caffeine_for_user_overall(*user*)
Return a series of caffeinated drinks per weekday for the whole timespan of a user's membership.

Parameters **user** (User) – user instance

Returns result dictionary

weekdaily_caffeine_overall()
Return a series of caffeinated drinks per weekday for the whole lifetime of the site.

Returns result dictionary

class caffeine.models.**Caffeine**(**args*, ***kwargs*)
Caffeinated drink model.

exception DoesNotExist

exception MultipleObjectsReturned

clean()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

class caffeine.models.**ActionManager**
Manager class for actions.

class caffeine.models.**Action**(**args*, ***kwargs*)
Action model.

exception DoesNotExist

exception MultipleObjectsReturned

caffeine.templatetags.caffeine

caffeine.templatetags.caffeine.**publicurl**(*context*, *username=None*)

caffeine.templatetags.caffeine.**ontherunurl**(*context*, *user=None*)

caffeine.templatetags.caffeine.**messagetags**(*value*, *tag*)

caffeine.views

Todo: document caffeine.views (there are import errors in django-registration)

3.5.2 Caffeine API v1 app

caffeine_api_v1.views.**add_drink**(*request*, *userinfo*, *messages*, **args*, ***kwargs*)
Submit a caffeinated beverage.

Parameters

- **request** (*HttpRequest*) – POST request
- **userinfo** (User) – *caffeine.models.User*
- **messages** (*dict*) – message dictionary

Returns messages array or django.http.HttpResponseBadRequest

caffeine_api_v1.views.**api_token_required**(*func*)
Decorator to force authentication with an on-the-run token.

caffeine_api_v1.views.**random_users**(*request*, ***_*)
Return a list of random user data.

Parameters **request** (*HttpRequest*) – POST request

Returns list of users

3.5.3 Core app

core.utils.**json_response**(*func*)
Decorator for wrapping the result of a function in a JSON response object.

3.6 Credits

Coffeestats was originally implemented in PHP by Florian Baumann and others, the code base got some major improvements by Jan Dittberner in 2013 but still used PHP. Jeremias Arnstadt contributed SASS based styling and general design improvements.

In November 2013 Florian released the coffeestats PHP code under the MIT license on Github. Clemens Lang contributed a REST API for submitting entries in February 2014.

Florian hosted coffeestats on an OpenBSD machine until 20th of June 2014.

During the Chemnitzer-Linux Tage 2014 Jan Dittberner decided to do a reimplementation of Coffeestats in Django. Jeremias did a complete redesign and the new site went live on 20th of June 2014 on one of Jan's Debian GNU/Linux machines.

3.6.1 Contributors

Here is a (hopefully) complete list of contributors:

- Clemens Lang

- Florian Baumann
- Holger Winter
- Jan Dittberner
- Jeremias Arnstadt
- cryzed

Please tell us if you think we missing to acknowledge your contribution.

Todo: document caffeine.views (there are import errors in django-registration)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/coffeestats/checkouts/latest/docs/code.rst, line 52.)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

caffeine.admin, 13
caffeine.authbackend, 13
caffeine.forms, 14
caffeine.middleware, 14
caffeine.models, 14
caffeine_api_v1.views, 17
core.utils, 17

HTTP Routing Table

/api

`POST /api/v1/add-drink`, 12
`POST /api/v1/random-users`, 12

Index

A

Action (class in caffeine.models), 16
Action.DoesNotExist, 16
Action.MultipleObjectsReturned, 16
ActionManager (class in caffeine.models), 16
add-drink, 12
add_drink() (in module caffeine_api_v1.views), 17
add_form (caffeine.admin.CaffeineUserAdmin attribute), 13
API, 11
api_token_required() (in module caffeine_api_v1.views), 17
authentication, 11

C

Caffeine (class in caffeine.models), 16
caffeine.admin (module), 13
caffeine.authbackend (module), 13
caffeine.css, 10
Caffeine.DoesNotExist, 16
caffeine.forms (module), 14
caffeine.middleware (module), 14
caffeine.models (module), 14
Caffeine.MultipleObjectsReturned, 16
caffeine.scss, 10
caffeine_api_v1.views (module), 17
CaffeineManager (class in caffeine.models), 14
CaffeineUserAdmin (class in caffeine.admin), 13
clean() (caffeine.forms.SettingsForm method), 14
clean() (caffeine.forms.SubmitCaffeineForm method), 14
clean() (caffeine.models.Caffeine method), 16
clean_email() (caffeine.forms.SettingsForm method), 14
clean_password2() (caffeine.admin.UserCreationForm method), 13
clean_username() (caffeine.forms.CoffeestatsRegistrationForm method), 14
CoffeestatsRegistrationForm (class in caffeine.forms), 14
core.utils (module), 17

D

daily_caffeine() (caffeine.models.CaffeineManager method), 15
daily_caffeine_for_user() (caffeine.models.CaffeineManager method), 15

E

EnforceTimezoneMiddleware (class in caffeine.middleware), 14

F

form (caffeine.admin.CaffeineUserAdmin attribute), 13

G

get_csv_data() (caffeine.models.CaffeineManager method), 15

H

has_usable_password() (caffeine.models.User method), 14
hourly_caffeine() (caffeine.models.CaffeineManager method), 15
hourly_caffeine_for_user() (caffeine.models.CaffeineManager method), 15
hourly_caffeine_for_user_overall() (caffeine.models.CaffeineManager method), 15
hourly_caffeine_overall() (caffeine.models.CaffeineManager method), 15

J

json_response() (in module core.utils), 17

L

latest_caffeine_for_user() (caffeine.models.CaffeineManager method), 15

LegacyCoffeestatsAuth (class in caffeine.authbackend), 13

M

messagetags() (in module caffeine.templatetags.caffeine), 16
monthly_caffeine_for_user() (caffeine.models.CaffeineManager method), 15
monthly_caffeine_overall() (caffeine.models.CaffeineManager method), 15

O

ontherunurl() (in module caffeine.templatetags.caffeine), 16

P

publicurl() (in module caffeine.templatetags.caffeine), 16

R

random-users, 12
random_users() (in module caffeine_api_v1.views), 17
resources, 12
REST, 11

S

Sass, 10
save() (caffeine.admin.UserCreationForm method), 13
SelectTimeZoneForm (class in caffeine.forms), 14
set_password() (caffeine.models.User method), 14
SettingsForm (class in caffeine.forms), 14
SubmitCaffeineForm (class in caffeine.forms), 14

T

total_caffeine() (caffeine.models.CaffeineManager method), 16
total_caffeine_for_user() (caffeine.models.CaffeineManager method), 16

U

User (class in caffeine.models), 14
User.DoesNotExist, 14
User.MultipleObjectsReturned, 14
UserChangeForm (class in caffeine.admin), 13
UserCreationForm (class in caffeine.admin), 13

W

weekdaily_caffeine_for_user_overall() (caffeine.models.CaffeineManager method), 16
weekdaily_caffeine_overall() (caffeine.models.CaffeineManager method), 16