FEC Website Documentation

Release 1.1.3

2014, Pavan Rikhi

December 02, 2015

Contents

CHAPTER 1

Introduction

This is the documentation for the FEC's website. It runs on the Mezzanine CMS, the Django Framework and the Python Programming Language.

This documentation is useful for the website's contributors and maintainers, it will teach you *Getting Started*, *Contribution Guidelines*, `Deployment` and *the API*.

CHAPTER 2

Getting Started

This section covers setting up a dev machine to work on the website's source code.

2.1 Grab the Source

Use `git` to clone the current source code repository:

```
$ git clone http://bugs.sleepanarchy.com/fec.git
```

Switch over to the `develop` branch, where new development occurs. Only completed versions should be merged directly into the `master` branch.

2.2 Install Prerequisites

You should install python 2 and pip via your package manager.

On Arch Linux:

```
$ sudo pacman -S python2 python2-pip
```

On Slackware:

```
$ sudo /usr/sbin/slackpkg install python
$ wget https://raw.github.com/pypa/pip/master/contrib/get-pip.py
$ sudo python get-pip.py
```

On Debian/Ubuntu:

```
$ sudo apt-get install python-pip
```

Optionally you may want to install virtualenv and virtualenvwrapper to manage and isolate the python dependencies.

```
$ sudo pip install virtualenv virtualenvwrapper
```

Make sure to do the initial setup for virtualenv:

```
$ export WORKON_HOME=~/.virtualenv/
$ mkdir -p $WORKON_HOME
$ source virtualenvwrapper.sh
```

Then you may create an environment for the FEC dependenies:

```
$ mkvirtualenv FEC
```

You may then install dependencies into this virtual environment. There are multiple tiers of dependencies:

- `base` - minimum requirements needed to run the application
- `test` - requirements necessary for running the test suite
- `local` - development prerequisites such as the debug toolbar and documentation builders
- `production` - all packages required for real world usage

A set of dependencies may be installed via pip:

```
$ workon FEC
$ pip install -r requirements/local.txt
```

2.3 Configuration

Some settings are set through environmental variables instead of files. These include settings with sensitive information, and allows us to keep the information out of version control.

You may set these variables directly in the terminal or add them to your virtualenv's `activate` script:

```
$ DB_USER='prikhi' DB_NAME='FEC' ./manage.py <command>
$ export DB_NAME='FEC'
$ ./manage.py <command>
```

The required environmental variables are `DJANGO_SECRET_KEY`, `DB_NAME` and `DB_USER`.

2.4 Create the Database

Create the initial database by running `createdb`:

```
$ export DJANGO_SETTINGS_MODULE=core.settings.local
$ cd fec
$ ./manage.py createdb
```

2.5 Running

You should now be able to run the server:

```
$ ./manage.py runserver
```

You can visit `http://localhost:8000/` in a web browser to check the site out.

2.6 Testing

After making changes, run the test suite with `py.test`:

```
$ cd fec
$ py.test
```

Every test should pass before you commit your changes.

CHAPTER 3

Contribution Guidelines

This section describes development standards and best practices within the project.

3.1 Code Conventions

The **PEP 8** is our baseline for coding style.

In short we use:

- 4 spaces per indentation
- 79 characters per line
- One import per line, grouped in the following order: standard library, 3rd party imports, local application imports
- One statement per line
- Docstrings for all public modules, functions, classes and methods.

The following naming conventions should be followed:

- Class names use CapitalWords
- Function names are lowercase, with words separated by underscores
- Use `self` and `cls` for first argument to instance and class methods, respectively.
- Non-public methods and variables should be prefixed with an underscore
- Constants in all uppercase.

Code should attempt to be idiomatic/pythonic, for example:

- Use list, dict and set comprehensions.
- Test existence in a sequence with `in`.
- Use `enumerate` instead of loop counters.
- Use `with ... as ...` for context managers.
- Use `is` to compare against `None` instead of ==.
- Use parenthesis instead of backslashes for line continuations.

For more information and full coverage of conventions, please read **PEP 8**, **PEP 257**, **PEP 20** and the Django Coding Style Documentation.

You can use `prospector` to check your code style:

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

3.2 Version Control

We use Git as our Version Control System.

3.2.1 Branches

We have 2 long-term public branches:

- `master` - The latest stable release. This branch should be tagged with a new version number every time a branch is merged into it.
- `develop` - The release currently in development. New features and releases originate from this branch.

There are also multiple short-term supporting branches:

- `hotfix` - Used for immediate changes that need to be pushed out into production. These branches should originate from `master` and be merged into `master` and either the `develop` or current `release` if one exists.
- `feature` - Used for individual features and bug fixes, these branches are usually kept on local development machines. These should originate from and be merged back into `develop`.
- `release` - Used for preparing the `develop` branch for merging into `master`, creating a new release. These branches should originate from `develop` and be merged back into `develop` and `master`. Releases should be created when all new features for a version are finished. Any new commits should only contain code refactoring and bug fixes.

This model is adapted from A Successful Git Branching Model, however we use a linear history instead of a branching history, so the `--no-ff` option should be omitted during merges.

3.2.2 Commit Messages

Commit messages should follow the format described in A Note About Git Commit Messages. They should generally follow the following format:

```
[TaskID#] Short 50 Char or Less Title

Explanatory text or summary describing the feature or bugfix, capped
at 72 characters per line, written in the imperative.

Bullet points are also allowed:

* Add method `foo` to `Bar` class
* Modify `Base` class to be abstract
* Remove `foobaz` method from `Bar` class
* Refactor `bazfoo` function

Refs/Closes/Fixes #TaskID: Task Name in Bug Tracker
```

For example:

```
[#367] Customize General Layout & Home Page

* Add the core app and a SeleniumTestCase class to the core.utils module
  to automatically start a live server and create a remote selenium
  driver.
* Add the functional_tests.general_page_tests module to test elements
  that should be on every page.
* Add the functional_tests.home_page_tests module to test elements that
  should appear on the home page.
* Add basic site customizations, like modifying the title, tagline and
  homepage content.
* Add requirements files for test and local environments.

* Modify manage.py to use the thefec module's settings files
* Move the functional tests into their own module.

* Remove the right sidebar.
* Remove the lettuce applications.
* Remove extraneous files from the thefec module.

Closes #367: Create Initial project
```

3.2.3 Workflow

The general workflow we follow is based on A Git Workflow for Agile Teams.

Work on a new task begins by branching from `develop`. Feature branch names should be in the format of `tasknumber-short-title-or-name`:

```
$ git checkout -b 142-add-account-history develop
```

Commits on this branch should be early and often. These commit messages are not permanent and do not have to use the format specified above.

You should fetch and rebase against the upstream repository often in order to prevent merging conflicts:

```
$ git fetch origin develop
$ git rebase origin/develop
```

When work is done on the task, you should rebase and squash your many commits into a single commit:

```
$ git rebase -i origin/develop
```

You may then choose which commits to reorder, squash or reword.

> **Warning:** Only rebase commits that have not been published to public branches. Otherwise problems will arise in every other user's local repository. NEVER rewrite public branches and NEVER force a push unless you know EXACTLY what you are doing, and have preferably backed up the upstream repository.

Afterwards, merge your changes into `develop` and push your changes to the upstream repository:

```
$ git checkout develop
$ git merge 142-add-account-history
$ git push origin develop
```

3.2.4 Preparing a Release

A Release should be forked off of develop into its own branch once all associated features are completed. A `release` branch should contain only bugfixes and version bumps.

1. Fork the `release` branch off of the `develop` branch:

```
$ git checkout -b release-1.2.0 develop
```

2. Branch, Fix and Merge any existing bugs.
3. Bump the version number and year in `setup.py` and `docs/source/conf.py`.
4. Commit the version changes:

```
$ git commit -a -m "Prepare v1.2.0 Release"
```

5. Create a new annotated and signed Tag for the commit:

```
   $ git tag -s -a v1.2.0

The annotation should contain the release's name and number, and optionally
a short description::

    Version 1.2.0 Release - Trip Entry Form

    This release adds a Trip Entry form for Communards and a Trip Approving
    form for Accountants.
```

6. Merge the branch into the `master` branch and push it upstream:

```
$ git checkout master
$ git merge release-1.2.0
$ git push origin master
$ git push --tags origin master
```

7. Make sure to merge back into the `develop` branch as well:

```
$ git checkout develop
$ git merge release-1.2.0
$ git push origin develop
```

8. You can then remove the `release` branch from your local repository:

```
$ git branch -d release-1.2.0
```

3.3 Version Numbers

Each release will be tagged with a version number, using the MAJOR.MINOR.PATCH Semantic Versioning format and specifications.

These version numbers indicate the changes to the public API.

The PATCH number will be incremented if a new version contains only backwards-compatible bug fixes.

The MINOR number is incremented for new, backwards-compatible functionality and marking any new deprecations. Increasing the MINOR number should reset the PATCH number to 0.

The MAJOR number is incremented if ANY backwards incompatible changes are introduced to the public API. Increasing the MAJOR number should reset the MINOR and PATCH numbers to 0.

Pre-release versions may have additional data appended to the version, e.g. `1.0.1-alpha` or `2.1.0-rc`.

The first stable release will begin at version 1.0.0, any versions before this are for initial development and should be not be considered stable.

For more information, please review the Semantic Versioning Specification.

3.4 Documentation

This documentation is written in reStructuredText and created using the Sphinx Documentation Generator. Sphinx's `autodoc` module is used to create the API specifications of the application by scraping docstrings(**PEP 257**).

Each class, function, method and global should have an accurate docstring for Sphinx to use.

Each feature or bug fix should include all applicable documentation changes.

To build the Documentation, install the prerequisites then run the make command to generate either html or pdf output:

```
$ pip install -r requirements/local.txt
$ cd docs/
$ make html; make latexpdf
```

The output files will be located in the `docs/_build` directory.

CHAPTER 4

Debian 7 (Wheezy) Deployment Guide

This section covers deploying the website on Debian 7(Wheezy). It assumes you have a working apache configuration and are logged in as `thefec`.

4.1 Install & Configure PostgreSQL

Start by installing the postgreSQL server & client. Development libraries are required for the python postgreSQL library:

```
$ sudo apt-get install postgresql postgresql-server postgresql-server-dev-9.1
```

Next, become the `postgres` user and create a new database and user:

```
$ sudo su - postgres
$ createuser -DERPS thefec      # No DB/user creation privleges, not a superuser, encrypt the passwo
$ createdb fec_website -O thefec
$ logout
```

4.2 Install Memcached

Memcached is used for caching. All you need to do is install it to get it working, it will automatically start a server at `127.0.0.1:11211`:

```
$ sudo apt-get install memcached
```

4.3 Enable Wheezy Backports & Install the LESS Compiler

`lessc` takes LESS source files and converts them to CSS. To get the package in Debian 7, we need to enable the `wheezy-backports` repository. To do this, add the following line to the end of `/etc/apt/sources.list`:

```
deb http://http.debian.net/debian wheezy-backports main
```

Then you can update your package list & install `lessc`:

```
$ sudo apt-get update
$ sudo apt-get -t wheezy-backports install node-less
```

4.4 Install & Create Python Virtual Environment

A virtual environment will let us separate our dependencies from the system's python libraries:

```
$ sudo apt-get install python-virtualenv
$ virtualenv ~/WebsiteEnv
```

We'll create a bash script at `~/load_website_env.sh` to set environmental variables that configure the website(database name, password, secret key, etc.):

```
# ~/load_website_env.sh
source ~/WebsiteEnv/bin/activate

export DB_NAME='fec_website'
export DB_USER='thefec'
export DB_PASS=YOUR_PASSWORD

export ALLOWED_HOST='www.thefec.org'
export DJANGO_SECRET_KEY=YOUR_SECRET_KEY
export DJANGO_SETTINGS_MODULE='fec.settings.production'
export CACHE_PREFIX='FECprod'
```

Since this contains our database user's password, we'll make sure only we can run/read/write it:

```
$ chmod 700 ~/load_website_env.sh
```

4.5 Download & Setup Application

First we'll need `git` to pull the source code and some image libraries:

```
$ sudo apt-get install git libjpeg-dev libfreetype6-dev
```

Activate our virtual environment, grab the source & install the python dependencies:

```
$ source ~/load_website_env.sh
$ cd ~
$ git clone http://bugs.sleepanarchy.com/fec.git ~/website
$ cd ~/website
$ pip install -r requirements/base.txt
```

Create the database schema and load the initial data if you have any:

```
$ cd ~/website/fec
$ ./manage.py migrate
$ ./manage.py loaddata ~/full_dump.json
```

Collect the static files & link it to our public HTML directory:

```
$ ./manage.py collectstatic
$ ln -s ~/website/fec/static ~/htdocs/static
```

4.6 Install & Configure Python Server

Dynamic requests will be served by the `uWSGI` server and proxied by apache. Static files like images, CSS and JavaScript will be served by apache.

Start by installing `uWSGI`:

```
$ sudo apt-get install uwsgi uwsgi-plugin-python
```

Note: You may want a newer version of uWSGI for page caching & gzipping support. There is no uWSGI package in `wheezy-backports` so you'll have to build the packages yourself. That's out of the scope of this guide, you should refer to the SimpleBackportCreation Page on the Debian Wiki. Once you've built the packages and have them on your server, install them using `dpkg` along with some dependencies for gzipping & uWSGI:

```
$ sudo apt-get install libpcre3-dev libz-dev
$ sudo apt-get -t wheezy-backports install libzmq3-dev
$ sudo dpkg -i libapache2-mod-uwsgi_2*.deb
$ sudo dpkg -i uwsgi-core_2*.deb
$ sudo dpkg -i uwsgi-plugin-python_2*.deb
$ sudo apt-get -f install
```

Add the following configuration to `/etc/uwsgi/apps-available/fec-website.ini`:

```
[uwsgi]
uid = thefec
gid = www-data
chdir = /home/thefec/website/fec

plugin = python2,transformation_gzip
pythonpath = /home/thefec/WebsiteEnv/lib/python2.7/site-packages/
pythonpath = /usr/lib/python2.7
virtualenv = /home/thefec/WebsiteEnv
no-site=True

socket = 127.0.0.1:3032
master = true
workers = 4
max-requests = 5000
vacuum = True

pidfile = /tmp/fec-website.pid
touch-reload = /tmp/fec-website.touch

env = DJANGO_SETTINGS_MODULE=fec.settings.production
env = DJANGO_SECRET_KEY=YOUR_SECRET_KEY
env = DB_NAME=fec_website
env = DB_USER=thefec
env = DB_PASS=YOUR_PASSWORD
env = ALLOWED_HOST=www.thefec.org
env = CACHE_PREFIX=FECprod
wsgi-file = /home/thefec/website/fec/fec/wsgi.py

# uWSGI v1.9+ only
# route to gzip if supported
route-if = contains:${HTTP_ACCEPT_ENCODING};gzip goto:mygzipper
route-run = last:

route-label = mygzipper
route = ^/$ gzip:
```

Link the file to `apps-enabled` to enable it, restart uwsgi, then `touch` the touch-file to restart the python server:

```
$ sudo ln -s /etc/uwsgi/apps-available/fec-website.ini /etc/uwsgi/apps-enabled/
$ sudo service uwsgi restart
$ touch /tmp/fec-website.touch
```

4.7 Configure Virtual Host

First we need to install the apache module for uWSGI:

```
$ sudo apt-get install libapache2-mod-uwsgi
```

Then add the following configuration to `/etc/apache2/sites-available/thefec.org.conf`:

```
<VirtualHost 72.249.12.147:80>
    ServerName www.thefec.org

    DocumentRoot /home/thefec/website/fec

    ErrorLog /home/thefec/logs/error_log
    CustomLog /home/thefec/logs/access_log common

    Alias /static /home/thefec/htdocs/static
    <Directory /home/thefec/htdocs/static>
        Options Indexes FollowSymLinks MultiViews
        allow from all
        AllowOverride All
    </Directory>

    # Redirect requests to the python server
    <Location "/">
        Options FollowSymLinks Indexes
        SetHandler uwsgi-handler
        uWSGISocket 127.0.0.1:3032
    </Location>
    # Except for requests to /static/
    <Location /static>
        SetHandler none
        allow from all
    </Location>

    # Cache all the things
    ExpiresActive On
    ExpiresByType text/html "access plus 5 minutes"
    ExpiresByType text/css "access plus 10 years"
    ExpiresByType text/javascript "access plus 10 years"
    ExpiresByType application/x-javascript "access plus 10 years"
    ExpiresByType text/javascript "access plus 10 years"
    ExpiresByType application/javascript "access plus 10 years"
    ExpiresByType image/jpg "access plus 10 years"
    ExpiresByType image/gif "access plus 10 years"
    ExpiresByType image/jpeg "access plus 10 years"
    ExpiresByType image/png "access plus 10 years"
    ExpiresByType image/x-icon "access plus 10 years"
    ExpiresByType image/icon "access plus 10 years"
    ExpiresByType application/x-ico "access plus 10 years"
    ExpiresByType application/ico "access plus 10 years"
```

```
    # Gzip all the things
    <IfModule mod_deflate.c>
        AddOutputFilterByType DEFLATE text/text text/html text/plain text/xml text/css
        AddOutputFilterByType DEFLATE application/x-javascript application/javascript image/x-icon
    </IfModule>

    # Seperate browser caching for gzip-encoded things
    <FilesMatch ".(js|css|xml|gz|html)$">
        Header append Vary: Accept-Encoding
    </FilesMatch>
</VirtualHost>


# Redirect other domains to www.thefec.org, preserving the URL path
<VirtualHost 72.249.12.147:80>
    ServerName thefec.org
    ServerAlias *.thefec.org
    ServerAlias thefec.skyhouseconsulting.com
    Redirect permanent / http://www.thefec.org/
</VirtualHost>
```

Then enable the site and restart apache:

```
$ sudo a2ensite thefec.org
$ sudo apache2ctl -k restart
```

The site should now be visible at http://www.thefec.org

4.8 Setup Cronjobs

We'll setup `cron` to run two scripts on a regular basis. One script will backup the database and uploads while the other will compress & optimize uploaded images.

First install the optimizing tools:

```
$ sudo apt-get install optipng jpegoptim
```

Now make directories for the scripts & backups to live in:

```
$ mkdir ~/bin ~/backups
```

Create a script called `backup_website.sh` in `~/bin` containing the following:

```
#!/usr/bin/env bash
# ~/bin/backup_website.sh
mv ~/backups/database.gz ~/backups/database.gz.2
pg_dump fec_website | gzip > ~/backups/database.gz

mv ~/backups/uploads.tar.gz ~/backups/uploads.tar.gz.2
tar -cpzf ~/backups/uploads.tar.gz ~/website/fec/static/media/
```

This will keep 2 days worth of backups in `~/backups`.

Now create `optimize_website_images.sh` in `~/bin` containing the following:

```
#!/usr/bin/env bash
# ~/bin/optimize_website_images.sh
```

```
find ~/htdocs/static/media -type f -iname "*.png" -exec optipng -o7 {} \;
find ~/htdocs/static/media -type f -iname "*.jpeg" -o -iname "*.jpg" -exec jpegoptim -t --all-progres
```

Make sure to mark them as executable:

```
chmod +x ~/bin/backup_website.sh ~/bin/optimize_website_images.sh
```

We'll set the backup script to run daily & the image optimizing to run weekly. Edit the enabled cronjobs by running `crontab -e` (or something like `EDITOR=vim crontab -e`) and add the following lines:

```
# Backup Website Database & Uploads
@daily ~/bin/backup_website.sh > /dev/null 2>&1

# Optimize Images Uploaded to the Website
@weekly ~/bin/optimize_website_images.sh > /dev/null 2>&1
```

CHAPTER 5

API Documentation

The following pages describe the interface of any custom code that is available to contributors.

5.1 core package

This package contains files used throughout the application, or those that are too general to fit into any other package(like the template overrides).

5.1.1 fec.utils module

5.1.2 fec.templatetags.core_filters module

This module defines general template filters.

`fec.templatetags.core_filters.`**`get_first_by`**(*value*, *stop_char*)
Split the string by the `stop_char` and return the first item.

Parameters

- **`value`** (*string*) – The value passed to the filter.
- **`stop_char`** (*string*) – The character the split the `value` by.

Returns The resultant sub-string.

5.2 communities package

This package is responsible for the `Community` model, including the listing and detail views.

5.2.1 Models

5.2.2 Views

5.2.3 Template Tags

5.2.4 Admin Forms

5.3 documents package

This package is responsible for the `Document` model, including `DocumentCategories` and listing views and widgets.

5.3.1 Models

5.3.2 Views

5.3.3 Template Tags

5.4 functional_tests package

This package contains functional tests which test the application as a whole, running a live test server and automating user interaction in real browsers using `Selenium`.

5.4.1 functional_tests.admin_tests module

5.4.2 functional_tests.general_page_tests module

5.4.3 functional_tests.home_page_tests module

5.4.4 functional_tests.community_page_tests module

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

f

F

G

P