Troop 89 Medfield Website

Aug 20, 2019

Contents:

The troop89medfield.org is the official website of the Boy Scouts of America's Troop 89 Medfield, a member of the Mayflower Council.

The Troop 89 website is powered by Django, a Python web framework.

This site intended to be designed and maintained by the youth members of Troop 89.

CHAPTER 1

Installing the Troop 89 Website

Note: This guide assumes that you already have Python installed. At the time of this writing, the troop 89 website requires Python 3.6 or greater.

Getting a local copy of the troop 89 website running is a fairly simple process, though there are several steps. This guide is intended to be especially detailed so that it may serve as a partial reference for future webmasters in training.

1.1 Fetching the Source

The easiest way to get a copy of the troop 89 website source is by cloning the public github repository. This has the added benefit of integrating git into your development environment, which will be necessary if you intend on making changes to the website's source.

```
$ git clone git://github.com/blueschu/troop89medfield.org.git
$ cd troop89medfield.org
```

If you are planning on submitting contributions to the website via a pull request, you should begin by forking the repository and then cloning that repository instead.

1.2 Installing the Dependencies

You will need to create a virtual environment for Python libraries. If you are using Python3.6+ (which you should be), the tools needed to create virtual environments ship with the interpreter as the venv module.

Note: Third party packages for creating virtual environments also exist. Some popular options are the virtualenv and virtualenvwrapper packages, which preceded the standard venv module and provide additional convenience tools.

Both of these packages can be installed with pip, and can be freely used inplace of the venv module in the following instructions.

To install the site dependencies, execute the following:

```
$ python3 -m venv troop89_venv          # Create a virtual environment in the folder
↪'troop89_venv'
$ source troop89_venv/bin/active        # Active the virtual environment
$ pip install -r requirements/dev.txt   # Install the development dependencies
```

On Windows, run `venv\Scripts\activate.bat` in place of `source troop89_venv/bin/active`.

1.3 Compiling the Stylesheets

The stylesheets for our website are written in sass, an extension language to css. You can either install sass system wide, use a feature in your IDE (if one is offered), or you can use the Python libsass package. Regardless of what method you use, you want to compile all the files in `assets/scss/*.scss` that don't begin with an underscore to `assets/css/`.

If you have already installed the site dependencies (*Installing the Dependencies*), you will have the libsass package in your virtual environment, which provides the `sassc` utility. With this program on your path (which will be the case if you have activated the virtual environment), you can run the following script to compile the stylesheets.

```
$ ./bin/build-scss.sh
```

1.4 Adding the Configuration File

You will need to create a file called `.secrets.json` in the root directory of the project sources. This is a config file that stores sensitive information such as database credentials and encryption keys.

An example configuration is located in the `demo.secrets.json` file, which you can temporarily copy to get the site running:

```
$ cp demo.secrets.json .secrets.json
```

This file includes the credentials for an SQLite database, which is a single-file based relational database that ships with Python. Before beginning significant development, you should install and configure a PostreSQL database (for which the site is designed) and substitute its credentials into `.secrets.json`. See *Database Configuration* for more details.

1.5 Initializing the Database

Assuming all configurations are good, you should only need to run

```
$ ./manage.py migrate
```

This will create the necessary tables and relations, but will not populate the database with data.

1.6 Populating the Database

The Troop 89 website ships with some default data to populate the database. This data is provided using Django fixtures, which are contained in the `fixtures` directory.

To load the default hostnames for the `django.contrib.sites` app, run

```
$ ./mange.py loaddata ./fixtures/sites.json
```

To load the demo flatpages, run

```
$ ./mange.py loaddata ./fixtures/demo_flatpages.json
```

1.7 Creating the Django Superuser

Run the following command

```
$ ./manage.py createsuperuser
```

and follow the prompts. This will create a user instance in the database that has all possible site privileges. You will need this to access the site admin.

1.8 Collecting the Static Media

Simply run

```
$ ./manage.py collectstatic
```

This will collect the static files and media from across the project into a single directory (`./static/`) so that they can be served by the web server. See the django staticfiles docs for more information

1.9 Updating Local Hostnames (Optional)

If you would like to use a host name (e.g. troop89.localhost) in place of a numeric IP (e.g. 127.0.0.1) when accessing the development site, you will want to update your machines hostname configuration. For Unix system (MacOS, Linux, etc), add the following entry to your `/etc/hosts` file:

```
127.0.0.1 troop89.localhost
```

Note that the `.localhost` TLD is reserved for loop back addresses of this sort. In fact, some browsers will treat `.localhost` domains as loop back addresses even without a DNS configuration or modified `/etc/hosts` file.

1.10 Running the Server

Warning: The following instructions are for development only. For production, a fully fledged HTTP server such as Apache or Nginx should be used in place of the lightweight server that ships with Django. See the Django runserver docs for more information.

To run the testing server, simple run

```
$ export DJANGO_SETTINGS_MODULE=troop89.settings.dev  # Use the development settings.␣
↪Run once per session.
$ ./mange.py runserver
```

If you updated your hosts files to include a local hostname, you can run the following instead

```
$ ./manage.py runserver troop89.localhost
```

Do note that by default, the production setting will be used (as defined in `troop89/wsgi.py`). To run the development flavor, set the environment variables `DJANGO_SETTINGS_MODULE` to `troop89.settings.dev`. This can be done by modifying your `~/.bashrc` file (to set it every time you begin a new bash session), by running `export DJANGO_SETTINGS_MODULE=troop89.settings.dev` in your terminal (as in the commands above), or by preceding the run server command itself with `DJANGO_SETTINGS_MODULE=troop89.settings.dev`.

CHAPTER 2

Running the Tests

The unit tests for the troop 89 website are pretty sparse at the moment. Contributions are always welcome!

Unit tests can be run via Django's test runner

```
$ ./manage.py test
```

To speed-up the tests by running them in parallel, you can pass the `--parallel` flag. To preserve the testing database after the tests run, you pass the `--keepdb` flag.

For more information, see the Django testing docs.

CHAPTER 3

Contributing to the Troop 89 Website

3.1 How Can I Contribute?

Members of Troop 89 are welcome and encouraged to contribute to the website in any way that they can. Common ways to contribute include *Reporting a Bug* that you found or *Requesting a Feature* to be implemented by the webmasters.

All members of Troop 89, particular those holding leadership positions such as Historian or Scribe, are welcome to submit content to be published on our website. If you are interested contributing in this way, please contact our webmasters over email or during a weekly meeting.

Scout who are interested in exploring the fields of web design, computer science, graphic design, or any of their relatives are encouraged to help with the maintenance of the troop's website. Possible ways to contribute include:

- Improving the presentation of our website by modifying the stylesheets
- Creating artwork or media to be posted to our website
- Developing new features for the website with the Django framework

3.2 Reporting a Bug

Bugs are tracked with GitHub issues. Before reporting a bug, be sure to check if it has already been reported by searching the existing issues on GitHub. If no report exists, go ahead a submit a new issue.

When submitting a report, be sure to include sufficient details for our webmasters to replicate the problem:

- Use a clear and descriptive title for the issue to identify the problem.
- Describe the exact steps to reproduce the problem with as many details as possible.
- Explain the behavior you expected to see instead and why.

When reporting a bug that affects the visual appearance of the Troop 89 website, please also provide details about how you are accessing the website. Specifically, please note what browser you are using along with any details about your system that may be relevant to the issue, such as Ad-blocking software.

You may also consider submitting a patch for the bug by *Making a Pull Request*.

3.3 Requesting a Feature

Enhancements requests are also tracked with GitHub issues. As with bug reports, please provide a clear and descriptive title that identifies the request. Be sure to explain the desired behavior of the new feature and how it would benefit the Troop 89 website.

3.4 Making a Pull Request

Contributions to the Troop 89 website should be submitted as a pull request on GitHub. The basic workflow for contributing a change is as follows.

1. Fork the Troop 89 Website repository.
2. Clone the fork repository to your machine.
3. Commit changes to a new feature branch.
4. Push your changes to the fork repository.
5. Submit a pull request to merge your work into the Troop 89 website.

This repository uses branching conventions that are adapted from A successful Git branching model:

- New features branches should be named `feature/your-feature-description` and should be based off of the `development` branch
- "Hot fixes" should be named `hotfix/v{SEMVER}` where `{SEMVER}` is the semantic version of the latest release with the patch version incremented. Hotfix branches should be based off of the `master` branch.

Commit messages should adhere to these general guidelines.

All Python code should adhere to the standard style guide for Python code (PEP 8).

CHAPTER 4

How to Maintain the Troop 89 Website

This document outlines how to maintain the Troop 89 website without any coding or web design experience. If you are interested in contributing to the website's source code, please see *Contributing to the Troop 89 Website*.

4.1 Accessing the Admin site

Link: The Troop 89 admin site.

Most management of the Troop 89 website can be accomplished within the customized admin site. If you are already logged in to the Troop 89 and have been granted access to the admin site, you should be able to access it by the link listed above. If you unable to reach the admin site and believe that you should have access, contact a webmaster.

4.2 Posting Announcements

From the admin site, navigate to Troop Announcement | Announcements | Add Announcement.

In the announcement creation form, provide a title, a date of publication, and some content for the announcement. Currently, announcements can be written in either plain text or Markdown. For a brief tutorial on how to stylize writing with Markdown, see GitHub's Mastering Markdown guide. In the future, a rich WYSIWYG editor may be provided for announcement creation, such as the one available for flatpage creation.

Advanced

At the bottom of the announcement creation form, you will find a collapsed fieldset title "Advanced". Expanding this fieldset will give you access to two additional fields: the user, and the slug.

The user field determines the person who is *displayed* as the author of a post. Note that this is separated from the system that records actions in the admin interface: the users responsible for creating and editing an announcement will be visible in the changelog.

The slug field determine the url that the announcement can be accessed with. By default, this field is generated from the the announcement's title, but you may wish to edit it in some circumstances to create a more expressive url.

After saving your announcement, you can view it on the live site by either navigating to the homepage or clicking the "View on Site" buttom in editing form.

4.3 Creating events

From the admin site, navigate to the Events | Events | Add Event.

In the announcement creation form, provide a title, type, description, and start and end time for the event. Currently, event descriptions can be written in either plain text or Markdown. For a brief tutorial on how to stylize writing with Markdown, see GitHub's Mastering Markdown guide. In the future, a rich WYSIWYG editor may be provided for announcement creation, such as the one available for flatpage creation.

For information on how to create a report for an event, see *Creating an Event Report*.

Advanced

At the bottom of the event creation form, you will find a collapsed fieldset title "Advanced". Expanding this fieldset will give you access to one additional fields: the slug.

The slug field determine the url that the event can be accessed with. By default, this field is generated from the the event's title, but you may wish to edit it in some circumstances to create a more expressive url.

4.4 Creating and editing static pages

The management of static or "flat" pages is handled by our custom flatpage app.

Navigate to the Flat page listing in the admin to view the page currently available on the site. From there, you may search for a page by title in the search box, filter what pages are shown by their parent page, or sort the pages by url or title.

To create a new flat page, click the "Add Flat Page" button in the top right corner of the page. In the flat page creation form, provide a URL, title, and rich body for the new page.

Take particular care when picking the URL for the new page. The URL that you provide will be used to determine where your new page should be displayed on the site. For example, a page with the URL `/about/squirrels` will be listed as a subpage on the `/about/` page and will visible as a related page on the `/about/chipmunks` page.

Advanced

At the bottom of the flat page creation form, you will find a collapsed fieldset title "Advanced options". Expanding this fieldset will give you access to two additional fields: a "registration required" checkbox, and a template name.

If selected, the "registration required" flag will restrict access to the flat page to users who are logged in to the website with a Troop 89 account. The page will not be listed for users who are not logged in to the site.

The "template name" field specifies the Django template from source control to use to render the page. This field defaults to `flatpages/default.html`. You should only edit this field if you need to heavily customize the rendering of your flatpage, such as loading a custom template tags or querying additional data from the database.

4.4.1 Creating an Event Report

Event Reports are a specific type of flat page that can be automatically generated from an Event.

To create a new Event Report, navigate to the event you intend to describe in the calendar. On the event's detail page, you will find a large button underneath the navigation toolbar that says "Create an Event Report". If you cannot see this button, you likely do not have permission to post new flatpages.

After clicking the "Create an Event Report" button, you will be redirected to a form that is prepopulated with information to create a flatpage from the event's details. From here, write the content of your report for the event in the space provided.

Be sure to save your report before closing the page.

4.4.2 Updating the newsletter archive

The newsletter archive can be edited in the same way as any other flat page. To view all of the current newsletter archive, navigate to the Flat page app in the admin and select `/records/newsletters/` under the parent page filter.

To update the archive, first upload the most recent newsletter and its supplementary document to the Troop 89 Google Drive. You may direct any questions regarding the upload process to a webmaster or past troop historian.

Once you have uploaded the relevant files, create a section in the newsletter year archive in the following format:

> {MONTH} Newsletter & Trip Information
>
> {MONTH} {YEAR} Newsletter
>
> {MONTH} {YEAR} Dates
>
> Supplementary Documents
>
> - Document-1
> - Document-2
> - ...

For each document listed above, highlight the document's name and then type `CTRL+K` to insert a link. In the window that pops up, enter the public URL for the document that you uploaded the the Google Drive. This can be found by right-clicking on the document in the Drive folder, clicking "Share", and the copying the shareable link shown.

CHAPTER 5

Notes to Webmasters

5.1 Repository Structure

The notable parts of the Troop 89 website repository are listed below:

```
$ tree -a -L 2 --dirsfirst
.
├── assets
│   ├── img
│   ├── scss
│   └── robots.txt
├── bin
├── docs
├── fixtures
├── requirements
│   ├── base.txt
│   ├── dev.txt
│   └── prod.txt
├── static
├── templates
│   ├── admin
│   └── includes
├── troop89
│   ├── announcements
│   ├── auth
│   ├── date_range
│   ├── events
│   ├── flatpages
│   ├── json_ld
│   ├── settings
│   ├── trooporg
│   ├── __init__.py
│   ├── urls.py
│   └── wsgi.py
```

(continues on next page)

(continued from previous page)

```
├── CONTRIBUTING.rst
├── demo.secrets.json
├── LICENSE
├── manage.py
├── README.rst
└── requirements.txt
├── .coveragerc
├── .secrets.json
├── .secrets.json.travis
├── .travis.yml
```

Briefly, the purpose of each file and directory is as follows:

- `assets`: A directory containing the site-wide static files
  - `scss`: The site's Sass stylesheets, which are describe to overall appearance of the site. These files are compiled to CSS before being served by the production server.
  - `img`: The static media and graphics used by the website.
  - `robots.txt`: A file that contains instructions for web crawlers, such as those used by Google and other search engines.
- `bin`: A directory containing executable scripts to help manage the site.
- `docs`: The root directory for the site's Sphinx documentation.
- `fixture`: A directory containing data fixtures for the site's Django models.
- `requirements`: A directory containing the site's Python package dependencies, which can be installed using `pip`.
  - `base.txt`: Fundamental package requirements for both production and development
  - `prod.txt`: Production-only package requirements
  - `dev.txt`: Development-only package requirements, e.g. debug tools, test coverage
- `static`: The root directory for servable static media. The `assets` directory and all `*/static` directories are copied here to be served by the production server.
- `templates`: The root directory for site-wide Django templates, such as the homepage and the error pages.
- `troop89`: A Python package containing the site's Django apps and configuration files
  - `announcements`: A Django app for troop announcements.
  - `auth`: A Django app for custom user authentication.
  - `date_range`: A helper app for creating models that can reason about ranges of dates.
  - `events`: A Django app for handling event creation and calendar display.
  - `flatpages`: A Django app for customized hierarchical flatpages.
  - `json_ld`: A helper app for rendering json-ld formatted structured data.
  - `settings`: A Python module for site settings.
  - `trooporg`: A Django app for troop organization (patrols, election terms, positions, etc).
  - `__init__.py`: The Python package file.
  - `urls.py`: The root url configuration.
  - `wsgi.py`: The WSGI application entry-point.

- `CONTRIBUTING.rst` and `README.rst`: Files containing documentation for the site's repository.
- `demon.secrets.json`: A configuration file for jump-starting a new site instance for development.
- `LICENSE`: The project license file (MPL-2.0).
- `manage.py`: A Python script provided by Django for managing the site's Django apps.
- `requirements.txt`: A requirements file that references the production package dependencies. It is equivalent to `requirements/prod.txt`.
- `.coveragerc`: A configuration file for the `coverage` Python package, which is used generate test coverage data.
- `.travis.yml` and `.secrets.json.travis`: Configuration files for Travis-CI continuous integration.
- `.secrets.json`: A configuration file for sensitive data, such as database passwords and encryption keys. This file is not kept in version control.

5.2 External Services

5.2.1 Travis CI

The Troop 89 website uses Travis CI for continuous integration. Documentation for Travis is available online, though you will likley never need to worry about changing the configuration.

All configuration for Travis CI is contained in the `.travis.yml` file.

5.2.2 Coveralls

The Troop 89 website uses Coveralls.io to display its code coverage data. Code coverage analysis is performed by the Python coverage package. If you have installed the development dependencies, you can generate a manual coverage report by running the following commands

```
$ coverage run manage.py test
$ coverage report
```

Documentation for `coverage` is available on coverage Read the Docs.

Coverage data is submitted to Coveralls.io by a Travis CI job phase. This step is handled by the coveralls-python package. Note that the coveralls API key is stored by Travis CI in an environment variable.

5.2.3 Uptime Robot

Uptime Robot periodically monitors the Troop 89 Website for server issues. Uptime statistics are available on the Troop 89 Website Public Status page.

5.2.4 Read the Docs

Documentation for the Troop 89 website is automatically built and published by ReadTheDocs.org. If you are not already, you can browse this documentation online through Read the Docs. Otherwise, you can manually build the documentation using Sphinx by running the following commands.

```
$ cd docs
$ make html
```

The documentation will then be available in `_build/html`. Open the `index.html` file in a browser to begin browsing the documentation.

CHAPTER 6

Deployment Considerations

6.1 Django Settings Module

Django uses the environment variable `DJANGO_SETTINGS_MODULE` to determine which Python module to import as the Django settings. Per the Django settings docs:

> When you use Django, you have to tell it which settings you're using. Do this by using an environment variable, `DJANGO_SETTINGS_MODULE`.
>
> The value of `DJANGO_SETTINGS_MODULE` should be in Python path syntax, e.g. `mysite.settings`. Note that the settings module should be on the Python import search path.
>
> **The django-admin utility**
>
> When using `django-admin`, you can either set the environment variable once, or explicitly pass in the settings module each time you run the utility.
>
> Example (Unix Bash shell):
>
> ```
> $ export DJANGO_SETTINGS_MODULE=mysite.settings
> $ django-admin runserver
> ```
>
> Example (Windows shell):
>
> ```
> > set DJANGO_SETTINGS_MODULE=mysite.settings
> > django-admin runserver
> ```
>
> Use the `--settings` command-line argument to specify the settings manually:
>
> ```
> $ django-admin runserver --settings=mysite.settings
> ```
>
> —Django settings docs

6.1.1 Settings in Production

In most cases, you do not have to worry about explicitly setting the Django settings module on the production server. This is because most entry points for interacting with Django (namely, `manage.py` and `troop89/wsgi.py`) will default to using `troop89.settings.prod`.

Note that if you use `django-admin` in place of `manage.py` when executing Django commands, you will have to explicitly define the settings module with `DJANGO_SETTIGNS_MODULE` or the `--settings` flag, as explained above.

6.1.2 Settings in Development

For development, you'll want to use the `troop89.settings.dev` setting module. This module adds some helpful development tools such as the django debug toolbar and removes some access constraints such as forced redirects to HTTPS.

This can be explicitly set by setting the `DJANGO_SETTINGS_MODULE` variable, or by passing the `--settings` flag to `manage.py` or `django-admin`, as detailed above.

6.2 Initializing the `sites` app

The Troop 89 website makes use of the Django sites framework. In order for the website to function, a `Site` model with an appropriate domain name needs to be added to the database.

Since the domain name that the Troop 89 websites operates behind will vary between instances, this model is not created by a database migration.

For development, a default model is provided in a fixture. See *Populating the Database* for details on how to load it.

For production, you must manually create the `Site` model. This can be accomplished in three ways:

1. **Load a fixture**. Created a file (say `prod_site.json`) with the following contents

   ```
   [{"model": "sites.site", "pk": 1, "fields": {"domain": "YOUR_DOMAIN_NAME", "name
   ↪": "Troop 89 Website"}}]
   ```

   where `YOUR_DOMAIN_NAME` is the domain for the production server. Then, execute the following command:

   ```
   ./manage.py loaddata ./prod_site.json
   ```

2. **Use the Djano Admin**. If your Django instance is already running, you can navigate to `YOUR_DOMAIN_NAME/admin/sites/site/1/change/` to update the default site model with the correct domain name.

3. **Use the Django shell**. Start a Django shell session and enter the following:

   ```
   >>>  from django.contrib.sites.models import Site
   >>> site = Site(pk=1, domain='YOUR_DOMAIN_NAME', name='Troop 89 Website')
   >>> site.save()
   ```

   Note that your should not use `Site.objects.create()`, since you want to override the default site rather than create a new one.

6.3 Database Configuration

The Troop 89 website is designed and tested with a PostgreSQL database server. It is highly recommended that you continue to use a PostgreSQL database in production to ensure that no compatibility errors occur. At the time of this writing, Django requires PostregreSQL 9.4 or higher. See the Django database installation docs for further details on how to run Django with a PostgreSQL backend.

6.4 Redirecting Traffic to HTTPS

The Troop 89 website implements many web security standards to ensure the safety its users' data. Notably, the Troop 89 website is configured for HTTPS Strict-Transport-Security, which mandates that browsers only access the site over an encrypted connection.

To ensure compatibility with HSTS standards, the production server should always redirect HTTP traffic to HTTPS. How this is accomplished will vary between web servers and hosts.

If you are running the Troop 89 website on a Apache server, Webfaction recommends directing all HTTP traffic to a site that has an `.htaccess` file with the following rules:

```
RewriteEngine On
RewriteCond %{HTTP:X-Forwarded-SSL} !on
RewriteCond %{REQUEST_URI} !^/(.well-known)(/|$)
RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI} [R=301,L]
```

Note: The `troop89.settings.prod` setting module defines the `SECURE_SSL_REDIRECT` option for Django's SecurityMiddleware. When this option is set, Django will emit a permanent redirect to HTTPS whenever it receives a request over HTTP. However, it is recommended that this redirect be performed by the webserver itself instead of Django. Performing redirects with the webserver will yield better performance and will reduce the risk of misconfiguration in the future.
