Pinax Symposion Documentation

Release 0.5dev

Eldarion Team

February 28, 2014

Contents

The PyCon web site is an open-source Django project for the PyCon conference, based on Pinax Symposion.

It came out of development done by Eldarion for DjangoCon US and US PyCon but has been independently used for a number of other conferences.

We are in the process of cleaning things up and making them more generic.

The project homepage is http://github.com/caktus/pycon.

See `README` for the installation instructions.

Apps:

CHAPTER 1

Conference App

The overall conference settings are managed via the `conference` app.

Conferences and their sections are added and configured via the Django admin.

1.1 Models

Each conference needs an instance of a `Conference` model. In most cases you will only need one of these but Symposion does support multiple conferences sharing a database. Similar to the Django Sites framework, the conference your project is for is selected by the `CONFERENCE_ID` setting which defaults to `1` but can be changed to the pk of another conference if you have more than one.

The conference model has an optional `start_date` and `end_date` indicating when the conference will run. These are optional so you can begin to configure your conference even if you don't know the exact dates.

The conference model also has a `timezone` field which you should set to the timezone your conference will be in.

There is also a `Section` model. This is useful if your conference has different parts to it that run of different days with a different management, review or scheduling process. Example of distinct sections might be "Tutorials", "Talks", "Workshops", "Sprints", "Expo". Many aspects of Symposion can be configured on a per-section basis.

Each section has an optional `start_date` and `end_date` similar to the overall conference.

1.2 Proposals

Create different kinds of proposals, e.g. *talk* or *tutorial*, by creating ProposalKind objects. You'll also need to create a Form in the code for that kind of proposal, and update the setting `PROPOSAL_FORMS` with the ProposalKind's `slug` as key, and the full package path to the form to use as value. For example:

```
PROPOSAL_FORMS = {
    "tutorial": "pycon.forms.PyConTutorialProposalForm",
    "talk": "pycon.forms.PyConTalkProposalForm",
    "poster": "pycon.forms.PyConPosterProposalForm",
}
```

To allow submitting proposals for a particular Section of the conference, create a ProposalSection. The site will allow submitting proposals for that Section between the ProposalSection's `start` and `end`, unless `closed` has been set.

1.3 Helper Functions

A `conference.models.current_conference()` function exists to retrieve the `Conference` selected by `CONFERENCE_ID`.

CHAPTER 2

Sponsorship App

Sponsorship is managed via the `sponsorship` app.

Sponsorship levels and sponsors are added via the Django admin.

2.1 Models

Each sponsor level has a `name` (e.g. "Gold", "Silver") and an `order` field which is an integer that is used to sort levels (lowest first). Each level also has a `description` which is not currently exposed anywhere but can be used for private annotation.

Each sponsor has a `name`, `external_url` (i.e. link to the sponsor's website), `contact_name` and `contact_email`, `logo`, and `level`.

A sponsor may also have a private `annotation` that can be used by organizers to take notes about the sponsor.

A sponsor will not appear on the site until the `active` flag is set true.

2.2 Template Snippets

The easiest way to include sponsor logos, grouped by level, is to either:

```
{% include "sponsorship/_vertical_by_level.html" %}
```

or:

```
{% include "sponsorship/_horizontal_by_level.html" %}
```

You can get a wall of sponsors (without level designation) with:

```
{% include "sponsorship/_wall.html" %}
```

You can always tweak these templates or use them as the basis for your own. This is often all you'll need to do to display sponsors on your site.

If you want to display a specific sponsor logo you can use:

```
{% include "sponsorship/_sponsor_link.html" with sponsor=sponsor %}
```

or:

```
{% include "sponsorship/_sponsor_link.html" with sponsor=sponsor dimensions="100x100" %}
```

if you want different dimensions than the default 150 x 150.

2.3 Template Tags

If you want to retrieve the sponsors and traverse them yourself, you can use the provided template tags:

```
{% load sponsorship_tags %}

{% sponsors as all_sponsors %}
```

or:

```
{% load sponsorship_tags %}

{% sponsors "Gold" as gold_sponsors %}
```

if you want to just get a specific level.

You can get the levels with:

```
{% load sponsorship_tags %}

{% sponsor_levels as levels %}
```

and you can always iterate over those levels, calling `level.sponsors` to get the sponsors at that level.

CHAPTER 3

Registration

The homepage template utilizes a configurable item to display information about conference registration. Go to `/YEAR/admin/constance/config/` and set `REGISTRATION_STATUS` to either `soon`, `open`, or `closed`. If the value is an empty string or a value other than the three valid entries, the homepage template will not include any specific registration status information or link. If other valid states are required, the homepage template will have to be modified accordingly.

The registration link (to actually register for the conference) just goes to a page that uses an iframe to wrap the real registration site, provided by a vendor.

There are a couple of configuration items that need to be agreed with the vendor each year, a shared secret and a URL. These should then be configured using the admin in the `constance` app. Go to `/YEAR/admin/constance/config/` and set `CTE_SECRET` to this year's shared secret and `REGISTRATION_URL` to this year's URL.

When ready to open up registration, make sure the vendor is ready, then put a link on the front page that goes to the URL named "registration_login", e.g.:

```
<a href="{% url 'registration_login' %}">Register!</a>
```

3.1 Tutorial Registration Data

Once the Schedule has been set, and Tutorials are open for registration, a management command that consumed registration data from the registration provider can be placed on a cron job for periodic updates.

One must configure the URL of the external CSV report to be consumed. These should be configured using the admin in the `constance` app. Got to `/YEAR/admin/constance/config` and set `CTE_TUTORIAL_DATA_URL` to this year's URL.

Once this is set, running the command job will update the Tutorial registrants via consumed emails, as well as set the max attendees for the Tutorial:

```
python manage.py update_tutorial_registrants
```

CHAPTER 4

Financial Aid

4.1 Settings

Create a FINANCIAL_AID setting in Django settings. It should be a dictionary. Values can include:

email The email address that messages related to financial aid come from, and that users should email with questions. Defaults to `pycon-aid@python.org`.

To enable applications, use the admin to create new FinancialAidApplicationPeriod records with the desired start and end dates.

4.2 Reviewer email to applicants

Reviewers can select one or more applicants on the application list page and click "Send email". On the next page, they can enter a subject, pick a template, and click "Send". Each applicant selected will receive an email customized for them using the template chosen.

Templates for this function are created and edited in the admin, at e.g. `/2014/admin/finaid/financialaidemailtemplate/`.

Each template has a name, which is just used to identify the template here and on the mail sending page, and a body, which uses Django templating to render the body of each email.

In the template body, you have access to the usual Django template tags, and some variables that you can access:

- application - a `FinancialAidApplication` object. This gives access to a lot of useful information from the user's application that can be used in your email, e.g.:

```
Dear {{ application.user.get_full_name }},

{% if application.travel_grant_requested %}You requested a travel grant...{% endif %}
```

- review - a `FinancialAidReviewData` object. This gives access to the information from the review of the application. E.g.:

```
{% if review.hotel_amount %}You are being granted ${{ review.hotel_amount }}
toward your hotel stay.{% endif %}
```

You can test your template by sending yourself email messages.

The fields in the FinancialAidApplication and FinancialAidReviewData records are subject to change, but you can review their current definitions at https://github.com/caktus/pycon/blob/production/pycon/finaid/models.py

4.3 Templates

4.3.1 Editing applications

To create or edit an application, the app uses the `finaid/edit.html` template. The context provides a `form` variable containing the form. A default template is provided that is customized to work with the PyCon site and uses `js/finaid.js` to hide some of the fields unless some other inputs have been checked, but the view doesn't care; it just wants the form submitted.

4.3.2 Email notices

The text for many emails comes from templates whose paths start with "finaid".

Email template file names have this format:

finaid/{{ recipient }}/{{ event }}/[subject|body].txt

recipient can be:

- applicant
- reviewer

event can be:

- edited
- submitted
- message (a message was added to an application)

subject and body should be self-evident.

Subject templates should be a single line.

So for example, the templates used to notify a reviewer that the applicant has edited their application are `finaid/reviewer/edited/subject.txt` and `finaid/reviewer/edited/body.txt`.

CHAPTER 5

Deploying

Some notes on deploying.

5.1 Periodic tasks

Arrange to run this command every day or so to expunge the data from deleted accounts if more than 48 hours since they were deleted:

```
python manage.py expunge_deleted
```

CHAPTER 6

Translation

The PyCon site is set up for use in English and French, for the most part.

6.1 Which language is displayed

By default, the request headers control which language is displayed. A user can change their brower's settings to say what their preferred languages are, and if French comes before English, the site will use French when available. It'll fall back to English for text that isn't translated.

A language selector can optionally be displayed on the Dashboard page. This allows a user to temporarily override the displayed language for the current session. Whether the language selector is displayed is controlled by the django-constance setting SHOW_LANGUAGE_SELECTOR, which can be changed in the admin at /YEAR/admin/constance/config/.

6.2 Translating

For CMS pages, there are two body fields. The first is for English. The second is for French. You'll have to scroll down a ways to see it when editing a CMS page.

Text on most other pages, forms, etc is translatable using Django's internationalization support. To add or update translations, a developer would:

- set up a local development environment for PyCon according to the README
- make a new branch off the *develop* branch
- make sure you have Gnu gettext installed
- install fabric: `pip install fabric`
- run `fab make_messages` to update the .po files, in case any translatable text has changed
- edit `locale/fr/LC_MESSAGES/django.po`, filling in `msgstr` with the translated version of whatever text is in the `msgid` just above it.
- run `fab compile_messages` to update the .mo file with the new translations
- commit the updated .po and .mo files
- open a pull request against the main repo to get your updates included

If a non-developer is going to help with translation, a developer could do all the steps except editing the .po file, just sending the .po file to the translator for them to edit and send back.

Any text not translated in the translation files will be displayed as English.

CHAPTER 7

API

There's a very basic API.

7.1 Authentication

To use the API requires an authentication key. Admins can add records to the pycon.APIAuth table and then give the randomly generated key to a user. They can also set a record to disabled (or just delete it) to revoke access.

7.2 Proposal data

The proposal data methods allow associating an arbitrary blob of text (perhaps JSON) with a proposal, and retrieving it later.

7.3 IRC logs

The IRC logs methods allow associating IRC log lines with a proposal, and retrieving them later.

The API tracks timestamps to the microsecond (if the database supports it), but be warned that the Django admin will lose the microseconds if you edit a log line there.

CHAPTER 8

Indices and tables

- *genindex*
- *modindex*
- *search*