Django Name Documentation

Release 3.0.0

University of North Texas Libraries

Sep 08, 2020

Contents

CHAPTER 1

About

The Name App is a tool originally developed for documenting names by the UNT Libraries for use in its various digital library systems and collections. The app provides a consistent way of communicating the authorized version of a name and information about the name that is useful for reuse. The Name App generates a unique URL for each name that can be used to unambiguously refer to a person, organization, event, building or piece of software. In addition to an HTML page for each name there are a number of other formats available for each record including a MADS XML version and a simple JSON representation. A key feature of the Name App is the ability to link to other vocabularies such as the Virtual International Authority File (VIAF), the Library of Congress Name Authority File, or Wikipedia.

CHAPTER 2

Table of Contents

2.1 Installation

2.1.1 Requirements

- Django 1.11
- Postgres or MySQL
- Django Admin - `django.contrib.admin`
- Humanize - `django.contrib.humanize`

Note: Django Name is intended to be installed within a Django project. If you are unfamiliar with Django, check out the docs.

2.1.2 Installation

1. Install the package from PyPI.

```
$ pip install django-name
```

2. Add `name` to your `INSTALLED_APPS`. Be sure to add `django.contrib.admin` and `django.contrib.humanize` if they are not already present.

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.humanize',
    # ...
    'name',
)
```

3. Configure the context processors.

```
TEMPLATES = [
    {
        'BACKEND': '...',
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                # ...
                'django.contrib.auth.context_processors.auth',
                'django.template.context_processors.request',
                'name.context_processors.name'
            ],
            # ...
        },
    },
]
```

Note: The `request` context processor is required by the Name app. The built-in templates require access to request parameters. The `name` processor enables the filter component of the search action as well as some optional branding (see *Branding*).

4. Include the URLs.

```
from name import urls as name_urls

urlpatterns = [
    # ...
    url(r'^name/', include(name_urls))
]
```

5. Migrate the database.

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

6. **Optional**: Load the Identifier Type fixtures. See *Loading Fixtures*.

2.1.3 Loading Fixtures

Note: This is an optional installation step.

The app comes with a fixture of predefined Indentifier Types. Issue one of the following commands below install them.

```
$ ./manage.py loaddata --app name identifier_types
```

2.2 Configuration

The Name App provides a few configurable settings.

2.2.1 Settings

Branding

NAME_APP_TITLE

Default: `"Django Name"`

This is displayed in the navbar and throughout the templates.

NAME_ADMIN_EMAIL

Default: `None`

When set, this will display on the about page as a point of contact for adding name records to the app.

Feed

Note: All feed settings are optional.

For the feed to be valid according to the Atom specification, an `<author/>` element containing a `<name/>` element is required.

NAME_FEED_AUTHOR_NAME

Default: `"Django Name"`

The author's name for the Name feed.

NAME_FEED_AUTHOR_EMAIL

Default: `None`

The author's email for the Name feed.

NAME_FEED_AUTHOR_LINK

Default: `None`

The author's URI for the Name feed.

2.3 Models

- *Name*
- *Identifier Type*

2.3.1 Name

Name objects have a variety of configurable options.

Fields

`name` - The canonical form of the name.

`name_type` - One of *Personal*, *Organization*, *Software*, *Building*, or *Event*

`biography` - Markdown enabled biography of the entity that the Name record represents.

`begin` - The starting date for the Name record. This will be different for each Name Type, for instance, this field would be the birth date for a *Personal* name and the erected date for a *Building* name.

`end` - Similar to `begin`

`disambiguation` - Clarification to whom or what the record pertains.

Variants

Variants are additional ways that the Name can be displayed.

Options include:

- *Acronym*
- *Abbreviation*
- *Expansion*
- *Translation*
- *Other*

Identifiers

The Identifier contains a type, (see *Identifier Type*), and value which is often represented as a permalink. For instance, the link to the person's Twitter profile would be an Identifier.

Notes

Additional notes regarding the person or the Name record.

Notes can be any of the following type:

- *Biographical/Historical*
- *Deletion Information*
- *Nonpublic*
- *Source*
- *Other*

Locations

Locations are represented by a geographic coordinate, which enable some mapping features within the app when present. A Name's location may be either `current` or `former`, and a Name may only have one `current` location at any given time.

Misc Options

Name records are capable of being merged with other Name records. Once merged with another record, any attempts to retrieve information about the merged record will redirect users to the Name record the was the target of the merge.

2.3.2 Identifier Type

These are customizable types for the Name *Variants*.

Fields

`label` - How the Identifier should be displayed.

`icon_path` - Relative path to the icon.

`homepage` - URL to the homepage of the service or website.

There are 13 `Identifier Types` that are included in a fixture. Those types are

- Academia
- Facebook
- Google Scholar
- Homepage
- Linkedin
- LOC
- ORD ID
- ResearchGate
- Scopus
- Tumblr
- Twitter
- VIAF
- Wikipedia

See *Loading Fixtures* for installation information.

2.4 Development

2.4.1 Development Environment

To take advantage of the dev environment that is already configured, you need to have Docker(>= 1.3) and Docker Compose installed.

Install Docker

Install Docker Compose.

```
$ pip install docker-compose
```

Clone the repository.

```
$ git clone https://github.com/unt-libraries/django-name.git
$ cd django-name
```

Start the app and run the migrations.

```
# start the app
$ docker-compose up -d

# wait about 2 minutes to let the database finish initializing, then run the␣
↪migrations
$ docker-compose run --rm web ./manage.py migrate

# optional: add a superuser in order to login to the admin interface
$ docker-compose run --rm web ./manage.py createsuperuser
```

The code is in a volume that is shared between your workstation and the web container, which means any edits you make on your workstation will also be reflected in the Docker container. No need to rebuild the container to pick up changes in the code.

However, if the requirements files change, it is important that you rebuild the web container for those packages to be installed. This is something that could happen when switching between feature branches, or when pulling updates from the remote.

```
# stop the app
$ docker-compose stop

# remove the web container
$ docker-compose rm web

# rebuild the web container
$ docker-compose build web

# start the app
$ docker-compose up -d
```

2.4.2 Running the Tests

To run the tests via Tox, use this command.

```
$ docker-compose run --rm web tox
```

To run the tests only with the development environment.

```
$ docker-compose run --rm web ./runtests.py
```

Note: This is the same command that Tox issues inside each test environment it has defined.

CHAPTER 3

Acknowledgements

- Joey Liechty
- Damon Kelley
- Lauren Ko
- Mark Phillips
- Gio Gottardi
- Madhulika Bayyavarapu