arpegio Documentation

Release 0.3.4

Gildardo Adrian Maravilla Jacome

August 12, 2016

Contents

Arpegio is a set of apps that work well together and are easily extendable and customizable. Following the best practices in the Django world and website development Arpegio takes away from you all the boilerplate needed to make an awesome website/webapp.

You can plug all the apps you need for the next big project you're working on. There's no need to learn yet another layer of abstraction. Arpegio is vanilla Django with some helper functions. You have more tools without the need to learn new rules.

The template structure makes really simple to create custom themes. For those that say that Django is great for webapps but bad for webdesign will think twice about it now.

CHAPTER 1

Getting started

1.1 Installation

Arpegio can be installed using **pip**:

```
pip install [--user] arpegio
```

1.1.1 Dependencies

Arpegio works with the following packages:

- Python 2.7, 3.3+
- Django >= 1.8, <1.11
- Pillow >= 3.3.0
- pytz

This packages are installed when using **pip**.

1.1.2 Applications

All the applications provided with Arpegio are optional. The only exception is `core`. To install the applications just add them to the `INSTALLED_APPS` section of your project's settings.

```
INSTALLED_APPS = (
    ...
    'arpegio.core',
    #'arpegio.blog',
    #'arpegio.pages',
    #'arpegio.categories',
    #'arpegio.tags',
)
```

1.1.3 Context processors

Add the settings template context processor. This context processor passes information about the settings of the project to the templates. The settings defined with Arpegio are usefull for extending and customizing themes.

```
TEMPLATES = [
   {
      'BACKEND': 'django.template.backends.django.DjangoTemplates',
      'DIRS': [],
      'APP_DIRS': True,
      'OPTIONS': {
            'context_processors': [
               ...
               'arpegio.core.context_processors.settings',
            ],
      },
   },
]
```

1.1.4 URLs

The urlpatterns defined in Arpegio for testing purposes are the following:

```
urlpatterns = [
   url(r'^blog/', include('arpegio.blog.urls')),
   url(r'^pages/', include('arpegio.pages.urls')),
   url(r'^category/', include('arpegio.categories.urls')),
   url(r'^tags/', include('arpegio.tags.urls')),
]
```

The above urlpatterns are not defined in Arpegio because all the apps are optional. You can copy and paste the above code or prefix the urls used in your project individually.

Note: When using Django 1.8 you need to define the app name and namespace of the urls with `include('path.to.urls', namespace='app', app_name='app')`.

For example, to include the blog app you need to include it with `include('arpegio.blog.urls', namespace='blog', app_name='blog')`.

If you don't set a namespace the url resolution methods will break.

1.1.5 Syncing the database

Arpegio is in alpha state and doesn't include migrations. To sync the database you will need to run:

```
python manage.py makemigrations
python manage.py migrate
```

The migrations are going to be provided when reaching the beta state.

1.1.6 Arpegio's settings

Arpegio uses a custom system to manage settings. To use it you have to include the following dictionary in your settings file:

```
ARPEGIO_SETTINGS = {
   'CATEGORY': {
      'Variable': {
```

```
            'value': 'Some value'
            }
        }
    }
```

The settings are passed to templates using the context variable `arpegio` and can be used like `{{ arpegio.category.variable.value }}`. Note that the keys are lowercased when they are processed.

This dictionary structure allows passing custom settings to the templates in a flexible way. You can easily include metada with the variables and pass information like this:

```
ARPEGIO_SETTINGS = {
    'GENERAL': {
        'Site_name': {
            'value': 'My site'}
    },
    'SOCIAL_MEDIA': {
        'Github': {
            'value': 'http://www.github.com',
            'css-class': 'github-link'
        },
        'Twitter': {
            'value': 'http://www.twitter.com'
            'css-class': 'twitter-link',
        },
        'Facebook': {
            'value': 'http://www.facebook.com'
            'css-class': 'facebook-link',
        }
    }
}
```

The settings can also be defined in reusable apps. Just add a **settings.py** file in your app's folder and include the following code:

```
from arpegio.core.settings import site

settings = {'GENERAL': {'SITE_NAME':{'VALUE': 'App defined'}}}

site.register(settings)
```

Project settings take precedence over App settings. Using this workflow you can make reusable apps and override specific settings in your configuration file without touching the templates. This makes template inheritance even easier to work with and the site's configurations are keept in a central file that can be versioned and/or forked.

CHAPTER 2

Reference

2.1 Blog

The blog app contains the boilerplate code needed to create a basic blog

2.1.1 Models

class arpegio.blog.models.**Post**(**args*, ***kwargs*)
Post Model.

get_absolute_url()
Get the absolute url of a post

2.1.2 Managers

class arpegio.blog.managers.**PostManager**
Post manager

public()
Filter the queryset to obtain the public posts.

sticky()
Filter the queryset to obtain the sticky posts.

2.1.3 Views

class arpegio.blog.views.**PostList**(***kwargs*)
Post list view.

model
alias of Post

class arpegio.blog.views.**PostDetail**(***kwargs*)
Post detail view.

model
alias of Post

2.2 Pages

The pages app contains logic to display pages from the database.

2.2.1 Models

class arpegio.pages.models.**Page**(**args*, ***kwargs*)
Page model.

get_absolute_url()
Get the absolute url of a page

2.2.2 Views

class arpegio.pages.views.**PageDetail**(***kwargs*)
Page detail view.

model
alias of Page

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

a

A

B

G

M

P

S