django-fiber Documentation

Release 1.6.dev0

Ride The Pony

Jan 19, 2018

Contents

This is the documentation project for django-fiber.

CHAPTER 1

Contents

1.1 Usage

1.1.1 Usage instructions

Page tree

Fiber was designed to coexist with your existing Django apps. In addition to the URLs that you define in your urls.py files you can create a tree of URLs that Fiber uses to create menus. Each node of the tree is a Page, which:

- has a title
- lives at a specific URL
- references a template
- serve as a placeholder for Content items
- can optionally redirect to another URL
- has some more tricks up its sleeve we'll talk about later

So before you can use Fiber template tags in your templates, you first have to create the Page tree. A simple Page tree might look like this:

- mainmenu
  - Home
  - About us
    - Mission
    - Our people
  - News

The root node is a special Page, whose sole purpose is to group Pages into a menu. The title of the root node is used to reference the menu when writing out the menu in your templates. The other properties of the root node (like URL) are ignored by Fiber.

Note: You can create multiple root nodes to create multiple, independent menus.

Pages

The URL of a Page can be specified in 3 different ways:

- as an absolute URL, like this: /this/is/an/absolute-url/
- as a relative URL, like this: relative-url
- as a named URL, like this: "news_item_list"

Absolute URLs

Using absolute URLs, you can specify the full URL path to the Page. Absolute URLs should start (and preferably also end) with a slash. You can also specify links to external sites by providing the full URL (starting with http:// or https://).

Relative URLs

Relative URLs are like folders on your computer. The full absolute URL of the Page is determined by walking up the tree, while concatenating relative URLs until the root node is reached, or until an absolute URL is encountered. Relative URLs should not contain slashes.

Named URLs

Named URLs are looked up in the urls.py files of all registered apps. Currently only named URLs that don't have parameters are supported.

Templates

When you have created your Page tree, you can start using the Fiber template tags in your templates. At the beginning of your template(s), load the Fiber template tags:

```
{% load fiber_tags %}
```

Using the Fiber template tags, you can:

- write out content items, that either
  - have a specified name
  - are linked to a specific location on the current page
  - are linked to a specific location on another page
- write out valid XHTML menu structures
  - of pages below a named root page (this is the menu name),
  - limited to a minimum and maximum level (depth),

  - that mark the currently active page,
  - optionally expanding all descendants of the currently active page,
  - with all possible css hooks you could ever need

Content items

You can write out content items with the 'show_content' and 'show_page_content' template tags:

```
{% show_content "content_item_name" %}
{% show_page_content "block_name" %}
{% show_page_content other_page "block_name" %} other_page being a Page object
```

Examples

This shows content item named 'address':

```
{% show_content "address" %}
```

This shows content items that are linked to the location named 'content' on the current page:

```
{% show_page_content "content" %}
```

This shows content items that are linked to the location named 'content' on another page 'other_page':

```
{% show_page_content other_page "content" %}
```

Menus

You can write out menus with the 'show_menu' template tag:

```
{% show_menu "menu_name" min_level max_level ["all_descendants / all"] %}
```

The menu name refers to a top-level node in the page tree.

Examples

The examples below assume the pages are structured like this:

- mainmenu
  - Home
  - About us
    - Mission
    - Our people
  - News
  - Products
    - Product A
      - Testimonials

            - Downloads
            - Technical data sheet
            - User manual
        * Product B
            - Downloads
        * Product C
            - Downloads
    - Contact
        * Newsletter
        * Directions
- generalmenu
    - Disclaimer
    - Privacy statement

Main menu

Show first and second level pages, below the root page named 'mainmenu':

```
{% show_menu "mainmenu" 1 2 %}
```

When the user is currently visiting the 'Home' page, this will show (current pages are bold):

- **Home**
- About us
- News
- Products
- Contact

When the user is currently visiting the 'Products' page, this will show:

- Home
- About us
- News
- **Products**
    - Product A
    - Product B
    - Product C
- Contact

As you can see, the sub pages of the currently active 'Products' page are automatically expanded.

When the user is currently visiting the 'Product A' page, this will show:

- Home

- About us
- News
- **Products**
  - **Product A**
  - Product B
  - Product C
- Contact

The sub pages of the 'Product A' page are not shown, because they are outside of the specified minimum and maximum levels.

Sub menu

Show pages from level 3 to 5, below the root page named 'mainmenu', and also show all descendants of the currently active page:

```
{% show_menu "mainmenu" 3 5 "all_descendants" %}
```

When the user is currently visiting the 'Home' page, this will show an empty menu, since it cannot be determined what level 3 pages are currently active.

However, when the user is currently visiting the 'Product A' page, this will show:

- **Product A**
  - Testimonials
  - Downloads
    - Technical data sheet
    - User manual
- Product B
- Product C

Notice that all pages below the currently active 'Product A' page are expanded because of the 'all_descendants' parameter.

Sitemap

Show all pages, with all pages expanded:

```
{% show_menu "mainmenu" 1 999 "all" %}
{% show_menu "generalmenu" 1 999 "all" %}
```

1.2 Advanced usage

This document is used to gather more advanced usage examples.

1.2.1 Optional settings

These settings are optional (default values are shown):

```
FIBER_LOGIN_STRING = '@fiber'

FIBER_DEFAULT_TEMPLATE = 'base.html'
FIBER_TEMPLATE_CHOICES = []
FIBER_CONTENT_TEMPLATE_CHOICES = []

FIBER_EXCLUDE_URLS = []

FIBER_IMAGES_DIR = 'uploads/images'
FIBER_FILES_DIR = 'uploads/files'

FIBER_EDITOR = 'fiber.editor_definitions.ckeditor.EDITOR'

FIBER_PAGE_MANAGER = 'fiber.managers.PageManager'
FIBER_CONTENT_ITEM_MANAGER = 'fiber.managers.ContentItemManager'

FIBER_METADATA_PAGE_SCHEMA = {}
FIBER_METADATA_CONTENT_SCHEMA = {}

FIBER_AUTO_CREATE_CONTENT_ITEMS = False

COMPRESS = [the opposite of DEBUG]

API_RENDER_HTML = False  # If set to True, you must include 'djangorestframework' in␣
↪INSTALLED_APPS as well

FIBER_IMAGE_PREVIEW = True  # If set to False, you don't need 'easy_thumbnails' in␣
↪INSTALLED_APPS
FIBER_LIST_THUMBNAIL_OPTIONS = {'size': (111, 111)}
FIBER_DETAIL_THUMBNAIL_OPTIONS = {'size': (228, 228)}
```

1.2.2 Set or override fiber_page in a view

In this example, the news_item_detail view looks up the Page of the news_item_list by looking up its named URL. This way, you can reuse the content you have placed on the news_item_list Page for each news_item_detail Page.

```
def news_item_detail(request, news_item_slug):
    news_item = get_object_or_404(NewsItem, slug=news_item_slug)

    fiber_page = Page.objects.get(url__exact='"news_item_list"')

    t = loader.get_template('news_item_detail.html')
    c = RequestContext(request, {
        'fiber_page': fiber_page,
        'news_item': news_item
    })
    return HttpResponse(t.render(c))
```

1.2.3 Set or override fiber_page in a class based view

In this example, the NewsItemDetailView's context is enriched with fiber_page and fiber_current_pages.

```
from django.core.urlresolvers import reverse
from django.views.generic import DetailView
from fiber.views import FiberPageMixin


class NewsItemDetailView(FiberPageMixin, DetailView):

    def get_fiber_page_url(self):
        return reverse('news_item_list')
```

1.2.4 Templates

In this example 4 page-templates will be available in the front- and backend-admin:

```
FIBER_TEMPLATE_CHOICES = (
    ('', 'Default template'),
    ('tpl-home.html', 'Home template'),
    ('tpl-intro.html', 'Intro template'),
    ('tpl-with-sidebar.html', 'With sidebar template'),
)
```

The first choice '' will load the FIBER_DEFAULT_TEMPLATE, default this is 'base.html'

In this example 2 content-templates will be available in the front- and backend-admin:

```
FIBER_CONTENT_TEMPLATE_CHOICES = (
    ('', 'Default template'),
    ('special-content-template.html', 'Special template'),
)
```

The first choice '' will load the default content-template, this is 'fiber/content_item.html'

1.2.5 Metadata

In this example metadata (key-value pairs) for pages will be available in the backend-admin:

```
FIBER_METADATA_PAGE_SCHEMA = {
    'title': {
        'widget': 'select',
        'values': ['option1', 'option2', 'option3',],
    },
    'bgcolor': {
        'widget': 'combobox',
        'values': ['#ffffff', '#fff000', '#ff00cc'],
        'prefill_from_db': True,
    },
    'description': {
        'widget': 'textarea',
    },
}
```

The first key key is 'title'. Because it has widget 'select' you will have 3 fixed values to choose from.

The second key is 'bgcolor'. Because it has widget 'combobox' you will have 3 fixed values to choose from and the choice to add your own 'bgcolor'. By setting prefill_from_db to True, the custom values you have chosen will also appear in the selectbox of fixed values.

The third key is 'description'. Because it has widget 'textarea' you can enter the value in a big textarea field.

Available widgets are: select combobox textarea textfield (default widget)

Only the combobox can prefill from the database by setting prefill_from_db = True (default=False)

The same metadata schema is available for metadata for content:

```
FIBER_METADATA_CONTENT_SCHEMA
```

1.2.6 CKEditor config settings

Some default CKEditor config settings can be altered by creating a file called admin-extra.js, which should be placed in a folder structure like this:

```
appname/static/fiber/js/admin-extra.js
```

Make sure 'appname' is placed _before_ 'fiber' in settings.INSTALLED_APPS, otherwise the admin-extra.js file won't override the default admin-extra.js provided by Django Fiber.

The following config settings can be used in admin-extra.js to override default CKEditor behavior:

```
window.CKEDITOR_CONFIG_FORMAT_TAGS = 'p;h1;h2;h3;h4';
window.CKEDITOR_CONFIG_STYLES_SET = [
    { name: 'intro paragraph', element: 'p', attributes: { 'class': 'intro' } }
];
window.CKEDITOR_CONFIG_EXTRA_PLUGINS = 'fpagelink,ffilelink,fimagelink,fcustomlink,
↪funlink,fimage,table,tabletools';
window.CKEDITOR_CONFIG_REMOVE_PLUGINS = 'scayt,language,menubutton,forms,image,link';
window.CKEDITOR_CONFIG_ALLOWED_CONTENT = false;
window.CKEDITOR_CONFIG_EXTRA_ALLOWED_CONTENT = 'a[*]{*}(*);img[*]{*}(*);iframe[*];
↪object[*];param[*];embed[*]';
window.CKEDITOR_TOOLBAR_CAN_COLLAPSE = false;
window.CKEDITOR_CONFIG_MAX_WIDTH = 610;
window.CKEDITOR_BASE_FLOAT_Z_INDEX = 1100;
```

You can also override the entire CKEditor toolbar, by setting the variable:

```
window.CKEDITOR_CONFIG_TOOLBAR
```

To see how this works, check the fiber.ckeditor.js file in the Django Fiber source: https://github.com/django-fiber/django-fiber/blob/master/fiber/static/fiber/js/fiber.ckeditor.js

1.2.7 Custom permissions

Fiber provides a *fiber.permissions* module. The Permission class defined here can be overridden by writing a custom permission class and pointing *PERMISSION_CLASS* in your settings module to that class.

Here's an example module that implements object level permissions:

```
"""
    >>> from django.contrib.auth.models import User
    >>> u = User.objects.get(username='example-user')
    >>> from guardian.shortcuts import assign
    >>> from fiber.models import Page
    >>> p = Page.objects.get(title='A')
    >>> assign('change_page', u, p)
```

```
"""

from django.contrib.auth.models import User
from django.db.models import Q
from django.contrib.contenttypes.models import ContentType
from django.db.models.signals import pre_delete

from guardian.shortcuts import get_objects_for_user, get_perms, assign
from guardian.models import UserObjectPermission

from fiber.permissions import Permissions
from fiber.models import Image, File, Page, PageContentItem, ContentItem


PAGE_PERMISSIONS = ('change_page', 'delete_page')
CONTENTITEM_PERMISSIONS = ('change_contentitem', 'delete_contentitem')


def remove_obj_perms_connected_with_object(sender, instance, **kwargs):
    filters = Q(content_type=ContentType.objects.get_for_model(instance),
        object_pk=instance.pk)
    UserObjectPermission.objects.filter(filters).delete()


class CustomPermissions(Permissions):

    def __init__(self):
        """
        Since guardian does not delete permission-objects, when the objects that
        they point to are deleted, we must take care of deleting them our selves.
        See http://packages.python.org/django-guardian/userguide/caveats.html?
↪highlight=caveat
        """
        pre_delete.connect(remove_obj_perms_connected_with_object, sender=Image)
        pre_delete.connect(remove_obj_perms_connected_with_object, sender=File)
        pre_delete.connect(remove_obj_perms_connected_with_object, sender=Page)
        pre_delete.connect(remove_obj_perms_connected_with_object,␣
↪sender=PageContentItem)
        pre_delete.connect(remove_obj_perms_connected_with_object, sender=ContentItem)

    def filter_objects(self, user, qs):
        """
        Returns all objects that `user` is allowed to change, based on guardian␣
↪permissions.
        Returns all objects if user is superuser.
        """
        if user.is_superuser:
            return qs
        return qs.filter(id__in=get_objects_for_user(user, 'change_%s' % qs.model.__
↪name__.lower(), qs.model))

    def can_edit(self, user, obj):
        """
        Returns True if `user` is allowed to edit `obj` based on guardian permissions.
        """
        return 'change_%s' % obj.__class__.__name__.lower() in get_perms(user, obj)

    def can_move_page(self, user, page):
```

```
        """
        A user with change-permissions is allowed to move the page.
        A superuser always has all permissions as far as guardian is concerned.
        """
        return 'change_page' in get_perms(user, page)

    def object_created(self, user, obj):
        """
        Create 'change' permission to `obj` for `user`.
        """
        assign('change_%s' % obj.__class__.__name__.lower(), user, obj)

    def _filter_user_and_superuser(self, user, qs):
        """
        A user should see files and images owned by him and by the superuser.
        Files uploaded by other non-superusers should not be listed.

        Note - there should be only one superuser in the User model.
        """
        superuser = User.objects.get(is_superuser=True)

        qs = qs.filter(Q(id__in=get_objects_for_user(user, 'change_%s' % qs.model.__
↪name__.lower(), qs.model)) |
            Q(id__in=get_objects_for_user(superuser, 'change_%s' % qs.model.__name__.
↪lower(), qs.model)))
        return qs

    def filter_images(self, user, qs):
        return self._filter_user_and_superuser(user, qs)

    def filter_files(self, user, qs):
        return self._filter_user_and_superuser(user, qs)
```

1.2.8 Sitemap

The Sitemaps protocol allows a webmaster to inform search engines about URLs on a website that are available for crawling. Django comes with a high-level framework that makes generating sitemap XML files easy.

Install the sitemap application as per the instructions in the django documentation, then edit your project's `urls.py` and add a reference to Fiber's Sitemap class in order to included all the publicly-viewable Fiber pages:

```
...
from fiber.sitemaps import FiberSitemap
...
(r'^sitemap\.xml$', 'django.contrib.sitemaps.views.sitemap', {'fiber': FiberSitemap,
                                                               ... other sitemaps... })
...
```

1.3 Contributors

```
Bart Heesink <bheesink@leukeleu.nl>
Bram Simons <bsimons@leukeleu.nl>
Chi Shang Cheng <cscheng@leukeleu.nl>
Chris Clark <chris@untrod.com>
```

```
David Filipovic <david.filipovic@gmail.com>
Dennis Bunskoek <dbunskoek@leukeleu.nl>
Diederik van der Boor <vdboor@edoburu.nl>
Douwe van der Meij <vandermeij@gw20e.com>
Dzjon Hessing <dhessing@ridethepony.nl>
Jaap Roes <jroes@leukeleu.nl>
Jaco Bovenschen <jbovenschen@leukeleu.nl>
Luke Plant <L.Plant.98@cantab.net>
Maarten Draijer <maarten@madra.nl>
Marco Braak <mbraak@ridethepony.nl>
Marko Tibold <mtibold@leukeleu.nl>
Michael van de Waeter <mvandewaeter@leukeleu.nl>
Nick Badoux <nbadoux@leukeleu.nl>
Niels van Dijk <nvandijk@leukeleu.nl>
Ramon de Jezus <rdejezus@leukeleu.nl>
Richard Barran
Selwin Ong <selwin@ui.co.id>
Zenobius Jiricek <zenobius.jiricek@gmail.com>
```

1.4 Development

1.4.1 New releases

In order to make releasing a new version slightly easier, this project has been set up to use Zest-releaser, which will need to be installed on your local dev machine:

```
pip install zest.releaser[recommended]
```

The *[recommended]* adds a number of optional packages that make Zest-releaser more convenient.

To use Zest.releaser: rather than copy-and-paste the excellent instructions from their docs, here is a link to their documentation.

Note that the `prerelease` command has some custom hooks in order to:

1. Run all the unit tests before a release.
2. There is no 2!

Note: In order for Django-Fiber to be uploaded to PyPI, you will need write privileges.

1.4.2 Translations

Translation strings are managed through Transifex - here's the Django-Fiber project page.

The intention is to always have several Transifex maintainers (at present there are 2).

If you are listed as maintainer on Transifex, here's a checklist of common actions that you can perform:

Pull from Transifex

Get translated strings from Transifex and add them to your local copy of the project - the following assumes that you have already set up your connection to Transifex on your local computer:

```
cd <fiber folder of project>
tx pull -a --force
django-admin.py compilemessages
```

Push to Transifex

If any strings in the project have been modified, they need to be sent up to Transifex for translation:

```
cd <fiber folder of project>
django-admin.py makemessages --all
tx push --source --translations
```

1.5 app_settings

Custom permissions

fiber.app_settings.**DETAIL_THUMBNAIL_OPTIONS = {'size': (228, 228)}**
Point this class to your own Permission Class as declared in *fiber.permissions*.

1.6 Library

1.6.1 permissions

Module that provides a base Permission class. This class may be overridden by changing the *PERMISSION_CLASS* value in the settings module.

class fiber.permissions.**Permissions**
This class defines the methods that a Permission class should implement.

By default all permissions are granted to a staff user.

can_edit(*user*, *obj*)
Should return `True` if user is allowed to edit *obj*.

can_move_page(*user*, *page*)
Should return `True` if user is allowed to move page.

filter_files(*user*, *qs*)
Called by API while listing files.

filter_images(*user*, *qs*)
Called by API while listing images.

filter_objects(*user*, *qs*)
Should only return those objects whose *user* is allowed to edit. *qs* can consist of type *Page* or *ContentItem*.

is_fiber_editor(*user*)
Determines if the user is allowd to see the Fiber admin interface.

object_created(*user*, *obj*)
Called whenever a new instance has been created of one of Fiber's models by *user*.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index