django-templation Documentation

Release 0.1.0

QDQ media S.A.U.

April 25, 2014

Contents

Contents:

CHAPTER 1

django-templation

The easy way to allow designers edit templates and assets.

1.1 Documentation

The full documentation is at http://django-templation.rtfd.org.

1.2 Installation

Install django-templation:

```
pip install django-templation
```

1.3 Features

- Resource Access administration via Django admin.
- WebDAV access for designers to easily edit templates and static files from anywhere.
- Sandboxed templates: restrict the use of Django builtin template tags and filters

CHAPTER 2

Installation

At the command line:

```
$ easy_install django-templation
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv django-templation
$ pip install django-templation
```

CHAPTER 3

Tutorial

Welcome to Templation's tutorial. In this document you will integrate *django-templation* in a sample project. Our project will be a very simple hello world with the feature multiple themes.

3.1 Setting up the environment and project

First things first, we are going to create our development environment and the *hello world* project.

Create project directory

```
$ mkdir hello-templation
$ cd hello-templation
```

Create virtualenv

```
$ mkvirtualenv hello-templation
```

Create requirements.txt file:

```
# requirements.txt
django==1.6
django-templation
```

Install requirements

```
$ pip install -r requirements.txt
```

Create a new Django project

```
$ django-admin.py startproject hellotemplation
$ cd hellotemplation
$ django-admin.py startapp core
```

Edit `hello-templation/hellotemplation/core/models.py`

```
from django.db import models


class Theme(models.Model):
    name = models.CharField(max_length=50, unique=True)
```

Edit `hello-templation/hellotemplation/core/views.py`

```
from django.views.generic import TemplateView
from templation.views import ResourceStoreMixin
from .models import Theme

class Index(ResourceStoreMixin, TemplateView):
    template_name = "core/index.html"

    def get_templation_object(self, *args, **kwargs):
        try:
            return Theme.objects.get(name=kwargs.get('theme-name', ''))
        except Theme.DoesNotExist:
            return Theme.objects.first()
```

Edit `hello-templation/hellotemplation/hellotemplation/urls.py`

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
admin.autodiscover()

from templation.urls import templation_static
from core.views import Index

urlpatterns = patterns('',
    url(r'^$', Index.as_view(), name='index'),
    url(r'^admin/', include(admin.site.urls)),
) + templation_static()
```

Create index template (`hello-templation/hellotemplation/core/templates/core/index.html`)

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Hello world!</title>
    </head>
    <body>
        <p>Hello world!</p>
    </body>
</html>
```

Configure settings

```
...
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'templation'
    'core',  # Add your new app
)
...

TEMPLATE_LOADERS = (
    'templation.loaders.TemplationLoader',
    'django.template.loaders.filesystem.Loader',
    'django.template.loaders.app_directories.Loader'
```

```
)

MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'templation.middleware.TemplationMiddleware',
)

TEMPLATE_CONTEXT_PROCESSORS = (
    'django.contrib.auth.context_processors.auth',
    'django.core.context_processors.debug',
    'django.core.context_processors.i18n',
    'django.core.context_processors.media',
    'django.core.context_processors.static',
    'django.core.context_processors.tz',
    'django.contrib.messages.context_processors.messages',
    'templation.context_processor.templation_info'
)

# Django-templation settings
TEMPLATION_DAV_ROOT = os.path.join(BASE_DIR, '..', 'dav')  # Make sure you create this folder
TEMPLATION_DAV_STATIC_URL = '/static_templation/'
TEMPLATION_RESOURCE_MODEL = 'core.models.Theme'
```

Launch for the first time

```
$ python manage.py syncdb
$ python manage.py runserver
```

Go to http://127.0.0.1:8000 and you will see the *Hello world!*.

3.2 Setting up WebDAV shared resources

The first thing to do is to make sure you have created the root folder for the WebDAV service (the one defined in `TEMPLATION_DAV_ROOT`):

```
$ sudo mkdir <TEMPLATION_DAV_ROOT>
$ sudo chown youruser.yourgroup <TEMPLATION_DAV_ROOT>
```

Edit *wsgi.py* file in your Django project to activate WsgiDav middleware:

```
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "hellotemplation.settings")

from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()

from templation.middleware import WsgiDAVMiddleware
application = WsgiDAVMiddleware(application)
```

3.2.1 Add boilerplate template to settings

When a user is linked to a resource by a `ResourceAccess` object a default template is copied to its WebDAV folder if we define `TEMPLATION_BOILERPLATE_FOLDER` setting.

Create boilerplate files (in this example this folder is located at the same level of `TEMPLATION_DAV_ROOT`):

```
dav_boilerplate/
dav_boilerplate/templates
dav_boilerplate/templates/core
dav_boilerplate/templates/core/index.html
dav_boilerplate/static
dav_boilerplate/static/css
dav_boilerplate/static/css/main.css
dav_boilerplate/static/js
dav_boilerplate/static/js/main.js
```

Edit `settings.py`

```
...
TEMPLATION_BOILERPLATE_FOLDER = os.path.join(BASE_DIR, '..', 'dav_boilerplate')
...
```

3.2.2 Create some themes

```
$ python manage.py shell

>>> from core.models import Theme
>>> Theme.objects.create(name='simple')
<Theme: Theme object>
>>> Theme.objects.create(name='red')
<Theme: Theme object>
```

3.2.3 Create ResourceAccess

```
$ python manage.py shell

>>> from templation.settings import get_resource_access_model
>>> from django.contrib.auth import get_user_model
>>> from templation.models import ResourcePointer
>>> from core.models import Theme
>>> resource_pointer = ResourcePointer.objects.create(resource=Theme.objects.get(name='simple'))
>>> get_resource_access_model().objects.create(user=get_user_model().objects.get(username='admin'), r
<ResourceAccess: ResourceAccess object>
```

3.2.4 Accessing WebDAV folder

`http://127.0.0.1:8000/<TEMPLATION_PROVIDER_NAME>/<RESOURCE_PK>/`

http://127.0.0.1:8000/templation/1/

Note: When accessing the above URL you will be asked for the user credentials corresponding to the user linked in the `ResourceAccess` object. There are several more ways to access and change a WebDAV folder, more info on WsgiDAV docs.

3.3 Overriding a template

Now that the WebDAV environment is set up, the next step is to modify the template.

Edit `dav/1/templates/core/index.html`:

```
{% load static from templation_tags %}
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Hello world!</title>
        <link rel="stylesheet" href="{% static 'css/main.css' %}">
    </head>
    <body>
        <h1>Hello overriden world!</h1>
    </body>
</html>
```

Edit `dav/1/static/css/main.css`:

```
h1 {
    color:#333333;
    font-family:serif;
    text-shadow: 4px 4px 2px rgba(150, 150, 150, 1);
}
```

Go to http://127.0.0.1:8000 and you will see a fancier *Hello world!*.

> **Warning:** Not showing the fancy Hello World? Checkout *Visibility of custom templates* section.

CHAPTER 4

Usage

To use django-templation in a project

4.1 Django settings

4.1.1 Minimal Django configuration

```
INSTALLED_APPS = [
    'templation.builtins',
    "django.contrib.sessions",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.staticfiles",
    "django.contrib.sites",
    "templation",
]

TEMPLATE_LOADERS = (
    'templation.loaders.TemplationLoader',
    'django.template.loaders.filesystem.Loader',
    'django.template.loaders.app_directories.Loader'
)

MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'templation.middleware.TemplationMiddleware',
)

TEMPLATION_BOILERPLATE_FOLDER = '/path/to/boilerplate/folder/'
TEMPLATION_DAV_ROOT = '/path/to/webdav/folder/'
TEMPLATION_DAV_STATIC_URL = '/templationdav/'  # URL to bind templation statics
TEMPLATION_RESOURCE_MODEL = 'yourapp.models.MyResource'
```

4.1.2 Settings in detail

TEMPLATION_DAV_ROOT

Default value: `N/A`

Required: Yes

Defines the root path of *WebDAV* folder where designers will edit templates and static files.

TEMPLATION_DAV_STATIC_URL

Default value: `N/A`

Required: Yes

Defines the root url to access custom static files, it acts the same way as Django's `STATIC_URL`, but only for `django-templation`.

TEMPLATION_RESOURCE_MODEL

Default value: `N/A`

Required: Yes

The model that represents the *tenant*, templates will be bound to it.

TEMPLATION_PROVIDER_NAME

Default value: `'templation'`

Required: No

Provider name for WebDAV server. It also acts as the root url to access WebDAV folders.

TEMPLATION_BOILERPLATE_INITIALIZER

Default value: `'templation.models.copy_boilerplate_folder'`

Required: No

Path to a Python callable that will be executed when resource access object is created for the first time.

TEMPLATION_BOILERPLATE_FOLDER

Default value: `None`

Required: No

Path to the folder containing the initial data for WebDAV shared folders.

TEMPLATION_DUMP_EXCEPTION

Default value: `('TemplateDoesNotExist', 'TemplateSyntaxError')`

Required: No

Iterable of exception names that will be shown to the designers.

TEMPLATION_SECRET_KEY

Default value: `SECRET_KEY`

Required: No

`SECRET_KEY` used to generate access tokens.

TEMPLATION_SANDBOX

Default value: `False`

Required: No

Activate sandbox environment for templates. Only whitelisted tags and filters will be available.

TEMPLATION_WHITELIST_TAGS

Default value: `DEFAULT_WHITELIST_TAGS`

Required: No

Safe template tags for sandbox.

TEMPLATION_WHITELIST_FILTERS

Default value: `DEFAULT_WHITELIST_FILTERS`

Required: No

Safe template filters for sandbox.

TEMPLATION_EXTRA_LIBRARIES

Default value: `DEFAULT_EXTRA_LIBRARIES`

Required: No

Preloaded tags and filters for sandbox.

TEMPLATION_DEBUG

Default value: `False`

Required: No

Activate templation's custom 500 error debug page.

DEFAULT_WHITELIST_TAGS

```
DEFAULT_WHITELIST_TAGS = [
    'comment', 'csrf_token', 'cycle', 'filter', 'firstof', 'for', 'if',
    'ifchanged', 'now', 'regroup', 'spaceless', 'templatetag', 'url',
    'widthratio', 'with', 'extends', 'include', 'block'
]
```

DEFAULT_WHITELIST_FILTERS

```
DEFAULT_WHITELIST_FILTERS = [
    'add', 'addslashes', 'capfirst', 'center', 'cut', 'date', 'default',
    'default_if_none', 'dictsort', 'dictsortreversed', 'divisibleby', 'escape',
    'escapejs', 'filesizeformat', 'first', 'fix_ampersands', 'floatformat',
    'force_escape', 'get_digit', 'iriencode', 'join', 'last', 'length', 'length_is',
    'linebreaks', 'linebreaksbr', 'linenumbers', 'ljust', 'lower', 'make_list',
    'phone2numeric', 'pluralize', 'pprint', 'random', 'removetags', 'rjust', 'safe',
    'safeseq', 'slice', 'slugify', 'stringformat', 'striptags', 'time', 'timesince',
    'timeuntil', 'title', 'truncatewords', 'truncatewords_html', 'unordered_list',
    'upper', 'urlencode', 'urlize', 'urlizetrunc', 'wordcount', 'wordwrap', 'yesno'
]
```

DEFAULT_EXTRA_LIBRARIES

```
DEFAULT_EXTRA_LIBRARIES = [
    'templation.templatetags.templation_tags',
]
```

4.2 Enable WebDAV in your Django project

django-templation uses WsgiDAV to expose WebDAV folders. To enable this functionality you must edit your *wsgi.py* file:

```
import os

# We defer to a DJANGO_SETTINGS_MODULE already in the environment. This breaks
# if running multiple sites in the same mod_wsgi process. To fix this, use
# mod_wsgi daemon mode with each site in its own daemon process, or use
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "yourproject.settings")

# This application object is used by any WSGI server configured to use this
# file. This includes Django's development server, if the WSGI_APPLICATION
# setting points here.
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()

# Apply WSGI middleware here.
# from helloworld.wsgi import HelloWorldApplication
# application = HelloWorldApplication(application)

from templation.middleware import WsgiDAVMiddleware
application = WsgiDAVMiddleware(application)
```

Required settings	Example value
TEMPLATION_DAV_ROOT	/var/www/dav/
TEMPLATION_PROVIDER_NAME	templation

4.3 Serving static content

TEMPLATION_DAV_STATIC_URL defines the URL which serves customized statics. You need to configure your web server (like NGINX) to serve this files properly

In this example TEMPLATION_DAV_STATIC_URL is set to /templationdav/:

```
server {
    listen 80;

    location ~ ^/templationdav/(\d+)/(.*)$ {
        alias /your/davroot/$1/static/$2;
    }

    location /static/ {
        alias /your/static/path/;
    }

    location / {
        include uwsgi_params;
        uwsgi_pass 127.0.0.1:3031;
        uwsgi_param    SCRIPT_NAME '';
    }
}
```

4.3.1 Static content in development mode

To serve templation's static content from development server (python manage.py runserver) it is necessary to add templation_static() to your url patterns in your urls.py:

```
from django.conf.urls import patterns, url, include
from django.contrib import admin
from templation.urls import templation_static  # Important line
from .views import *

admin.autodiscover()

urlpatterns = patterns(
    '',
    url(r'^admin/', include(admin.site.urls)),
    url(r'^index/$', index, name='index'),
) + templation_static()  # Important line
```

4.4 Customizations

4.4.1 Resource Model

The *Resource Model* can be any Django model.

4.4.2 Resource Access Model

Resource Access Model controls when 'development' templates and static files are shown. **Templation** comes with a default *Resource Access Model* but you can inherit from `AbstractResourceAccess` and make your custom one

```
from templation.models import AbstractResourceAccess


class CustomResourceAccess(AbstractResourceAccess):
    """ django-templation """
```

4.4.3 Restricting template tags and filters

You can set up a sandboxed environment for template designers restricting the use of builtin tags and filters and preloading the desired ones.

In django settings:

```
TEMPLATION_SANDBOX = True  # Enables the sandbox mode

# List of allowed tags
TEMPLATION_WHITELIST_TAGS = [
    'comment', 'csrf_token', 'cycle', 'filter', 'firstof', 'for', 'if',
    'ifchanged', 'now', 'regroup', 'spaceless', 'templatetag', 'url',
    'widthratio', 'with', 'extends', 'include', 'block'
]

# List of allowed filters
TEMPLATION_WHITELIST_FILTERS = [
    'add', 'addslashes', 'capfirst', 'center', 'cut', 'date', 'default',
    'default_if_none', 'dictsort', 'dictsortreversed', 'divisibleby', 'escape',
    'escapejs', 'filesizeformat', 'first', 'fix_ampersands', 'floatformat',
    'force_escape', 'get_digit', 'iriencode', 'join', 'last', 'length', 'length_is',
    'linebreaks', 'linebreaksbr', 'linenumbers', 'ljust', 'lower', 'make_list',
    'phone2numeric', 'pluralize', 'pprint', 'random', 'removetags', 'rjust', 'safe',
    'safeseq', 'slice', 'slugify', 'stringformat', 'striptags', 'time', 'timesince',
    'timeuntil', 'title', 'truncatewords', 'truncatewords_html', 'unordered_list',
    'upper', 'urlencode', 'urlize', 'urlizetrunc', 'wordcount', 'wordwrap', 'yesno'
]

# Preloaded tags
TEMPLATION_EXTRA_LIBRARIES = [
    'yourapp.templatetags.yourapp_tags',
]
```

4.4.4 Debug 500 errors for designers

Designers may be overwhelmed by django's default 500 error page in debug mode, so *django-templation* includes a custom 500 error view that shows debug information for the exceptions defined in `TEMPLATION_DUMP_EXCEPTION` setting.

To activate this functionality you have to add these lines to your `urls.py`

```
from django.conf.urls import *
handler500 = 'templation.views.server_error'
```

Required settings	Example value
TEMPLATION_DEBUG	True

4.5 Visibility of custom templates

The ResourceAccess (*RA*) object defines if a user can access a *WebDAV* folder associated with a object of class TEMPLATION_RESOURCE_MODEL.

ResourceAccess has two interesting properties:

- resource_pointer foreign key: Accesses the resource properties, where you have is_validated field, that indicates if the customized resources will be available for everyone.
- get_access_token() method: Returns an access token that allows everyone to see the customized version for this resource.

Note: You can also get the access token in the admin detail view of ResourceAccess object.

Table defining whether or not the customized template will be shown:

User type	No RA	RA (not validated)	RA (validated)	Access token
User with ResourceAccess	No	Yes	Yes	Yes
Others	No	No	Yes	Yes

4.6 Extra goodies

templation_tags.is_trusted_request is a template tags which tells whether a request comes from a trusted source. (an ip address listed in settings.INTERNAL_IPS or an active, staff user). This can be used to show further debugging information in the template being rendered.

django-templation comes with a custom context processor and a template tag which will help showing this additional information to the designer.

The context_processor.templation_info context processor pushes two new variables into the templates context. Together with the Django admin documentation generator, you can point designers to documentation which is relevant to the page they're working on.

Variable name	Example value
templation_view	app.views.ItemList
templation_template	item_list.html

templation_tags.get_model_info can additionally be used to link to models documentation in the Django admin documentation generator.

This is an example template block showcasing the integration that can be achieved:

```
{% load templation_tags %}

{% if is_trusted_request %}
<div>
    {% if object %}
        {% get_model_info object as model_info %}
    {% elif object_list %}
        {% get_model_info object_list as model_info %}
    {% endif %}
```

```
    <a href="{% url "django-admindocs-docroot" %}">Documentation</a> -
    <strong>Model:</strong>
        <a href="{% url 'django-admindocs-models-detail' app_label=model_info.app_label model_name=mo
    <strong>View:</strong>
        <a href="{% url 'django-admindocs-views-detail' templation_view %}">{{ templation_view }}</a>
    <strong>Template:</strong>
        {{ templation_template }}
</div>
{% endif %}
```

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/qdqmedia/django-templation/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

5.1.4 Write Documentation

django-templation could always use more documentation, whether as part of the official django-templation docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/qdqmedia/django-templation/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *django-templation* for local development.

1. Fork the *django-templation* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/django-templation.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv django-templation
   $ cd django-templation/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 templation tests
$ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/qdqmedia/django-templation/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_templation
```

CHAPTER 6

Credits

6.1 Development Lead

- QDQ media S.A.U. <tecnologia@qdqmedia.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.1.0 (2014-01-24)

- First release.