Template Documentation

Release 0.0.2

Nickolas Fox

Sep 27, 2017

Contents

Contents:

CHAPTER 1

Installation

Dependecies

List of whole dependencies split into backend and frontend ones. But before start you should install of check your list for this software:

requirements list

1. python 2.7, 3.3, 3.4
2. nodejs-0.10.x with npm (bower for js/css deps)
3. psycopg2 for PostgreSQL databases connection
4. python-mysql for MySQL/MariaDB

optional

3. python-virtualenv or pip

Backend dependencies

Backend store as pip requirement file, placed in `requirements/base.txt` folder. You can install using:

```
user@localhost template$ pip install -r requirements/base.txt
```

for automatic installation or install listed packages by yourself.

`requirements/docs.txt` contains dependence list for building documents for this project. It's not required for the proper run or work, so it could be treat as additional package deps.

Frontend dependencies

Nodejs `bower` had been chosen as frontend dependecies package manager. So you can install it via `npm install -g bower` and install frontend dep list:

```
user@localhost template$ bower install
```

Configuration and first run

Step 1

Tune/set up you database connection, store you settings into the `settings/local.py` file, for example:

```
DATABASES = {
    'default': {
        # Add 'postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
        'ENGINE': 'django.db.backends.postgresql_psycopg2', # Add 'postgresql_psycopg2
↪', 'postgresql', 'mysql', 'sqlite3' or 'oracle'.
        'NAME': 'template',                      # Or path to database file if using␣
↪sqlite3.
        'USER': 'user',                      # Not used with sqlite3.
        'PASSWORD': 'password',                  # Not used with sqlite3.
        'HOST': 'localhost',                      # Set to empty string for localhost.
↪ Not used with sqlite3.
        'PORT': '5432',                      # Set to empty string for default. Not␣
↪used with sqlite3.
    }
}
```

or leave default settings which use sqlite engine.

Step 2

run `./manage.py syncdb --migrate` script from the top level of your project:

```
root@localhost template# npm install -g bower
user@localhost template$ python ./manage.py syncdb --migrate
```

Step 3

run submodules initialization and update them

```
user@localhost template$ git submodule init media/less/select2-bootstrap-css
user@localhost template$ git submodule update media/less/select2-bootstrap-css
...
```

Step 4

compile bootstrap css file from less scheme one

```
user@localhost template$ lessc --yui-compress --no-color media/less/bootstrap.less >␣
↪media/css/bootstrap.css
```

Step 5

run dev server then open url in your browser

```
(venv) user@localhost template$ python ./manage.py runserver 0.0.0.0:8000
```

CHAPTER 2

Modules

Template standard modules:

Accounts

Forms

An account forms.

class apps.accounts.forms.**BruteForceCheckMixin**(**args*, ***kwargs*)
depends on RequestModelForm

is_valid(**args*, ***kwargs*)
Increase brute force iteration value if form is invalid

Parameters

- **args** –
- **kwargs** –

Returns is_valid statement (True or False)

save(*commit=True*)
Save brute force iteration value

Parameters **commit** – saves form instance, True by default

Returns form instance

class apps.accounts.forms.**LoginForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*)
Login Form

class apps.accounts.forms.**PasswordChangeForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*)

Password Change Initiation Form

class apps.accounts.forms.**PasswordRestoreForm**(**args*, ***kwargs*)

Password Restore Form

class apps.accounts.forms.**PasswordRestoreInitiateForm**(*data=None*, *files=None*, *auto_id=u'id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*)

Password Restore Initiation Form

Models

Models

class apps.accounts.models.**User**(**args*, ***kwargs*)

Base class implementing a fully featured User model with admin-compliant permissions.

Username, password and email are required. Other fields are optional.

static get_change_password_url()

returns change password url

get_full_name()

Returns the first_name plus the last_name, with a space in between.

get_group_permissions(*obj=None*)

Returns a list of permission strings that this user has through their groups. This method queries all available auth backends. If an object is passed in, only permissions matching this object are returned.

static get_profile_url()

returns profile url

static get_recover_password_url()

returns recover password url

get_short_name()

Returns the short name for the user.

has_module_perms(*app_label*)

Returns True if the user has any permissions in the given app label. Uses pretty much the same logic as has_perm, above.

has_perm(*perm*, *obj=None*)

Returns True if the user has the specified permission. This method queries all available auth backends, but returns immediately if any backend returns True. Thus, a user who has permission from a single auth backend is assumed to have permission in general. If an object is provided, permissions for this specific object are checked.

has_perms(*perm_list*, *obj=None*)
Returns True if the user has each of the specified permissions. If object is passed, it checks if the user has all required perms for this object.

class apps.accounts.models.**UserSID**(**args*, ***kwargs*)
UserSID class for security hashes store, uses for password recover process

Views

Class based views

An account classed based views.

class apps.accounts.views.**IndexView**(***kwargs*)
Index view

class apps.accounts.views.**LoginView**(***kwargs*)
serves for account login into current web application

form_class
alias of LoginForm

class apps.accounts.views.**LogoutView**(***kwargs*)
serves for log out registered and logged in user for current web app

class apps.accounts.views.**PasswordChangeView**(***kwargs*)
Password Change View

starts password change process (if user remembers his current password and he/she is logged in)

form_class
alias of PasswordChangeForm

model
alias of User

class apps.accounts.views.**PasswordRestoreInitiateView**(***kwargs*)
Password Restore Initiate View

starts password restore process (initiates)

form_class
alias of PasswordRestoreInitiateForm

class apps.accounts.views.**PasswordRestoreView**(***kwargs*)
Password Restore View

restores forgotten password

form_class
alias of PasswordRestoreForm

class apps.accounts.views.**ProfileView**(***kwargs*)
Profile View

show user profile

Signals

Signals

apps.accounts.signals.**user_pre_saved**(*instance*, ***kwargs*)
user pre save signal

Parameters **instance** (apps.accounts.models.User) – user instance

Return type *apps.accounts.models.User*

Returns user instance

apps.accounts.signals.**setup_signals**()
setups signals

Core

Forms

Forms

class apps.core.forms.**RequestFormMixin**(**args*, ***kwargs*)
RequestFormMixin, store Django `request` instance in self.request

Note: `request` instance should be given via keywords while form instance init

```
class RequestForm(RequestFormMixin, forms.ModelForm):
    class Meta:
        model = SomeModel

def foo(request):
    form = RequestForm(request.POST or None, request=request)
    ...
```

class apps.core.forms.**RequestModelForm**(**args*, ***kwargs*)
RequestModelForm, bases on `forms.ModelForm` class use RequestFormMixin instead of it

Views

Views

class apps.core.views.**IndexView**(***kwargs*)
Index view

class apps.core.views.**LoginRequiredMixin**
LoginRequired View Mixin

Helpers

Helpers.

apps.core.helpers.**get_content_type**(*obj*)
Gets content_type for `Object`. Works with ModelBase based classes, its instances and with format string `'app_label.model_name'`, also supports django-sphinx models and instances modification retrieves content_type or raise the common django Exception

Returns `ContentType` instance

Parameters **obj** (*django.db.models.Model or str*) – Model class, instance, basestring classpath instance

Exception ObjectDoestNotExist, MultipleObjectsReturned

```
user_content_type = get_content_type(User)
user_content_type = get_content_type(onsite_user)
user_content_type = get_content_type('auth.user')
```

apps.core.helpers.**get_content_type_or_404**(*source*)
Gets source content_type or raises Http404

Returns ContentType instance

Exception Http404

apps.core.helpers.**get_content_type_or_None**(*source*)
Gets source content_type or returns None

Returns ContentType instance or None

apps.core.helpers.**get_int_or_zero**(*value*)
get int or int(0)

Parameters **| int value** (*str*) –

Return type int

Returns int

Shortcuts

apps.core.shortcuts.**direct_to_template**(*request*, *template*, *context=None*, *processors=None*)
return response object

Parameters

- **request** – Django HttpRequest instance
- **template** – template file place on filesystem and stored in template directory ex. 'accounts/profile.html'
- **context** – dict instance with render context

```
{'context': True, 'time': datetime.now()}
```

- **processors** – context processors

Returns HttpResponse object instance

Context Processors

Context processors

apps.core.context_processors.**global_referer**(*request*)
global_referer context processor

Returns

current_referer sets to current URI (HTTP_HOST + PATH_INFO)

global_referer sets to HTTP_REFERER

```
<input value='{{ global_referer }}' name='next' />
{{ current_date|timezone:"Europe/London"|date:"d.m.Y H:i" }}
```

apps.core.context_processors.**global_settings**(*request*)
global settings context processor

Returns

gs as django.conf.settings object

get_full_path as `request.get_full_path()` instance

current_date as `timezone.now()`

```
{% load tz i18n %}
I18N is currently {{ gs.I18N|yesno:"enabled, disabled" }}
{{ current_date|timezone:"Europe/London"|date:"d.m.Y H:i" }}
```

apps.core.context_processors.**session**(*request*)
session context processor

Returns session as `request.session` instance

apps.core.context_processors.**template**(*request*)
template context processor

Returns base as `settings.DEFAULT_TEMPLATE`

```
{% extends base %}
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

c

Index

P

R

S

T

U