django-sitegate Documentation

Release 1.3.1

Igor 'idle sign' Starikov

Oct 12, 2021

Contents

http://github.com/idlesign/django-sitegate

django-sitegate is a reusable application for Django to ease sign up & sign in processes.

This application will handle most common user registration and log in flows for you.

CHAPTER 1

Requirements

1. Python 3.6+
2. Django 2.0+
3. Django Auth contrib enabled
4. Django Admin contrib enabled (optional)
5. Django Messages contrib enabled (optional)

CHAPTER 2

Table of Contents

2.1 Getting started

- Add the **sitegate** application to INSTALLED_APPS in your settings file (usually `settings.py`).
- Apply DB migrations (`manage.py migrate`).

2.1.1 Quick example

Here follows the most straightforward way possible with `django-sitegate` to have both sign up & sign in functionality on your page.

1. Use `sitegate_view` decorator to mark your view as the one handling both signups and signins:

```
from django.shortcuts import render

from sitegate.toolbox import sitegate_view

@sitegate_view  # This also prevents logged in users from accessing our␣
↪sign in/sign up page.
def entrance(request):
    return render(request, 'entrance.html', {'title': 'Sign in & Sign up'}
↪)
```

2. Then in your template load `sitegate` tag library and put `sitegate_signup_form` & `sitegate_signin_form` tags in place where you want a registration and sign in forms to be.

```
{% extends "_base.html" %}
{% load sitegate %}

{% block page_contents %}
    <div class="my_signin_block">
        {% sitegate_signin_form %}
```

(continues on next page)

(continued from previous page)

```
    </div>
    <div class="my_signup_block">
        {% sitegate_signup_form %}
    </div>
{% endblock %}
```

You're done. Now your site visitors have an e-mail + password form to register and username/e-mail + password form to log in.

2.1.2 Sign in using remotes

You can configure **sitegate** to allow users to log in with remote services, like `Yandex` and `Google`.

1. Put `sitegates.py` file in one of your applications:

```
from sitegate.signin_flows.remotes.google import Google
from sitegate.signin_flows.remotes.yandex import Yandex
from sitegate.toolbox import register_remotes

# We register our remotes.
register_remotes(
    # Register OAuth clients (web application type) beforehand

    # https://oauth.yandex.ru/client/new
    # set <your-domain-uri>/rauth/yandex/ as a Callback URL
    Yandex(client_id='<your-client-id-here>'),

    # https://console.cloud.google.com/apis/credentials/oauthclient
    # set <your-domain-uri>/rauth/google/ as a Callback URL
    Google(client_id='<your-client-id-here>'),
)
```

2. Attach sitegate URL patterns in your `urls.py`:

```
from sitegate.toolbox import get_sitegate_urls

urlpatterns = patterns('',
    ...  # your urls here
)

# attach sitegate urls
urlpatterns += get_sitegate_urls()
```

After that your users should see links to proceed using remote auth. Those links are placed just below your Sign In form.

And mind that we've barely made a scratch of **sitegate**.

2.2 Preferences

Some of `django-sitegate` behavior can be customized using preferences system.

The following sitegate preferences are located in `sitegate.settings` module.

Note: `django-sitegate` supports changing preferences runtime with the help of `django-siteprefs` - https://github.com/idlesign/django-siteprefs

2.2.1 SIGNIN_ENABLED

This indicates whether sigin is enabled. If disabled signin forms won't be rendered.

You can override the default value by defining `SITEGATE_SIGNIN_ENABLED` in `settings.py` of your project.

2.2.2 SIGNIN_DISABLED_TEXT

This text will be rendered instead of a sign in form, if sign in is disabled (see `SIGNIN_ENABLED`).

2.2.3 SIGNUP_ENABLED

This indicates whether sig up is enabled. If disabled signup forms won't be rendered.

You can override the default value by defining `SITEGATE_SIGNUP_ENABLED` in `settings.py` of your project.

2.2.4 SIGNUP_DISABLED_TEXT

This text will be rendered instead of a sign up form, if sign up is disabled (see `SIGNUP_ENABLED`).

2.2.5 SIGNUP_VERIFY_EMAIL_NOTICE

Text shown to a user after a registration form is submitted.

You can override the default value by defining `SITEGATE_SIGNUP_VERIFY_EMAIL_NOTICE` in `settings.py` of your project.

2.2.6 SIGNUP_VERIFY_EMAIL_TITLE

Title of a message sent to a user to verify his email address.

You can override the default value by defining `SITEGATE_SIGNUP_VERIFY_EMAIL_TITLE` in `settings.py` of your project.

2.2.7 SIGNUP_VERIFY_EMAIL_BODY

Body (main text) of a message sent to a user to verify his email address.

Note: This **must** include `%(url)s` marker, which will be replaced with an account activation URL.

You can override the default value by defining `SITEGATE_SIGNUP_VERIFY_EMAIL_BODY` in ``settings.py` of your project.

2.2.8 SIGNUP_VERIFY_EMAIL_SUCCESS_TEXT

A message shown to a user after he has followed an account activation URL if activation was a success.

You can override the default value by defining `SITEGATE_SIGNUP_VERIFY_EMAIL_SUCCESS_TEXT` in `settings.py` of your project.

2.2.9 SIGNUP_VERIFY_EMAIL_ERROR_TEXT

A message shown to a user after he has followed an account activation URL if there was an error during an activation process.

You can override the default value by defining `SITEGATE_SIGNUP_VERIFY_EMAIL_ERROR_TEXT` in `settings.py` of your project.

2.3 Customizing signup

django-sitegate by default uses a simple e-mail + password form. Although it is a rather common use case, inevitably there should come times, when such a registration form is not suitable. Should it be only just styling, or entire form that you want to change, or provide several registration options **sitegate** has some answers for you.

Further we'll consider a number of approaches to signup customization. But just before we start, let's talk about signup flows.

2.3.1 Signup flows

sitegate uses the notion of `signup flows` to describe signup processes.

Besides some signup logic each flow features a form which is used for registration and a template to render that form.

And, of course, every signup flow has its own **name**, so that we can address them and use at our will like so:

```
from django.shortcuts import render

# Import a flow class to use.
from sitegate.signup_flows.classic import ClassicSignup
from sitegate.decorators import signup_view

# And use that class for registration.
@signup_view(flow=ClassicSignup)
def register(request):
    return render(request, 'register.html', {'title': 'Sign up'})
```

Hopefully you've already noticed this code is a spin off from *Getting Started* section. Not much have changed since, though now we have a classical Django registration form (user name + password + password check) instead of a modern one.

The above example should give you an idea of how signup flows can differ from each other.

2.3.2 Built-in signup flows

Signup flow classes are places in `sitegate.signup_flows` module.

These are the options:

- Modern flows - `sitegate.signup_flows.modern`
  - **ModernSignup**

    Modernized registration flow based on classic from Django built-in with unique e-mail field, without username and second password fields.

    Default form: e-mail + password

    Note: E-mail in this flow is unique and automatically stored both to e-mail and username.

    Note: This sign up flow is the default one. It means that it will be used if you decorate your view with `@signup_view` decorator both without any parameters, or without `flow` parameter.

  - **InvitationSignup**

    Modernized registration flow with additional invitation code field.

    Default form: invitation code + e-mail + password

- Classic flows - `sitegate.signup_flows.classic`
  - **ClassicSignup**

    Classic registration flow borrowed from Django built-in UserCreationForm.

    Default form: username + password + password retype

  - **SimpleClassicSignup**

    Classic registration flow borrowed from Django built-in without second password field.

    Default form: username + password

    Note: Keep in mind that e-mail in the classical flows below is not unique. It means that several users may have the same e-mail.

    If you're looking for unique e-mail functionality consider using **Modern** flow pack.

  - **ClassicWithEmailSignup**

    Classic registration flow borrowed from Django built-in with additional e-mail field.

    Default form: username + e-mail + password + password retype

  - **ClassicWithEmailSignup**

    Classic registration flow borrowed from Django built-in with e-mail field, but without second password field.

    Default form: username + e-mail + password

2.3.3 Combining signup flows

You can use more than one signup flow with the same view, by stacking `@signup_view` decorators:

```
from django.shortcuts import render

from sitegate.signup_flows.classic import ClassicSignup
from sitegate.decorators import signup_view

# We'll use some our mythical MySignup flow, so let's import it.
from .my_signup_flows import MySignup

# Stack our decorators.
@signup_view(flow=MySignup)
@signup_view(flow=ClassicSignup)
def register(request):
    return render(request, 'register.html', {'title': 'Sign up'})
```

Additionally you'll need to extend your template. Let's extend the one from *Getting started* section:

```
{% extends "_base.html" %}
{% load sitegate %}

{% block page_contents %}
    <div class="my_signup_block one">
        {% sitegate_signup_form for ClassicSignup %}
    </div>
    <div class="my_signup_block two">
        {% sitegate_signup_form for MySignup %}
    </div>
{% endblock %}
```

Now your users might use either of two registration methods.

2.3.4 Form templates

sitegate uses templates to render forms bound to signup flows, and is shipped with several of them for your convenience.

Signup form templates are stored under `sitegate/templates/sitegate/signup/`. Feel free to examine them in need.

The following templates are shipped with the application:

- **form_as_p.html** - This contents identical to that produced by *form.as_p*.

  Note: This is the **default template**. It means that it will be used if you decorate your view with `@signup_view` decorator both without `template` parameter given.

- **form_bootstrap.html** - This template produces HTML ready to use with Twitter Bootstrap Framework.
- **form_bootstrap3.html** - This template produces HTML ready to use with Bootstrap Framework version 3.

  Note: This also requires *form-control* class to be batch applied for every form widget for proper form fields styling.

  See *Batch styling form widgets* section below.

  E.g: widget_attrs={'class': 'form-control'}

- **form_foundation.html** - This template produces HTML ready to use with Foundation Framework.

2.3.5 Swapping form templates

If the built-in templates is not what you want, you can swap them for your own:

```
from django.shortcuts import render

from sitegate.decorators import signup_view

# I command: use my template. Its name is `my_sign_up_form.html` %)
@signup_view(template='my_sign_up_form.html')
def register(request):
    return render(request, 'register.html', {'title': 'Sign up'})
```

Note: You can address the built-in templates both by providing a full path and with a shortcut - *filename without an extension*.

For example: `sitegate/signup/form_bootstrap.html` and `form_bootstrap` are interchangeable.

And that's all what you need to tell **sitegate** to use your custom template.

2.3.6 Batch styling form widgets

Now if the only thing that makes you uncomfortable with sign up is that form widgets (e.g. text inputs) lack styling and, say, it is required by some CSS framework you use, **sitegate** will help you to handle it.

Use `widget_attrs` parameter for `@signup_view` decorator to accomplish the task:

```
from django.shortcuts import render

from sitegate.decorators import signup_view

# Let's use the built-in template for Twitter Bootstrap
# and align widgets to span6 column,
# and use field label as a placeholder, that will be rendered by Bootstrap as a hint␣
↪inside text inputs.
@signup_view(widget_attrs={'class': 'span6', 'placeholder': lambda f: f.label},␣
↪template='form_bootstrap')
def register(request):
    return render(request, 'register.html', {'title': 'Sign up'})
```

The most interesting thing here is probably *lambda*. It receives field instance, so you can customize widget attribute values in accordance with some field data.

2.3.7 Redirect after signup

You can redirect to a URL passing `redirect_to` parameter to `@signup_view` as follows:

```
from django.shortcuts import render

from sitegate.decorators import signin_view

```

(continues on next page)

(continued from previous page)

```
# Here we redirect to `/other_url`, but Django URL pattern names are also supported.
@signup_view(redirect_to='/other_url')
def register(request):
    return render(request, 'register.html', {'title': 'Sign up'})
```

2.3.8 Restricting signups

You can restrict signups from certain domains through Django Admin interface (*Blacklisted domains* under *Sitegate* section).

There you can define domain names that are not allowed in e-mail addresses.

Please note that all signup flows with e-mail fields will automatically validate domains against the mentioned blacklist by default. To change this behaviour either override *validate_email_domain* flow class attribute or provide *validate_email_domain* keyword attribute to *signup_view* decorator.

```
...
@signup_view(validate_email_domain=False)
...
```

2.3.9 Changing user activity status

By default every signup flow creates user account with status *active* = *True*.

To change this behaviour either override *activate_user* flow class attribute or provide *activate_user* keyword attribute to *signup_view* decorator.

```
...
@signup_view(activate_user=False)
...
```

2.3.10 Automatic sign in on sign ups

By default every signup is followed on success by an automatic sign in. That could be changed by *auto_signin* = *False*.

To change this behaviour either override *auto_signin* flow class attribute or provide *auto_signin* keyword attribute to *signup_view* decorator.

```
...
@signup_view(auto_signin=False)
```

2.3.11 Signup signals

These are signal bound to signup flows. They are stored in `sitegate.signals`.

You can listen to them (see Django documentation on signals), and do some stuff when they are happen:

- **sig_user_signup_success**

  Emitted when user successfully signs up.

Parameters: `signup_result` - result object, e.g. created User; `flow` - signup flow name, 'request' - Request object.

- **sig_user_signup_fail**

  Emitted when user sign up fails.

  Parameters: `signup_result` - result object, e.g. created User; `flow` - signup flow name, 'request' - Request object.

2.3.12 Custom signup flows examples

Adding terms of service to the ModernSignup flow

Define your signup form inheriting from *ModernSignupForm*:

```
from sitegate.signup_flows.modern import ModernSignup, ModernSignupForm


class CustomizedSignupForm(ModernSignupForm):

    tos = forms.BooleanField(
        error_messages={'required': _('You must accept the terms and conditions')},
        label=_('I Agree To The Terms & Conditions')
    )

    class Meta:  # Redefine Meta if we have a custom User model.
        model = CustomizedUser
        fields = ('email', 'password1', 'phone', 'tos')
```

Define your signup flow inheriting from *ModernSignup*:

```
class CustomizedSignup(ModernSignup):

    form = CustomizedSignupForm
```

Decorate your signup view with *@signup_view(flow=CustomizedSignup)*.

2.4 Signup tweaks

Here are some tips and tweaks for **django-sitegate** signup flows.

2.4.1 Sending confirmation email for email-aware signups

By default email-aware signup flows do not ask a user to verify his email address, to change this behaviour you need to take some additional steps:

Note: This feature depends upon django-sitemessage.

Make sure it is installed and configured to use SMTP.

Note: This feature also depends upon Django Messages Contrib. Make sure it is available.

- Either override `verify_email` flow class attribute or provide ``verify_email`` keyword attribute to `signup_view` decorator:

```
...
@signup_view(verify_email=True)
...
```

- Attach **sitegate** urls to urlpatterns of your project (`urls.py`):

```
from sitegate.toolbox import get_sitegate_urls

urlpatterns = patterns('',
    ...
)

urlpatterns += get_sitegate_urls()
```

- You're done. Upon registration user will be notified he needs to confirm his email address.

  An email with account activation link will be sent by **django-sitemessage**.

Note: Texts (both sent by email and shown on site) could be customized.

See *Preferences* chapter.

2.5 Customizing sign in

django-sitegate by default uses a simple username/e-mail + password form. Although it is a rather common use case, inevitably there should come times, when such a sign in form is not suitable. Should it be only just styling, or entire form that you want to change, or provide several sign in options **sitegate** has some answers for you.

Further we'll consider a number of approaches to sign in customization. But just before we start, let's talk about sign in flows.

2.5.1 Sign in flows

sitegate uses the notion of `sign in flows` to describe sign in processes.

Besides some sign in logic each flow features a form which is used for signing in and a template to render that form.

And, of course, every sign in flow has its own **name**, so that we can address them and use at our will like so:

```
from django.shortcuts import render

# Import a flow class to use.
from sitegate.signin_flows.classic import ClassicSignin
from sitegate.decorators import signin_view

# And use that class for sign in.
@signin_view(flow=ClassicSignin)
def login(request):
    return render(request, 'login.html', {'title': 'Sign in'})
```

Hopefully you've already noticed this code is a spin off from *Getting Started* section. Not much have changed since, though now we have a classical Django log in form (username + password) instead of a modern one.

The above example should give you an idea of how sign in flows can differ from each other.

2.5.2 Built-in sign in flows

Sign in flow classes are places in `sitegate.signin_flows` module.

These are the options:

- Modern flows - `sitegate.signin_flows.modern`
  - **ModernSignin**

    Modernized sign in flow based on classic from Django built-in with username/e-mail authentication support.

    Default form: username/e-mail + password

    Note: This sign in flow is the default one. It means that it will be used if you decorate your view with `@signin_view` decorator both without any parameters, or without `flow` parameter.

    Warning: This flow assumes that both E-mail and Username fields of Django User model are inhabited with the same value. Since Django imposes different limits to those fields, maximum value length is defined by the shortest of them (Username).

- Classic flows - `sitegate.signin_flows.classic`
  - **ClassicSignin**

    Classic log in flow borrowed from Django built-in AuthenticationForm.

    Default form: username + password

2.5.3 Combining sign in flows

You can use more than one sign in flow with the same view, by stacking `@signin_view` decorators:

```
from django.shortcuts import render

from sitegate.signin_flows.classic import ClassicSignin
from sitegate.decorators import signin_view

# We'll use some our mythical MySignin flow, so let's import it.
from .my_signin_flows import MySignin

# Stack our decorators.
@signin_view(flow=MySignin)
@signin_view(flow=ClassicSignin)
def login(request):
    return render(request, 'login.html', {'title': 'Sign in'})
```

Additionally you'll need to extend your template. Let's extend the one from *Getting started* section:

```
{% extends "_base.html" %}
{% load sitegate %}

{% block page_contents %}
    <div class="my_signin_block one">
        {% sitegate_signin_form for ClassicSignin %}
    </div>
    <div class="my_signin_block two">
        {% sitegate_signin_form for MySignin %}
    </div>
{% endblock %}
```

Now your users might use either of two log in methods.

2.5.4 Form templates

sitegate uses templates to render forms bound to sign in flows, and is shipped with several of them for your convenience.

Sign in form templates are stored under `sitegate/templates/sitegate/signin/`. Feel free to examine them in need.

The following templates are shipped with the application:

- **form_as_p.html** - This contents identical to that produced by *form.as_p*.

  Note: This is the **default template**. It means that it will be used if you decorate your view with `@signin_view` decorator both without `template` parameter given.

- **form_bootstrap.html** - This template produces HTML ready to use with Twitter Bootstrap Framework.
- **form_bootstrap3.html** - This template produces HTML ready to use with Bootstrap Framework version 3.

  Note: This also requires *form-control* class to be batch applied for every form widget for proper form fields styling.

  See *Batch styling form widgets* section below.

  E.g: widget_attrs={'class': 'form-control'}

- **form_foundation.html** - This template produces HTML ready to use with Foundation Framework.

2.5.5 Swapping form templates

If the built-in templates is not what you want, you can swap them for your own:

```
from django.shortcuts import render

from sitegate.decorators import signin_view

# I command: use my template. Its name is `my_sign_in_form.html` %)
@signin_view(template='my_sign_in_form.html')
def login(request):
    return render(request, 'login.html', {'title': 'Sign in'})
```

Note: You can address the built-in templates both by providing a full path and with a shortcut - *filename without an extension*.

For example: `sitegate/signin/form_bootstrap.html` and `form_bootstrap` are interchangeable.

And that's all what you need to tell **sitegate** to use your custom template.

2.5.6 Batch styling form widgets

Now if the only thing that makes you uncomfortable with sign in is that form widgets (e.g. text inputs) lack styling and, say, it is required by some CSS framework you use, **sitegate** will help you to handle it.

Use `widget_attrs` parameter for `@signin_view` decorator to accomplish the task:

```
from django.shortcuts import render

from sitegate.decorators import signin_view

# Let's use the built-in template for Twitter Bootstrap
# and align widgets to span6 column,
# and use field label as a placeholder, that will be rendered by Bootstrap as a hint␣
↪inside text inputs.
@signin_view(widget_attrs={'class': 'span6', 'placeholder': lambda f: f.label},␣
↪template='form_bootstrap')
def login(request):
    return render(request, 'login.html', {'title': 'Sign in'})
```

The most interesting thing here is probably *lambda*. It receives field instance, so you can customize widget attribute values in accordance with some field data.

2.5.7 Redirect after sign in

You can redirect to a URL passing `redirect_to` parameter to `@signin_view` as follows:

```
from django.shortcuts import render

from sitegate.decorators import signin_view

# Here we redirect to `/other_url`, but Django URL pattern names are also supported.
@signin_view(redirect_to='/other_url')
def login(request):
    return render(request, 'login.html', {'title': 'Sign in'})
```

2.5.8 Sign in signals

You can listen to Django built-in signals from `django.contrib.auth.signals` (**user_logged_in** and **user_login_failed**), and do some stuff when they are happen

See DjangoAuth contrib documentation for more information.

2.6 Utilities

django-sitegate provides some utility functions for your convenience.

2.6.1 @sitegate_view

This decorator is a shortcut comprising three basic decorators:

- @signin_view
- @signup_view
- @redirect_signedin

This decorator can accept the same keyword arguments as `@signin_view` and `@signup_view`:

```
from django.shortcuts import render

from sitegate.toolbox import sitegate_view

# Let's use Twitter Bootstrap template, and style both sign in & sign up form␣
↪accordingly.
@sitegate_view(widget_attrs={'class': 'span6', 'placeholder': lambda f: f.label},␣
↪template='form_bootstrap')
def entrance(request):
    return render(request, 'entrance.html', {'title': 'Sign in & Sign up'})
```

2.6.2 @redirect_signedin

sitegate knowns that in most cases you don't want users to access sign in and sign up pages after they are logged in, so it gives you `@redirect_signedin` decorator for your views:

Note: This decorator redirects logged in users to another location, when they try to access the page.

Default redirect URL is a site root - **/**

```
from django.shortcuts import render

from sitegate.toolbox import redirect_signedin

@redirect_signedin  # Let's prevent logged in users from accessing our sign in page.
def login(request):
    return render(request, 'login.html', {'title': 'Login'})
```

The decorator accepts the same parameters as `redirect` from `django.shortcuts`.

It means that you can instruct it where logged in users should be redirected to:

```
@redirect_signedin('/some/url/for/those/already/logged/in/')
def login(request):
    return render(request, 'login.html', {'title': 'Login'})
```

2.6.3 USER

Django 1.5 introduces custom user model support. To be compatible with that, **sitegate** is equipped with **USER** variable which resides in `sitegate.utils`. It will always address the appropriate User model, so that you can use it in your sign up and sign in flows and forms.

Warning: Please note, that **sitegate** with its' build-in sign in/up flows relies on the fact that User model has some basic attributes: *username*, *email*, *password*, *is_active*, *set_password*.

CHAPTER 3

Get involved into django-sitegate

Submit issues. If you spotted something weird in application behavior or want to propose a feature you can do that at https://github.com/idlesign/django-sitegate/issues

Write code. If you are eager to participate in application development, fork it at https://github.com/idlesign/django-sitegate, write your code, whether it should be a bugfix or a feature implementation, and make a pull request right from the forked project page.

Translate. If want to translate the application into your native language use Transifex: https://www.transifex.com/projects/p/django-sitegate/.

Spread the word. If you have some tips and tricks or any other words in mind that you think might be of interest for the others — publish them.

CHAPTER 4

Also

If the application is not what you want for user registration, you might be interested in considering the other choices — https://www.djangopackages.com/grids/g/registration/