django-simple-friends Documentation

Release 1.1.0

Atamert Ölçgen

Mar 05, 2017

Contents

1 About django-simple-friends **3**
1.1 Highlights . . . 3

2 Setting up django-simple-friends **5**
2.1 Installation . . . 5
2.2 Development Setup . . . 5

3 Changes **7**
3.1 Version 1.0.0 - Mar 16, 2013 . . . 7
3.2 Version 0.5 - Oct 7, 2012 . . . 7
3.3 Version 0.4 - Feb 4, 2010 . . . 7

4 Package Reference **9**
4.1 How to display a list of friends for a user . . . 9

5 Package Reference **11**
5.1 Views . . . 11
5.2 Models . . . 12
5.3 Signals . . . 14
5.4 Signal Handlers . . . 14

6 Indices and tables **15**

Python Module Index **17**

Contents:

CHAPTER 1

About django-simple-friends

I started developing *django-simple-friends* when I needed a simple app that handles friendships for my project. django-friends of Pinax was mature and well written but it was also handling contacts and invitations. So I decided to create my own app and steal some ideas from *django-friends*.

You can get community support at the mailing list.

Highlights

- Only the relationships between registered users are managed.
- Friending with double confirmation. When a user adds another user as a friend, only when the other user accepts or tries to add the first user as a friend, the friendship relationship is created.
- Blocking is possible.

CHAPTER 2

Setting up django-simple-friends

Installation

1. Install *django-simple-friends* package:

```
pip install django-simple-friends
```

2. Add *friends* to your *INSTALLED_APPS* setting:

```
INSTALLED_APPS = (
    # Other apps
    'friends',
)
```

3. Run *syncdb* to create tables and seed friendship data for existing users:

```
python manage.py syncdb
```

4. Run tests to make sure the app is installed correctly:

```
python manage.py test friends
```

5. Optionally include *friends.urls* to your URLconf:

```
urlpatterns = patterns('',
    # Other entries
    (r'^friends/', include('friends.urls')),
)
```

Development Setup

If you want to develop *django-simple-friends* you can follow the steps below to set up a development environment.

1. Log-in to your GitHub account and fork the project.
2. Create a virtual environment:

```
virtualenv --no-site-packages django-simple-friends
```

3. Create a local repository:

```
cd django-simple-friends
. bin/activate
git clone git@github.com:muhuk/django-simple-friends.git src
```

> **Warning:** You need to replace *muhuk* with your GitHub username in the command above.

4. Run the tests to make sure everyting is set up correctly:

```
cd src
python example/manage.py test friends
```

5. Pick an issue to work on.

CHAPTER 3

Changes

Version 1.0.0 - Mar 16, 2013

- Some view classes are renamed:

```
BaseFriendshipActionView  ->  BaseActionView
FriendshipBlockView       ->  UserBlockView
FriendshipUnblockView     ->  UserUnblockView
```

  If you are using only the view functions or the provided `urls.py`, then you don't need to change your code.

- `related_name`'s for the `ForeignKey`'s to `User` from `FriendshipRequest` are changed as `friendshiprequests_from` and `friendshiprequests_to`. Please replace any references to `invitations_from` and `invitations_to` in your code.

Version 0.5 - Oct 7, 2012

Tested with Django versions **1.3** & **1.4**.

- `friend_list` view is removed. See `friends` template filter.
- View functions are rewritten as class based views. But they still work as aliases.
- `post_syncdb` signals to fix the issue of `Users` without `Friendships`.
- Proper Sphinx powered documentation.
- German & Spanish translations.

Version 0.4 - Feb 4, 2010

- Initial release.

CHAPTER 4

Package Reference

How to display a list of friends for a user

Use `friends()` template filter to obtain a `QuerySet` containing the `User`'s who are friends with the filter's argument:

```
{% load friends  %}
<h3>Friends</h3>
<ul>{% for friend in user|friends %}
  <li>{{ friend.get_full_name }}</li>
{% endfor %}</ul>
```

The code above should produce a result like this:

```
<h3>Friends</h3>
<ul>
  <li>Lenda Murray</li>
  <li>Sharon Bruneau</li>
  <li>Cory Everson</li>
</ul>
```

CHAPTER 5

Package Reference

Views

Class Based Views

All the views are implemented as classes but *view functions* are also provided.

class `friends.views.`**`BaseActionView`**(***kwargs*)

Base class for action views.

`set_url`(*request*, ***kwargs*)

Set the `url` attribute so that it can be used when `get_redirect_url()` is called.

`url` is determined using the following methods, in order:

- It can be set in the urlconf using `redirect_to` keyword argument.
- If `redirect_to_param` keyword argument is set in urlconf, the request parameter with that name will be used. In this case the request parameter **must** be provided in runtime.
- If the request has `redirect_to` to parameter is present, its value will be used.
- If `REDIRECT_FALLBACK_TO_PROFILE` setting is `True`, current user's profile URL will be used.
- `HTTP_REFERER` header's value will be used if exists.
- If all else fail, `'/'` will be used.

class `friends.views.`**`FriendshipAcceptView`**(***kwargs*)

class `friends.views.`**`UserBlockView`**(***kwargs*)

class `friends.views.`**`FriendshipCancelView`**(***kwargs*)

class `friends.views.`**`FriendshipDeclineView`**(***kwargs*)

class `friends.views.`**`FriendshipDeleteView`**(***kwargs*)

class `friends.views.`**`FriendshipRequestView`**(***kwargs*)

class `friends.views.`**`UserUnblockView`**(***kwargs*)

View Functions

Tip: If you want to customize the views provided, check out *Class Based Views* first.

`friends.views.`**`friendship_request`**(*request*, **args*, ***kwargs*)

`friends.views.`**`friendship_accept`**(*request*, **args*, ***kwargs*)

`friends.views.`**`friendship_decline`**(*request*, **args*, ***kwargs*)

`friends.views.`**`friendship_cancel`**(*request*, **args*, ***kwargs*)

`friends.views.`**`friendship_delete`**(*request*, **args*, ***kwargs*)

`friends.views.`**`user_block`**(*request*, **args*, ***kwargs*)

`friends.views.`**`user_unblock`**(*request*, **args*, ***kwargs*)

Models

class `friends.models.`**`FriendshipRequest`**(**args*, ***kwargs*)

An intent to create a friendship between two users.

See also:

There should never be complementary `FriendshipRequest`'s, as in `user1` requests to be friends with `user2` when `user2` has been requested to be friends with `user1`. See how `FriendshipRequestView` checks the existence of a `FriendshipRequest` from `to_user` to `from_user`.

`accept`()

Create the `Friendship` between `from_user` and `to_user` and mark this instance as accepted.

`friendship_accepted` is signalled on success.

See also:

`FriendshipAcceptView`

`cancel`()

Deletes this `FriendshipRequest`

`friendship_cancelled` is signalled on success.

See also:

`FriendshipCancelView`

`decline`()

Deletes this `FriendshipRequest`

`friendship_declined` is signalled on success.

See also:

`FriendshipDeclineView`

class `friends.models.`**`FriendshipManager`**

are_friends(*user1*, *user2*)
Indicate if `user1` and `user2` are friends.

Parameters

- **user1** (`User`) – User to compare with `user2`.
- **user2** (`User`) – User to compare with `user1`.

Return type `bool`

befriend(*user1*, *user2*)
Establish friendship between `user1` and `user2`.

Important: Instead of calling this method directly, `FriendshipRequest.accept()`, which calls this method, should be used.

Parameters

- **user1** (`User`) – User to make friends with `user2`.
- **user2** (`User`) – User to make friends with `user1`.

friends_of(*user*, *shuffle=False*)
List friends of `user`.

Parameters

- **user** (`User`) – User to query friends.
- **shuffle** (`bool`) – Optional. Default `False`.

Returns `QuerySet` containing friends of `user`.

unfriend(*user1*, *user2*)
Break friendship between `user1` and `user2`.

Parameters

- **user1** (`User`) – User to unfriend with `user2`.
- **user2** (`User`) – User to unfriend with `user1`.

class friends.models.**Friendship**(**args*, ***kwargs*)
Represents the network of friendships.

friend_count()
Return the count of `friends`. This method is used in `FriendshipAdmin`.

Return type `int`

friend_summary(*count=7*)
Return a string representation of `friends`. This method is used in `FriendshipAdmin`.

Parameters **count** (*int*) – Maximum number of friends to include in the output.

Return type `unicode`

class friends.models.**UserBlocks**(**args*, ***kwargs*)
`User`'s blocked by `user`.

block_count()
Return the count of `blocks`. This method is used in `UserBlocksAdmin`.

Return type `int`

block_summary(*count=7*)

Return a string representation of `blocks`. This method is used in `UserBlocksAdmin`.

Parameters **count** (*int*) – Maximum number of blocked users to include in the output.

Return type `unicode`

Signals

friendship_accepted

`friends.signals.`**friendship_accepted**

Sent when a *FriendshipRequest* is accepted.

Arguments sent with this signal:

sender *FriendshipRequest* instance that is accepted.

friendship_declined

`friends.signals.`**friendship_declined**

Sent when a *FriendshipRequest* is declined by `to_user`.

Arguments sent with this signal:

sender *FriendshipRequest* instance that is declined.

friendship_cancelled

`friends.signals.`**friendship_cancelled**

Sent when a *FriendshipRequest* is cancelled by `from_user`.

Arguments sent with this signal:

sender *FriendshipRequest* instance that is cancelled.

Signal Handlers

`signals.`**create_friendship_instance**(*sender*, *instance*, *created*, *raw*, ***kwargs*)

Create a *FriendshipRequest* for newly created `User`.

See also:

`post_save` built-in signal.

`signals.`**create_userblocks_instance**(*sender*, *instance*, *created*, *raw*, ***kwargs*)

Create a *UserBlocks* for newly created `User`.

See also:

`post_save` built-in signal.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

friends.models, 12
friends.signals, 14
friends.views, 11

Index

A

accept() (friends.models.FriendshipRequest method), 12
are_friends() (friends.models.FriendshipManager method), 12

B

BaseActionView (class in friends.views), 11
befriend() (friends.models.FriendshipManager method), 13
block_count() (friends.models.UserBlocks method), 13
block_summary() (friends.models.UserBlocks method), 14

C

cancel() (friends.models.FriendshipRequest method), 12
create_friendship_instance() (friends.signals method), 14
create_userblocks_instance() (friends.signals method), 14

D

decline() (friends.models.FriendshipRequest method), 12

F

friend_count() (friends.models.Friendship method), 13
friend_summary() (friends.models.Friendship method), 13
friends.models (module), 12
friends.signals (module), 14
friends.signals.friendship_accepted (in module friends.signals), 14
friends.signals.friendship_cancelled (in module friends.signals), 14
friends.signals.friendship_declined (in module friends.signals), 14
friends.views (module), 11
friends_of() (friends.models.FriendshipManager method), 13
Friendship (class in friends.models), 13
friendship_accept() (in module friends.views), 12
friendship_cancel() (in module friends.views), 12
friendship_decline() (in module friends.views), 12
friendship_delete() (in module friends.views), 12
friendship_request() (in module friends.views), 12
FriendshipAcceptView (class in friends.views), 11
FriendshipCancelView (class in friends.views), 11
FriendshipDeclineView (class in friends.views), 11
FriendshipDeleteView (class in friends.views), 11
FriendshipManager (class in friends.models), 12
FriendshipRequest (class in friends.models), 12
FriendshipRequestView (class in friends.views), 11

S

set_url() (friends.views.BaseActionView method), 11

U

unfriend() (friends.models.FriendshipManager method), 13
user_block() (in module friends.views), 12
user_unblock() (in module friends.views), 12
UserBlocks (class in friends.models), 13
UserBlockView (class in friends.views), 11
UserUnblockView (class in friends.views), 11