djangotribune Documentation

Release 0.7.6

David THENON

October 19, 2014

Contents

Django-tribune is a *chat-like application* with some aspects of *IRC* but with a strong usage of message clocks.

Message clocks are always displayed and used in messages to reference answers or relation with other messages.

A sample part of Tribune messages will look like this in a plain-text version :

```
16:15:27     <superman>           First
16:16:13     Anonymous coward     16:15:27 oh no you don't !
16:15:27     <superman>           16:16:13 liar !
18:39:01     Mozilla/5.0          Hello world !
18:39:05     <superman>           18:39:01 hello
18:43:22     Anonymous coward     18:39:01 yo
```

The application has a rich interface but is also accessible from third client application (via a XML backend) and even in plain text.

CHAPTER 1

Features

- Easy embedding in your Django webapp;
- Various backends *Formats*;
- *Action commands*;
- *Message filtering system* usually called a *BaK*;
- A rich interface (currently in development);
- Full localization for french and english language;
- *Discovery* XML configuration file for third client applications (aka remote client or *coincoins*);
- Channel support;
- South support;

CHAPTER 2

Links

- Download his PyPi package;
- Clone it on his Github repository;

2.1 Contents

2.1.1 Install

First, register the app in your project settings like this :

```
INSTALLED_APPS = (
    ...
    'djangotribune',
    ...
)
```

Then after you should register the app urls in your project `urls.py` :

```
url(r'^tribune/', include('djangotribune.urls')),
```

Of course, you can use another mounting directory than the default `tribune/` or even use your own app urls, look at the provided `djangotribune.urls` to see what you have to map.

And finally don't forget to do the Django's *syncdb command* to synchronize models in your database.

If needed, you can change some Application settings in your settings file.

Note: The recommended database engine is **PostgreSQL**. With SQLite you could have problems because the application makes usage of case-insensitive matching notably in *Message filtering system*.

Project templates

A simple note about templates, djangotribune templates use a base template `djangotribune/base.html` to include some common HTML to add contents to your layout, and all other templates extend it to insert theirs.

This base template is made to extend a `skeleton.html` template that should be the root base of your project layout. Therefore, if you don't use a base template or use it with another name, just override `djangotribune/base.html` in project templates to fit it right within your project.

Also, note that templates have been written for Foundation3 so if you don't use it, it should not really matter to you as this is only HTML, you can style it yourself or at least change the templates to accomodate to your needed HTML structure. And if you want to use Foundation3, just add its required assets to your project, but for the url archives form you will have to patch some Javascript because of an added feature to use input checkbox inside button dropdown.

Here is a patch file for the Javascript file for buttons :

```
diff --git a/project/webapp_statics/js/foundation/jquery.foundation.buttons.js b/project/webapp_stati
--- a/project/webapp_statics/js/foundation/jquery.foundation.buttons.js
+++ b/project/webapp_statics/js/foundation/jquery.foundation.buttons.js
@@ -33,6 +33,12 @@
        button = $el.closest('.button.dropdown'),
        dropdown = $('> ul', button);

+        // let ".no-reset-click" elements to act by default to prevent dropdown closing
+        if($(e.target).hasClass('no-reset-click')){
+          e.stopPropagation();
+          return true;
+        }
+
```

And another patch file for your `app.js` :

```
diff --git a/project/webapp_statics/js/foundation/app.js b/project/webapp_statics/js/foundation/app.j
--- a/project/webapp_statics/js/foundation/app.js
+++ b/project/webapp_statics/js/foundation/app.js
@@ -11,6 +11,15 @@ function column_equalizer(){
}

$(document).ready(function() {
+    // Automatically add "no-reset-click" class on direct input parent label to
+    // follow their natural behavior (to propagate the click to their input child,
+    // usually only for radio or checkbox)
+    $("form .button.dropdown .no-reset-click").each(function(index) {
+        if($(this).parent().prop('nodeName')=='LABEL'){
+            $(this).parent().addClass('no-reset-click');
+        }
+    });
+
    //$.fn.foundationAlerts           ? $doc.foundationAlerts() : null;
    $.fn.foundationButtons          ? $doc.foundationButtons() : null;
    //$.fn.foundationAccordion        ? $doc.foundationAccordion() : null;
```

Updates

Since 0.6.6 version, South support is implemented, so for future updates you will have to use something like :

```
./manage.py migrate djangotribune
```

And model changes will be automatically applied to your database.

Application settings

All default app settings are located in the `settings_local.py` file of `djangotribune`, you can modify them in your project settings.

Note: All app settings are overwritten if present in your project settings with the exception of dict variables. This is

to be remembered when you want to add a new entry in a list variable, you will have to copy the default version in your settings with the new entry otherwise default variable will be lost.

TRIBUNE_LOCKED When set to `True` all anonymous users will be rejected from any request on remote views, post views and board views, only registred users will continue to access to these views.

By default this is set to `False` so anonymous and registred users have full access to any *public views*.

TRIBUNE_MESSAGES_DEFAULT_LIMIT Default message limit to display in backend.

Requires an integer, by default this is set to 50.

TRIBUNE_MESSAGES_MAX_LIMIT The maximum value allowed for the message limit option. Limit option used beyond this will be set to this maximum value.

Requires an integer, by default this is set to 100.

TRIBUNE_MESSAGES_POST_MAX_LENGTH Maximum length (in characters) for the content message.

Requires an integer, by default this is set to 500. You have no real limit on this value because this is stored in full text field without limit.

TRIBUNE_SMILEYS_URL Template string for smileys URL, this is where you can set the wanted smiley host. By default this is set to :

```
http://totoz.eu/{0}.gif
```

So the host will be *totoz.eu* that is the *safe for work* version, if you prefer the *non safe for work* use `nsfw.totoz.eu` instead.

TRIBUNE_TITLES List of titles randomly displayed on tribune boards.

The default one allready contains many titles.

TRIBUNE_LASTFM_API_URL The URL to use to request the LastFM API used within `lastfm` action command.

TRIBUNE_LASTFM_API_KEY The Application key to use for on requests made to LastFM API.

TRIBUNE_INTERFACE_REFRESH_SHIFTING The default time in milli-seconds between each backend refresh request on the interface.

TRIBUNE_SHOW_TRUNCATED_URL A boolean to define (if `True`) if URLs should be displayed as a truncated url of 100 characters maximum. Default behavior (when `False` or not in your settings) is to display them like `[url]` if it does not match any regex in the dictionnary `parser.URL_SUBSTITUTION`.

Internationalization and localization

This application make usage of the Django internationalization system, see the Django documentation about this if you want to add a new language translation.

2.1.2 Basics

Message backends

Backends are available in various formats, each format has its own special features. Usually, *JSON* is for webapp usage, *XML* for remote clients and *Plain* for some nerdz.

Formats

Plain-text Very light, use the raw message, ascendant ordered by default. Url path from the tribune is `remote/` for backend and `post/` for post view.

XML Very fast, use the remote message render, descendant ordered by default. Url path from the tribune is `remote/xml/` for backend and `post/xml/` for post view.

CRAP XML The *extended* XML version to go well with old tribune application clients. Currently the only diff is the XML structure wich is indented. Url path from the tribune is `crap/remote.xml` for backend and `crap/post.xml` for post view.

JSON Very *declarative*, use the web message render, descendant ordered by default. Url path from the tribune is `remote/json/` for backend and `post/json/` for post view.

Note: For channel backend and post urls you must prepend the path with the channel slug, by example with a channel slug `foo` for the XML backend you will need to do `foo/remote/xml/`.

Url arguments

On backend URLs, you can set some options by adding URL arguments like this :

```
/remote/?limit=42&direction=asc&last_id=77
```

limit An integer to specify how many messages can be retrieved. This value cannot be greater than the setting value `TRIBUNE_MESSAGES_MAX_LIMIT`. Default value comes from setting `TRIBUNE_MESSAGES_MAX_LIMIT` if this option is not specified.

direction Message listing direction specify whether the list should be ordered by `id` in ascendant or descendant way. Value can be `asc` for ascendant or `desc` for descendant. Each backend can have its own default direction.

last_id The last `id` from wich to retrieve the messages in the interval of the `limit` option.

For example, with a *tribune* with 42 messages numbered (on their `id`) from 1 to 42, and with default limit to 30 :

- Requesting a backend without any option will return messages from `id` 13 to 42;
- Requesting a backend with option `limit` to 10, will return messages from `id` 33 to 42;
- Requesting a backend with option `last_id` to 15 will return messages from `id` 16 to 42;
- Requesting a backend with option `limit` to 5 and option `last_id` to 15 will return messages from `id` 38 to 42;

No matter what direction you specify in option, the results will stay identical.

Message posting system

The tribune can either be used from the web interface or via remote client applications.

Remote client applications

Remote clients can send a new message directly within a **POST** request and by putting the content in a `content` argument.

- Validated messages from a request without `last_id` defined return an empty Http200 response in plain-text;

- Validated messages from a request with `last_id` defined return the last updated backend (from the *known* last id);
- Unvalid message return an Http error.

All POST response for validated message return a **X-Post-Id** header that contain the ID of the new message.

Url arguments options can be given for the POST request and they will be used for the returned backend in case of success.

In fact, remote client applications should always give the `last_id` option (taken from the last message they know just before sending the POST request) to receive only messages they didn't know (and not the whole backend).

Dealing with errors

- This is not really an error, but remote backend returns a **Http304** (*NotModified*) when you try to fetch a backend with no new message;
- If the *POST* request is invalidated (with the form), the returned response will be a **Http400** (*Bad Request*) with an explanation in Ascii;
- A **Http404** is returned when you try to use a channel remote backend that doesn't exists;
- You could receive a **Http500** (*Internal Server Error*) in case of bugs or bad configured server;
- Sometimes you can receive a **Http403** if you try to use a restricted command but there are not implemented yet.

Message filtering system

All users (registred and anonymous) can manage their own entries for filtering messages on various pattern. These filters are stored in the user session in an object called BaK as *Boîte à Kons* (eg: *Idiots box*) which is persistent in your session.

That being so, a user can lose its session (after a very long inactivity or when logged out) so there are option to **save** the filters of your BaK in your profile in a database. So you can **load** them in your session when needed.

There is two ways to manage filters from your bak :

- You can use **the easy way** which always assumes you use an exact pattern, this is the purpose of options **add** and **del** than expects only two arguments, a target and the pattern;
- Or you can use **the verbose way** which expects three arguments respectively the target, the kind and the pattern, this is the purpose of options **set** and **remove**;

Available arguments

target The part of the message which will be used to apply the filter, available targets are :

- `ua` for the user-agent;
- `author` for the author username only effective for messages from registered users;
- `message` for the message in his raw version (as it was posted).

kind The kind of matching filter that will be used. Only used in the *verbose way* options, for the *easy way* this is always forced to an exact matching.

Kinds are written like *operators*, the available kinds are :

- `*=` for Case-sensitive containment test;
- `|=` for Case-insensitive containment test;

- == for Case-sensitive exact match;
- ~= for Case-insensitive exact match;
- ^= for Case-sensitive starts-with;
- $= for Case-sensitive ends-with.

pattern The pattern to match by the filter. This is a simple string and not a regex pattern. You can use space in your pattern without quoting it.

Options details

add The *easy way* to add a new filter. This requires two arguments, the target and the pattern like that :

```
/bak add author Badboy
```

del The *easy way* to drop a filter. This requires two arguments, the target and the pattern that you did have used, like that :

```
/bak del author Badboy
```

set The *verbose way* to add a new filter. This requires three arguments, the target, the kind operator and the pattern like that :

```
/bak set author == Badboy
```

remove The *verbose way* to drop a filter. This requires three arguments, the target, the kind operator and the pattern like that :

```
/bak remove author == Badboy
```

save To save your current filters in your session to your profile in database, this works only for registered users.

Saving your filters will overwrite all your previously saved filters, so if you just want to add new filters, load the previously saved filters before.

This is option does not require any argument :

```
/bak save
```

load To load your previously saved filters in your current session. If you already have filters in your current session this will overwrite them.

This is option does not requires any argument :

```
/bak load
```

on To enable message filtering using your filters in current session. A new session have message filtering enabled by default.

This is option does not requires any argument :

```
/bak on
```

off To disable message filtering using your filters in current session. The filters will not be dropped out of your session so you can enable them after if needed.

This is option does not requires any argument :

```
/bak off
```

reset To clear all your filters in current session. You can use this option followed after by a save action to clear your saved filters too.

This is option does not requires any argument :

```
/bak reset
```

Note: Messages filters will not be retroactive on displays on remote clients, only for new message to come after your command actions. So generally you will have to reload your client to see applied filters on messages posted before your command actions.

2.1.3 Usage

Message board

The tribune can either be used from the web interface or via remote client applications.

Just enter in, and type your message in the input box at the bottom of the list of messages. You can use shortcut buttons to add html tags to surround your current text selection in the input (or just add it if you didn't select text).

Also in your message you can insert smileys (commonly called *totoz*) from the smiley host (by default http://totoz.eu/). Smileys syntax is to surround the smiley key word with `[:keyname]` like `[:totoz]` that will be replaced to a link to the image `http://totoz.eu/totoz.gif`. This image will be displayed when the mouse cursor hovers the link.

The default interface performs a periodical request on the remote backend to display any new message, so you don't have to reload the page to see new message. When the the periodical refresh is on progress you will see a sign in the input, if the server return a response error you will see a sign that will be hidden at the next refresh success response.

If your posted message is not validated, the input field will be displayed with red borders, the borders will be hidden just after a new validated post.

In fact, the only option you can manage is the *Active refresh* that you can disable to avoid any periodical request on the remote backend. But if you disable it and you post a new message, there will still be a *POST* request that will refresh the message list.

Action commands

Action commands can be passed to message content, generally this results in doing the action without saving a new message although some actions can push a message to save.

All action command must start with a `/` followed (without any separator) by the action name and then the action arguments if any. Invalid action commands will often result in saving the content as a new message.

name This allows anonymous users to display a custom name instead of their *User-Agent* in messages.

Name saving is made by a special cookie, so if the user loses or deletes his cookie, he loses his custom name.

Add new ua :

```
/name My name is bond
```

Remove the saved ua :

```
/name
```

Note that this name will only be directly visible for anonymous user, because registered users have their username displayed, but the name (or user-agent) is visible on mouse over their username. This behavior is only on HTML board, remote clients have their own behaviors.

nick This is an alias for `name` action command, it work exactly the same.

lastfm This command use the LastFM API to automatically post a *musical instant* for the current track played. This works only the **current** track played, not the last recent track played.

You should specify an *username* in argument within the action, it will be used as the username account on LastFM from where to search the current track.

Generally, you will do like this :

```
/lastfm instant myname
```

But if you are authenticated on the tribune and your username is the same as on your LastFM account, you can do like this :

```
/lastfm instant
```

This will result in a message like this :

====> Moment Artist - Title <====

bak Intended for users to manage their message filters, see *Message filtering system* for a complete explanation.

If you want to avoid displaying message from the registered user `BadBoy`, you will do :

```
/bak add author Badboy
```

You want to avoid displaying all message containing a reference to `http://perdu.com` you will do :

```
/bak set message *= http://perdu.com
```

You want to avoid displaying message from all user with an user-agent from `Mozilla` :

```
/bak set ua *= Mozilla
```

Discovery

Discovery files describes the needed configuration to use a tribune with third client applications.

They are simple XML files for describe configuration to access to the remote backend and to post new message, plus some other options and parameters.

You can access them at location `/discovery.config` under the path of the tribune, so for the default tribune this is usually :

```
/tribune/discovery.config
```

And for a channel with the slug name “foo”, it will be :

```
/tribune/foo/discovery.config
```

CHAPTER 3

Indices and tables

- *genindex*
- *search*