lck.django Documentation

Release 0.8.10

Łukasz Langa

Sep 27, 2017

Contents

1 How to run the tests **3**

2 Change Log **5**

2.1 0.8.10 . . . 5
2.2 0.8.9 . . . 5
2.3 0.8.8 . . . 5
2.4 0.8.7 . . . 6
2.5 0.8.6 . . . 6
2.6 0.8.5 . . . 6
2.7 0.8.4 . . . 6
2.8 0.8.3 . . . 6
2.9 0.8.2 . . . 6
2.10 0.8.1 . . . 7
2.11 0.8.0 . . . 7
2.12 0.7.14 . . . 7
2.13 0.7.13 . . . 7
2.14 0.7.12 . . . 7
2.15 0.7.11 . . . 8
2.16 0.7.10 . . . 8
2.17 0.7.9 . . . 8
2.18 0.7.8 . . . 8
2.19 0.7.7 . . . 8
2.20 0.7.6 . . . 8
2.21 0.7.5 . . . 8
2.22 0.7.4 . . . 9
2.23 0.7.3 . . . 9
2.24 0.7.2 . . . 9
2.25 0.7.1 . . . 9
2.26 0.7.0 . . . 9
2.27 0.6.7 . . . 9
2.28 0.6.6 . . . 9
2.29 0.6.5 . . . 10
2.30 0.6.4 . . . 10
2.31 0.6.3 . . . 10
2.32 0.6.2 . . . 10
2.33 0.6.1 . . . 10

2.34 0.6.0 . . . 10
2.35 0.5.8 . . . 11
2.36 0.5.7 . . . 12
2.37 0.5.6 . . . 12
2.38 0.5.5 . . . 12
2.39 0.5.4 . . . 12
2.40 0.5.3 . . . 12
2.41 0.5.2 . . . 12
2.42 0.5.1 . . . 13
2.43 0.5.0 . . . 13
2.44 Ancient history . . . 13

3 lck.django **15**
3.1 Overview . . . 15
3.2 Extensions for `settings.py` . . . 16
3.3 Custom `manage.py` commands . . . 19
3.4 License . . . 19
3.5 TODO . . . 19

4 Indices and tables **21**

This library consists of various Django-related routines that extend or modify the behaviour of the framework:

- lots of composable abstract models to use
- a user activity log app storing users' IP addresses and user agents (useful for hunting down multi-accounts)
- a `score` app enabling users on websites to vote on objects
- a `tags` app which supports tagging by users and localized tags
- a `badges` app which enables users to receive badges for actions on the website
- extensions for `settings.py` (current directory resolution, namespace package support, settings profile support)
- typical filters, template tags, form fields, etc.

Complete documentation for the package can be found here:

http://packages.python.org/lck.django/

The latest version can be installed via PyPI:

```
$ pip install lck.django
```

or:

```
$ easy_install lck.django
```

The source code repository and issue tracker are maintained on GitHub.

This package bundles some royalty free static images that are useful in almost every Django project:

- Silk icons 1.3 by FamFamFam - requires attributing the author
- Silk Companion 1 by Damien Guard - requires attributing the author
- Country Flags by SenojFlags.com - requires using the following HTML:

```
<a href="http://www.senojflags.com">Country flag</a> image from
<a href="http://www.senojflags.com">Flags of all Countries</a>
```

For the curious, `lck` stands for LangaCore Kit. LangaCore is a one man software development shop of mine.

Note: `lck.common` requires **Python 2.7** because all of its code is using the so-called four futures (`absolute_imports`, `division`, `print_function` and `unicode_literals`). One of the virtues in the creation of this library is to make the code beautiful. These switches give a useful transitional state between the old Python 2.x and the new Python 3.x. You should use them as well.

Note: Since 0.5.0 `lck.django` requires **Django 1.3** because it makes my monkey-patching efforts much easier. Moreover, 1.3 nicely deprecates behaviour which I consider ugly.

CHAPTER 1

How to run the tests

The easiest way would be to run:

```
$ DJANGO_SETTINGS_MODULE="lck.dummy.settings" DJANGO_SETTINGS_PROFILE="test" django-
↪admin.py test
```

This command runs the internal Django tests as well and that's fine because there are monkey patches and other subtleties that should better be tested for potential breakage.

The dummy project is also used as an example of setting up a Django project. However, it seems Django tests are not happy with some changes to the settings so we're using the `test` profile (which loads overrides from `settings-test.py`) to avoid that.

CHAPTER 2

Change Log

0.8.10

- `profile` now properly rolls back a failed transaction in `create_a_user_profile_ignoring_dberrors()` (fixes initial syncdb's superuser creation on PostgreSQL)
- introduced automatic resolution of dependencies for initial migration. If you use `activitylog`, `badges`, `score` or `tags`, unless you're using the plain `auth.User`, their initial migrations depend on your profile module being ready. Migrations currently depend on the initial migration of your respective `ACTIVITYLOG_PROFILE_MODEL`, `EDITOR_TRACKABLE_MODEL`, `SCORE_VOTER_MODEL` or `TAG_AUTHOR_MODEL`. If those migrations should depend on a different migration or none, new settings have been added.

0.8.9

- fixed a regression from 0.8.8 in synchronous `activitylog` models
- `dj.choices` requirement bumped to 0.9.2 (support for Python 2.6 - 3.3)
- `dj.chain` requirement bumped to 0.9.2 (support for Python 2.7 - 3.3)

0.8.8

- `activitylog` now properly encloses database updates in transactions
- `activitylog` on RQ and Celery now properly handles non-null constraints on models
- `SessionAwareLanguageMiddleware` simplified, now simply sets the language argument in the session. This requires changing middleware order: this middleware should come after `SessionMiddleware` and before `LocaleMiddleware`.

- The default `INSTRUMENTATION_RULE` is now simply `lambda request: False` which makes `TimingMiddleware` behave better with front-end caches (if session is not accessed, `Vary: Cookie` is not set).

0.8.7

- `activitylog` now sports new async modes with built-in support for RQ or Celery workers
- minor performance updates in `tags` models

0.8.6

- minor performance updates in `activitylog` middleware. Now behaves better in low-memory + slow I/O environments.

0.8.5

- `whois` management command introduced to help find users by session ID
- using `User` attributes proxied from a `Profile` instance no longer causes a query for each call

0.8.4

- `TimeTrackable` models can now force marking fields as dirty with `mark_dirty()` and `mark_clean()` methods

0.8.3

- `concurrent_get_or_create` will now raise AssertionErrors if given either too many fields (e.g. not all of which are unique or compose a unique-together constraint) or too few (e.g. fields do not form a whole unique-together constraint). Non-unique fields should be passed in the `defaults` keyword argument if needed at object creation time.
- `profile` now implements automatic profile account synchronization by registering a post-save signal on User and creating an `AUTH_PROFILE_MODEL` instance. A management command for existing applications called `sync_profiles` has been created.
- Unit tests converted to unittest2 format

0.8.2

- fixed regression from 0.8.1: removed savepoint support since the updated `concurrent_get_or_create` fails miserably on MySQL due to dogdy savepoint support in MySQL-python

0.8.1

- `concurrent_get_or_create` based on `get_or_create` from Django 1.4.2
- `namespace_package_support` extended to cover `django.utils.translation` as well (previously namespace-packaged projects only worked with I18N if `setup.py develop` or `pip install -e .` was used to install them)
- `dj.chain` requirement bumped to 0.9.1 (supports more collective methods)

0.8.0

- `lazy_chain` moved to a separate dj.chain package. The old interface is thus deprecated and will be removed in a future version.
- `activitylog` updates: removed redundant user fields so it works again with `ACTIVITYLOG_PROFILE_MODEL` set to `auth.User`
- `EditorTrackable` doesn't require overriding `get_editor_from_request` anymore if `EDITOR_TRACKABLE_MODEL` is set to a profile model instead of `auth.User`
- profile admin module includes a predefined `ProfileInlineFormSet` for inclusion of profile-tied models to the `UserAdmin` as inlines
- the dummy application now passes all internal Django unit tests in versions 1.4.0 - 1.4.2

0.7.14

- `lazy_chain`: the fix from 0.7.13 introduced a different kind of bug, reverted and fixed properly now. More tests included.
- `flatpages` now serve content in the default language if the language requested by the browser is unavailable.
- some internal cleanups

0.7.13

- `lazy_chain`: when iterating over a slice, the iterator fetched one item too many. It didn't yield it back so the result was correct but if using `xfilter()` that caused unnecessary iteration.
- `dj.choices` requirement bumped to 0.9.0 (choices are `int` subclasses, `unicode(choice)` is now equivalent to `choice.desc`)

0.7.12

- namespace package support now works with Unicode literals in settings.py
- dummy app settings refinements: timing middleware moved down the stack because it uses the user session, WSGI app definition was wrong

0.7.11

- No code changes
- `dj.choices` requirement bumped to 0.8.6 (fully compatible with 0.8.5 and significantly improves `ChoiceFields`)

0.7.10

- `BACKLINKS_LOCAL_SITES` setting to control if all configured sites should be considered local upon backlink discovery
- More backlink fixes data model fixes to make it more cross-compatible with different backends

0.7.9

- Fixed backlink hash generation in `activitylog`
- `activitylog` accepts UTF-8 characters in `User-Agent` headers
- `activitylog` South migration #0002 now also works on backends with DDL transactions (e.g. Postgres)

0.7.8

- Fixed South support for custom fields (`DefaultTags` and `MACAddressField`).

0.7.7

- South migrations supported across the board. For existing installations you should run:

```
$ python manage.py migrate APP_NAME 0001 --fake
$ python manage.py migrate APP_NAME
```

  where `APP_NAME` is `activitylog`, `badges`, `common`, `flatpages`, `profile`, `score` or `tags`.
- uniqueness constraints in `activitylog.models.Backlink` and `activitylog.models.UserAgent` moved to separate `hash` fields to make MySQL happy. South migrations should handle schema evolution regardless of the backend you're using.

0.7.6

- Further Django 1.4 compatibility improvements: auto-compelete foreign key mixin works correctly now

0.7.5

- Django 1.4 compatibility improved

0.7.4

- Django 1.4 `USE_TZ = True` compatibility
- example settings updated to support new Django 1.4 settings
- `User` attribute proxying in `Profile` models rewritten to support all built-in and custom attributes on the `User` model
- `activitylog.middleware` now records IPs and user agents for unauthenticated requests as well. Possibly a performance hit.

0.7.3

- Added *order_by* argument to TagStem.objects.get_content_objects()

0.7.2

- choices moved to a separate dj.choices package. The old interface is thus deprecated and will be removed in a future version.

0.7.1

- fixed a regression from 0.7.0 in `lck.django.score` after cleaning up helpers

0.7.0

- `lck.django.badges` introduced
- `lck.django.common` cleaned up, `lazy_chain` significantly upgraded (now properly supports multiple iterables with filtering, slicing and sorting)

0.6.7

- `lck.django.score`: send a signal on total score change (allows for caching strategies on the app side)
- `maxid` management command introduced: for every registered model returns the current maximum value for primary keys

0.6.6

- `MACAddressField` MAC address normalization ignores empty values, supports Cisco `0000.0000.0000` notation and fixes a minor regression from 0.6.5
- `SessionAwareLanguageMiddleware` introduced
- a convenient tag getter for taggables, improved compatibility with `EditorTrackable`

0.6.5

- more rigorous normalization of MAC addresses in `MACAddressField`

0.6.4

- `ImageModel` introduced
- `Named` models name field extended to 75 characters of length

0.6.3

- fixed an embarassing bug with the human-readable `timediff` filter

0.6.2

- `MACAddressField` normalization bug fixed

0.6.1

- buttonable Django admin with `ModelAdmin`
- "Edit separately" links for ForeignKey fields supported in `ModelAdmin`
- compressing `PyLibMCCache` backend in `lck.django.cache_backends`
- backlinks support in `activitylog`
- images crushed and optimized
- use Pillow instead of PIL

0.6.0

Oh boy, lots of changes!

- `TimeTrackable` just got a lot smarter. Includes `cache_version` attribute automatically updated on significant changes to the object. `modified` gets updated only when there are actual changes to the object. `dirty_fields` property shows changed attributes from last save (works also for objects composed from multiple models, including abstract ones).

  Inspired by David Cramer and Simon Willison at EuroPython 2011.

- The dogpile-safe `lck.django.cache` now supports custom invalidators which enables invalidation not only by time but also by e.g. model changes (think `TimeTrackable.cache_version`).
- Settings profile support now requires a modified `manage.py` script in the Django project. This is forced by the unfortunate design of how Django loads settings.

- Activity logging moved to its own app, `lck.activitylog`, which now also tracks IPs and user agents of logged-in visitors (useful in hunting multi-accounts).
- Introduced a `SavePrioritized` abstract model which adds priorities to saves on models. Various parts of the application can specify which priority they use. If they update an attribute which was first saved by something with higher priority, the update is silently ignored.
- Introduced a concurrency-aware variant of the popular `Model.objects.get_or_create` (unsurprisingly called `concurrent_get_or_create`)
- Introduced a `commit_on_success` variant that supports nesting (unsurprisingly called `nested_commit_on_success`)
- Introduced `BasicAuthMiddleware` for simplistic private URL protecting.
- `EditorTrackable` is now safe in terms of foreign key cascading (content authored or modified by a user won't get deleted after this user is removed from the DB). Plus some nice admin refinements.
- Now `TimingMiddleware` doesn't break other middlewares using `process_view()` and is generally smarter.
- Added `X-Slo` header in responses for `TimingMiddleware`.
- `render()` now calculates and emits ETags based on the rendering output.
- `typical_handler()` can now `redirect_on_success`.
- Links from the BBCode filter now open in a new window and have `rel="nofollow"` set.
- Introduced a `{%settings KEY%}` templatetag.
- Introduced a `{%git_version%}` templatetag which returns a short string useful to present as an app version. This is based on the latest commit in the Git repository where the Django project lies in.
- The `cycle_filter` template filter now supports explicit counter settings and incrementation.
- Introduced template filters converting to and from Base64.
- Introduced JQuery UI and JQueryMobile integrated radio widgets.
- Improved documentation.
- More complete translations.

0.5.8

- Simplistic `TimingMiddleware` introduced.
- Profiles based on `BaseProfile` now return `self` for `get_profile()`.
- Trophy icons added.
- Console tag library introduced with the {%color%} tag.
- Allow rendering non-request contexts.
- `Choices.ToNames` decorator introduced.
- Pre-importing in `manage.py shell` works also for models with a custom‘‘app_model‘‘.

0.5.7

- `EditorTrackable` introduced
- Choices can be rendered in grouped form. Currently requires adding `'--keyword=Group:2 '` to xgettext invocations in django/core/managemenet/commands/makemessages.py. Cleaning that up is planned for 0.6.0.
- `typical_handler` works now with forms w/o a `save()` method
- `upperfirst` filter introduced: ups only the first character
- Square thumbnails for wide images now work properly
- moved contents of helpers to common (enables i18n and cleans up the API), the helpers module is therefore deprecated
- some i18n updates

0.5.6

- in the thumbnail filter, support for automatic cropping to square introduced
- minor translation updates

0.5.5

- group members inherit shifted attributes

0.5.4

- minor updates to `PolishDateWidget`

0.5.3

- `AvatarSupport` abstract model for custom avatars. `GravatarSupport` can be used as fallback or independently.
- `typical_handler` now properly supports file uploads
- bugfixes: objects without any score don't cause exceptions anymore
- leftovers from namespace changes cleaned up

0.5.2

- monkey patches of core Django annotated and regrouped for easier management in the future (yup, more to come)
- a stats calculator
- minor bugfixes

0.5.1

- tags now support models with custom managers
- for Named and Titled models a read-only `name_urlencoded` and `title_urlencoded` properties were introduced. Useful as arguments in template tags.
- support for setting additional attributes on choices using an unholy << operator overload
- in tags, support for getting objects marked with specific stems

0.5.0

- migrated to the `lck` namespace from `langacore.kit`
- migrated licensing from GPL 3 to MIT
- bumped the trove from alpha status to beta, the code is in production for over a year now

Ancient history

- No proper change log was kept before 0.5.0

CHAPTER 3

lck.django

Overview

This library consists of a number of reusable Django apps and a set of common functionality that enables them to work.

Module details

For more detailed view on the modules, see the documentation below.

cache
choices
filters
common
common.forms
common.middleware
common.models
common.templatetags.bbcode
common.templatetags.converters
common.templatetags.cycle_filter
common.templatetags.strings
common.templatetags.thumbnail
activitylog.middleware
activitylog.models
badges.models
profile.models
score.models
score.templatetags.lckd_score
tags.helpers
tags.models

Choices objects

This is a much clearer way to specify choices for fields in models and forms. Current documentation can be found at the dj.choices PyPI page.

Note: The legacy `lck.django.choices` namespace will be removed in `lck.django` 1.0.

Filters

The filters below are usable directly from pure Python code and obviously work as templatetags as well:

`nbsp`
`numberify`
`nullify`
`slugify`
`strike_empty`
`thumbnail`
`timediff`
`title`
`transliterate`

To use these filters in your source code, simply import them from `lck.django.filters`. To use them as templatetags, add the `lck.django.common` app to `INSTALLED_APPS` within your `settings.py` and in the specific template use:

```
{%load LIB_NAME%}
```

where `LIB_NAME` is the templatetag library name for the specific filter (available in the filter description). For instance, for `strike_empty` or `title` it would be:

```
{%load strings%}
```

There are more filters whose implementation makes them useful only as templatetags. These include:

`bbcode.bbcode`

Extensions for `settings.py`

One of the most often customized parts of a Django project is `settings.py`. To ease the process and protect against possible mistakes when doing the same exact modification for the n-th time, a set of extensions for `settings.py` was developed. The extensions are activated by importing them and executing:

```
from lck.django import current_dir_support
execfile(current_dir_support)
```

This is done so to enable the extensions access to the current local namespace of `settings.py`. If anyone knows a more elegant way to do that, let me know. For now this seems a good approach though.

current_dir_support

This extension injects a CURRENT_DIR variable into settings.py so it is available from the moment of definitions onwards. CURRENT_DIR in this context means the root of the project ("current" because most of the time this is the same dir where settings.py resides). An additional feature is that it is always ending with os.sep so it's perfectly safe to do something like:

```
from lck.django import current_dir_support
execfile(current_dir_support)

(...)

MEDIA_ROOT = CURRENT_DIR + 'media/'
```

Moreover, CURRENT_DIR gets set properly also for the projects that use a **package** for specifying settings, e.g. a structure like that:

```
project_dir
- __init__.py
- an_app
|   - __init__.py
|   - ...
- an_app
|   - __init__.py
|   - ...
- other_app
|   - __init__.py
|   - ...
- settings
|   - __init__.py
|   - local.py
- yet_other_app
    - __init__.py
    - ...
```

To enable this extension, import and execute it in the settings.py context (**at the very beginning** of the config file is recommended since CURRENT_DIR is available only after the extension's initialization).

namespace_package_support

This extension monkey-patches[1] Django so that is supports namespace packages. Not all places are covered though so if you spot some feature where namespace packages still don't work, file an issue using our issue tracker.

To enable this extension, import and execute it in the settings.py context (anywhere within the config file is fine):

```
from lck.django import namespace_package_support
execfile(namespace_package_support)
```

Features supported:

- custom commands for manage.py loaded correctly from apps within namespace packages; since 0.1.8

[1] Yup, in the world of Python that's considered dangerous and a sign of bad design. Here it's simply a sane way to overcome Django core development inertia. Go ahead and ask for namespace package support in vanilla Django.

profile_support

Splitting global settings from instance-specific ones is also a very frequent task that most Django projects implement in one way or another. There are many reasons why such separation is desirable but the most important ones are:

- if one global settings file is not enough to run the project, it forces the administrator deploying every instance to specify the local settings. This way no incompatible settings are used by default
- when global settings are kept separate, adding, deleting or editing entries doesn't cause conflicts while using source code version control systems[2]

So, how do you get to split your `settings.py` file? The simplest approach is to create a file with the local settings as a module and just import it at the end of the global one. This approach has two significant drawbacks: you cannot use/edit variables specified in the global settings and the name of the local module is now hard-coded within the global file.

Enter `profile_support`! With this extension you can have several local-specific settings modules which are executed in the global settings context. This means you can within your local settings do things like:

```
INSTALLED_APPS += (
  'debug_toolbar',
)

MEDIA_ROOT = CURRENT_DIR + 'static'
```

where you add the debug toolbar to the active apps (notice the += operator) and `CURRENT_DIR` is the one calculated by enabling `current_dir_support`. To enable profiles, just add these lines **at the very end** of your `settings.py` file:

```
from lck.django import profile_support
execfile(profile_support)
```

By default, this enables loading settings found in `settings-local.py`. There are times when you need more than one config profile though, for instance:

- you might want to have a very verbose debugging configuration for squashing the most persistent of bugs; most of the time however this kind of verbosity wouldn't be desirable)
- the live instance of your project is using a database user without database schema modification rights but you still want to be able to run `manage.py syncdb` and `manage.py migrate`
- you need to specify separate settings for your unit testing needs

Without `profile_support` you would create some "toggle" variables like `SYNC_DB` or `VERBOSE_DEBUG` and use `if/else` within the settings. Thanks to `profile_support` you can treat `settings.py` files like regular configuration files without any logic and just use different local profiles. To change to a profile different from *"local"* when running a command, just specify the `DJANGO_SETTINGS_PROFILE` environment variable:

```
DJANGO_SETTINGS_PROFILE=syncdb python manage.py migrate
```

In that case, the local settings will be loaded from `settings-syncdb.py` and not from `settings-local.py`.

If you use profiles heavily, the root project folder gets quite cluttered with `settings-*.py` files. In that case you might switch to package based configuration. Just make a directory called `settings`, move your existing `settings.py` to `settings/__init__.py` and your `settings-*.py` files to `settings/*.py`. Then your project tree will look something like the one on the diagram in the `current_dir_support` description above.

[2] May I kindly suggest Git or Mercurial?

Footnotes

Custom `manage.py` commands

By adding `lck.django.common` to your `INSTALLED_APPS` you get some additional second-level commands for `manage.py`:

- `shell`: a version of the original manage.py shell command for lazy people: is using bpython if installed and imports all models automatically

Note: These commands require *namespace_package_support*.

License

TODO

Roadmap

Version 0.9

Convert unit tests to base on unittest2. Create introductory material for common models, activitylog, score and tags: tutorials, examples and FAQ.

Version 1.0

Drop support for Django 1.3.

Separate some of the more useful things as their own packages. Currently these are already separated:

- dj.chain
- dj.choices

These would be great to separate:

- cache (needs updating first)
- namespace package support (so that all other dj.* packages work out of the box for users)
- thumbnail filter

Make all separated packages compatible with Python 3.x.

Unit test the rest, remove what's supported out of the box in Django 1.4.

Clean up forms.

Random ideas

- make the timing middleware store the results somewhere somehow
- custom inlines which set lck-specific readonly fields by default
- future date filter for the admin
- Migrate configuration of lck.django.cache to the 1.3+ variant
- Base *django-admin.py shell* on 1.4
- Deprecate execfile()-based settings hacking
- There are not enough unit tests
  - in particular, `py.test` is used for the existing tests which makes proper unit testing for Django-specific bits harder. Struggling with `django-pytest` did not produce any results as of yet.
- No examples in the code
- Bits documented only by means of API, some proper introduction would be handy:
  - forms
  - models
  - score
  - tags
- No FAQ, Tutorial

CHAPTER 4

Indices and tables

- genindex
- modindex
- search