hrCMS Documentation

Release 0

Michael Kuty & Ales Komarek

January 07, 2016

Contents

This is a stable for daily use in development.

A collection of awesome Django libraries, resources and shiny things. Full featured framework for building everything based on Django, FeinCMS, Horizon, Oscar and tons of another apps.

Don’t waste your time searching stable solution for daily problems.

CHAPTER 1

Django-Leonardo

Full featured platform for fast and easy building extensible web applications.

Don't waste your time searching stable solution for daily problems.

Deploy and Enjoy ! No skills required !

1.1 For Users

- CMS, Page, Responsive, Layouts, Themes, Color Variations
- Widgets, Plugins, 3rd party app integrations
- Frontend Edit, Install/Uninstall modules in one click !
- Store, Form Designer, Blog, News, Folio, Links, Navigations, ..
- Media, Folders, Files, Images, Documents, Import - Export, ..
- LIVE settings, Auto loading modules, editable templates, ..
- Authentification, 3rd party backends, SAML standard, ..

1.2 For Developers

- Python, Django, FeinCMS, OpenStack Horizon
- AngularJS, React, Bootstrap, Compress, Bootswatch
- Crispy forms, Floppy forms, Select2
- Filer, DbTemplates, Reversion, Constance
- Haystack, Oscar, Django Admin Bootstrap
- and tons of other apps bundled as modules

Further reading:

- Leonardo Site
- Demo Site (a reference build of an Leonardo project)
- Demo Store (a reference build of an Leonardo Store project)
- Developer Documentation (documentation for Django Developers)

- User Documentation (documentation for Leonardo end-users)
- Modules Leonardo modules
- Continuous integration homepage

1.3 Installation

one liner

Wget

```
wget -O - https://github.com/django-leonardo/django-leonardo/raw/develop/contrib/scripts/install_leon
```

Install Blog

```
wget -O - https://github.com/django-leonardo/django-leonardo/raw/develop/contrib/scripts/install_blog
```

Install Store

```
wget -O - https://github.com/django-leonardo/django-leonardo/raw/develop/contrib/scripts/install_stor
```

Python

```
python -c 'import urllib; print urllib.urlopen("https://github.com/django-leonardo/django-leonardo/ra
sudo sh install_leonardo.sh
```

Command by command

```
virtualenv -p /usr/bin/python2.7 leonardo_venv
cd leonardo_venv
. $PWD/bin/activate

pip install -e git+https://github.com/django-leonardo/django-leonardo@develop#egg=django-leonardo
pip install -r $PWD/src/django-leonardo/requirements.txt
django-admin startproject --template=https://github.com/django-leonardo/site-template/archive/master

export PYTHONPATH=$PWD/myproject
cd myproject

python manage.py makemigrations --noinput
python manage.py migrate --noinput
python manage.py bootstrap_site --url=http://raw.githubusercontent.com/django-leonardo/django-leonard

echo "from django.contrib.auth.models import User; User.objects.create_superuser('root', 'mail@leonar

python manage.py runserver 0.0.0.0:80
```

Navigate your browser to your_ip/admin and login with `root:admin` For settings production mode could take inspiration from Leonardo Documentation.

1.4 Core

Leonardo in default state has enabled some modules which provides basic stuff for common sites:

- Web - precise FeinCMS integration

- Navigation - common navigation components
- Media - Filer integration with media widgets
- Search - Haystack integration
- Auth - Basic auth with standard user actions
- Devel - Widgets for site developers

1.5 Installed

These modules are included in default leonardo installation, but could be uninstalled anytime without affecting your DB:

- System - Common management stuff (listing installed packages, widgets version etc..)
- Sitestarter - simple site starter which handle missing site and create it from custom yaml template
- Admin - django admin
- Ckeditor - Default wysiwyg editor for now. Alternatives: Redactor,Summernote,Tinymce..

For uninstalling:

```
pip uninstall leonardo-system
pip uninstall leonardo-sitestarter
```

For switch wysiwyg:

```
pip uninstall leonardo-ckeditor
pip install leonardo-redactor
```

1.6 Extensions

Leonardo in default state using module loader which allows you to easy installation of new packages.

All modules lives in Package Index.

1.6.1 Modules

- Admin - Standard Django admin
- Admin Honeypot - Django admin honeypot
- Bootstrap Admin - Bootstrap Django admin
- Material Admin - Material Django admin
- Analytics - Analytics service integration for Leonardo projects
- Blog - Elephant Blog integration
- Celery - Celery workers for Leonardo CMS
- Multisite - Multi site with security
- Folio - Portfolio app
- Geo - Some geolocation related widgets (Google maps,..)

- Galleries - Awesome galleries
- Forms - Stable Form Designer integration with Remote Forms
- Store - Oscar Ecommerce - FeinCMS integration
- News - Heavy FeinCMS News
- **'Newswall'_** - Newswall
- Links - navigation helpers bundles as Leonardo module
- Redactor - A lightweight wysiwyg editor for Leonardo
- Sentry - end-user friendly error page
- Page Permissions - extend Page and provide new Navigation templates with permissions
- Team - team model and widgets
- Oembed - oembed objects with caching
- Static - client-side libraries like an AngularJS, React, D3.js, ..

1.6.2 Themes

- Bootwatch
- AdminLTE

1.6.3 Cookiecutters

Our cookiecutters are a good start for creating new Site, Module or Theme.

- Site
- Module
- Theme

1.6.4 Bundles

Leonardo defines a group of bundles that can be used to install Leonardo and the dependencies for a given feature.

You can specify these in your requirements or on the `pip` comand-line by using brackets. Multiple bundles can be specified by separating them by commas.

```
$ pip install "django-leonardo[multisite]"

$ pip install "django-leonardo[blog,store,multisite]"
```

Here is current list of bundles.

1.7 Looking for commercial support?

If you are interested in having an Leonardo project built for you, or for development of an existing Leonardo site. Please get in touch via mail@majklk.cz.

1.8 Read More

- http://docs.openstack.org/developer/horizon/quickstart.html
- http://feincms-django-cms.readthedocs.org/en/latest/index.html
- https://django-oscar.readthedocs.org/en/releases-1.0/

Installation

2.1 Installation

Installation Leonardo depend's on your case. See some examples

Via PIP

```
pip install django-leonardo

# or latest

pip install git+https://github.com/django-leonardo/django-leonardo@develop#egg=leonardo
```

WGET one-liner

```
wget -O install_leonardo.sh https://github.com/django-leonardo/django-leonardo/raw/develop/contrib/sc
```

Command by command

```
virtualenv -p /usr/bin/python2.7 leonardo_venv
cd leonardo_venv
. $PWD/bin/activate

pip install -e git+https://github.com/django-leonardo/django-leonardo@develop#egg=django-leonardo
pip install -r $PWD/src/django-leonardo/requirements.txt
django-admin startproject --template=https://github.com/django-leonardo/site-template/archive/master

export PYTHONPATH=$PWD/myproject
cd myproject

python manage.py makemigrations --noinput
python manage.py migrate --noinput
python manage.py bootstrap_site --url=http://raw.githubusercontent.com/django-leonardo/django-leonar

echo "from django.contrib.auth.models import User; User.objects.create_superuser('root', 'mail@leonar

python manage.py runserver 0.0.0.0:80
```

Using salt

With configured Salt use our Formula writte your pillars and run

```
salt-call state.sls leonardo
```

2.1.1 Bundles

Leonardo defines a group of bundles that can be used to install Leonardo and the dependencies for a given feature.

You can specify these in your requirements or on the `pip` comand-line by using brackets. Multiple bundles can be specified by separating them by commas.

For all Leonardo modules continue to https://github.com/leonardo-modules

```
$ pip install "django-leonardo[web]"

$ pip install "django-leonardo[web,nav,media,eshop]"
```

The following bundles are available:

CMS

- django-leonardo[blog] - ElephantBlog integration
- django-leonardo[folio] - Portfolio with translations
- django-leonardo[multisite] - Leonardo multi sites
- django-leonardo[forms] - Form Designer and Remote Forms
- django-leonardo[links] - Links
- django-leonardo[pagepermissions] - Page Permissions

Background Jobs

- django-leonardo[celery] - Celery Workers for background Jobs

Admin

- django-leonardo[admin] - Django Admin for Leonardo CMS

Auth

- django-leonardo[auth] - All auth
- django-leonardo[saml] - SAML auth backend

WYSIWYG Editors

- django-leonardo[redactor] - Redactor
- django-leonardo[summernote] - SummerNote

Themes

- django-leonardo[themes] - Leonardo themes [Bootstrap, AdminLTE]
- django-leonardo[adminlte] - AdminLTE theme

Ecommerce

- django-leonardo[store] - Django-Oscar integration
- django-leonardo[stores] - Django-Oscar Stores
- django-leonardo[cod] - Django-Oscar Cash On Delivery Payment Method

Common

- django-leonardo[sentry] - Raven integration with end-user friendly error page
- django-leonardo[static] - AngularJS, React, BootStrap, D3.js, ..
- django-leonardo[debug] - Debug toolbar
- django-leonardo[tests] - Tools for testing
- django-leonardo[redis] - Redis dep

2.2 Development Environment

Simplest way is using our SaltStack Formula where you can comfortably specify sources for leonardo with plugins.

2.2.1 Prerequisites

Prerequisites are installed on Ubuntu 12.04 LTS or Raspian Wheezy 7.0 with:

```
$ sudo apt-get install python-pip python-dev python-setuptools git python-virtualenv libtiff5-dev lib
    libfreetype6-dev liblcms2-dev libwebp-dev tcl8.6-dev tk8.6-dev python-tk
```

Prerequisites are installed on Ubuntu 14.04 LTS with:

```
$ sudo apt-get install python-pip python-dev python-setuptools git python-virtualenv libtiff5-dev lib
    libfreetype6-dev liblcms2-dev libwebp-dev tcl8.6-dev tk8.6-dev python-tk
```

Prerequisites are installed on Fedora 20 with:

```
$ sudo yum install python-pip python-dev python-setuptools git python-virtualenv libtiff-devel libjpe
    lcms2-devel libwebp-devel tcl-devel tk-devel
```

Note: If you have problems with installation, please see SaltStack Formula where are all steps of installation.

usualy after successfuly installated prerequisites you can start with Leonardo

```
virtualenv -p /usr/bin/python2.7 /srv/leonardo/sites/mysite

git clone https://github.com/django-leonardo/django-leonardo.git -b develop /srv/leonardo/sites/mysit

vim /srv/leonardo/sites/mysite/local_settings.py
```

put your config to `local_settings.py`:

```
# -*- coding: utf-8 -*-

from __future__ import absolute_import

import sys
from os.path import join, dirname, abspath, normpath

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'HOST': '127.0.0.1',
        'NAME': 'leonardo_mysite',
        'PASSWORD': 'mysite_password',
        'USER': 'leonardo_mysite'
    }
}

CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
        'TIMEOUT': 120,
        'KEY_PREFIX': 'CACHE_MYSITE'
    }
}

SECRET_KEY = 'my_secret_key'

DEBUG = True

MEDIA_ROOT = '/srv/leonardo/sites/mysite/media/'
STATIC_ROOT = '/srv/leonardo/sites/mysite/static/'

TIME_ZONE = 'Europe/Prague'

LANGUAGE_CODE = 'en'

# your app here
APPS = [
    'blog',
    'forms',
    'news'
]

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'root': {
        'level': 'WARNING',
        'handlers': ['file'],
    },
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        }
    },
    'formatters': {
        'verbose': {
            'format' : "[%(asctime)s] %(levelname)s [%(name)s:%(lineno)s] %(message)s",
            'datefmt' : "%d/%b/%Y %H:%M:%S"
        },
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
    },
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'verbose'
        },
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': '/srv/leonardo/sites/mysite/leonardo_server.log',
            'formatter': 'verbose'
        },
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler',
            'formatter': 'simple',
        },
    },
    'loggers': {
        'django.request': {
            'handlers': ['mail_admins', 'file'],
            'level': 'DEBUG',
            'propagate': True,
        },
    }
}
```

Run

There are several options which you can use, see contrib directory in the repo https://github.com/django-leonardo/django-leonardo/tree/master/contrib

Django `runserver`

```
python /srv/leonardo/sites/mysite/leonardo/contrib/django/manage.py runserver 0.0.0.0:80
```

Tornado

```
python /srv/leonardo/sites/mysite/leonardo/contrib/tornado/server
```

2.3 Leonardo in Production

Leonardo in production is standard Django application. We use Gunicorn under Supervisor with Nginx proxy.

2.3.1 Supervisor

```
[program:leonardo_demo]
command=/srv/leonardo/sites/demo/leonardo/contrib/gunicorn/server
stdout_logfile=/srv/leonardo/sites/demo/logs/access.log
stderr_logfile=/srv/leonardo/sites/demo/error.log
user=leonardo
autostart=true
autorestart=true
```

2.3.2 Gunicorn

```
#!/bin/bash

NAME="leonardo_demo"
DJANGODIR=/srv/leonardo/sites/demo
USER=leonardo
GROUP=leonardo
NUM_WORKERS=3
DJANGO_SETTINGS_MODULE=leonardo.settings
DJANGO_WSGI_MODULE=wsgi

echo "Starting $NAME as `whoami`"

# Activate the virtual environment
cd $DJANGODIR
source /srv/leonardo/sites/demo/bin/activate
export DJANGO_SETTINGS_MODULE=$DJANGO_SETTINGS_MODULE
export PYTHONPATH=$DJANGODIR:$PYTHONPATH

# Start your Django Unicorn
# Programs meant to be run under supervisor should not daemonize themselves (do not use --daemon)
exec gunicorn ${DJANGO_WSGI_MODULE}:application \
  --name $NAME \
  --workers $NUM_WORKERS \
  --user=$USER --group=$GROUP \
  --log-level=debug \
  --bind=0.0.0.0:9754
```

for Tornado see Github page

2.3.3 Nginx

```
upstream leonardo_server_leonardo_demo {
    server localhost:9754 fail_timeout=0;
}

server {
  listen 80;

  server_name demo.cms.robotice.cz;

  client_max_body_size 20M;

  access_log  /var/log/nginx/demo-access;
  error_log   /var/log/nginx/demo-error;

  keepalive_timeout 5;

gzip on;
gzip_min_length  1100;
gzip_buffers  4 32k;
gzip_types    text/plain application/x-javascript text/xml text/css;
gzip_vary on;

  location / {
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_redirect off;

    if (!-f $request_filename) {
      proxy_pass http://leonardo_server_leonardo_demo;
      break;
    }
  }

  location /static {
    autoindex on;
    alias /srv/leonardo/sites/demo/static;
    expires    30d;
 }

  location /media {
    autoindex on;
    alias /srv/leonardo/sites/demo/media;
    expires    30d;
  }

}
```

2.4 Bootstrap site

We don't repeat yourself and for really quick start with new site we provide simple API called Bootstrap which has simple format in `yaml` or `json` and may have contains basic stuff for your site:

```
auth.User:
  admin:
    password: root
    mail: root@admin.cz
web.Page:
  QuickStart:
    title: Quickstart
    slug: quickstart
    override_url: /
    featured: false
    theme: __first__
    in_navigation: true
    active: true
    color_scheme: __first__
    content:
      header:
        web.SiteHeadingWidget:
          attrs:
            site_title: Leonardo Site
            content_theme: navbar
            base_theme: default
          dimenssions:
            md: 2
        web.TreeNavigationWidget:
          attrs:
            depth: 2
            content_theme: navbar
            base_theme: default
          dimenssions:
            md: 6
        web.UserLoginWidget:
          attrs:
            inline: true
            type: 2
            content_theme: navbar
            base_theme: default
          dimenssions:
            md: 4
elephantblog.Entry:
  Test:
    title: Test
    slug: test
    author:
      type: auth.User
      pk: 1
    content:
      main:
        elephantblog.HtmlTextWidget:
          attrs:
            text: Hello world !
            content_theme: default
            base_theme: default
          dimenssions:
            md: 2
```

From local source

```
python manage.py bootstrap_site --name=demo.yaml
```

This mechanismus is really simple without any magic features. Just define your model entyties with some parameters. For FeinCMS models is there field called `content` which is dictionary of content regions like `col3` with some `Widgets`.

From remote host

```
python manage.py bootstrap_site --url=http://raw.githubusercontent.com/django-leonardo/django-leonar
```

2.5 Salt Installation

if your infrastucutre is managed by Salt use and contribute to our Salt Formula

https://github.com/django-leonardo/salt-formula-leonardo

2.5.1 Sample pillar

```
leonardo:
  server:
    enabled: true
    app:
      example_app:
            site_name: 'My awesome site'
        enabled: true
        development: true
        workers: 3
        bind:
          address: 0.0.0.0
          port: 9754
          protocol: tcp
        source:
          type: 'git'
          address: 'git@repo1.robotice.cz:python-apps/leonardo.git'
          rev: 'master'
        secret_key: 'y5m^_^ak6+5(f.m^_^ak6+5(f.m^_^ak6+5(f.'
        database:
          engine: 'postgresql'
          host: '127.0.0.1'
          name: 'leonardo'
          password: 'db-pwd'
          user: 'leonardo'
        mail:
          host: 'mail.domain.com'
          password: 'mail-pwd'
          user: 'mail-user'
        admins:
          mail@majklk.cz:
            name: majklk
          mail@newt.cz: {}
        managers:
          mail@majklk.cz:
            name: majklk
          mail@newt.cz:
            name: newt
        plugin:
          eshop: {}
          static: {}
          sentry: {}
          my_site:
            site: true
          blog:
            source:
              engine: 'git'
              address: 'git+https://github.com/django-leonardo/leonardo-module-blog.git#egg=leonardo_
        languages:
          en:
            default: true
          cs: {}
          de: {}
```

CHAPTER 3

Configuration

3.1 Leonardo Configuration

Leonardo is Django powered. All important settings is related with standard Django settings, but is there some leonardo specific configuration.

3.1.1 Configure files

- `local_settings` in your `PYTHONPATH` for all stuff
- or `your_site/local/settings.py`

Note: `leonardo_site` must be in the `PYTHONPATH`

```
SITE_ID = 1
SITE_NAME = 'leonardo'
# or full domain
SITE_DOMAIN = 'www.leonardo.cz'

LANGUAGE_CODE = 'en'

RAVEN_CONFIG = {}
```

Note: Leonardo finds and includes all modules which has descriptor(leonardo modules).

if you want you can specify your custom APPS:

```
APPS = [
    'blog',
    'eshop',
    'leonardo_site',   # our app
]
```

Leonardo template https://github.com/django-leonardo/site-template

```
manage.py makemigrations --noinput
manage.py migrate --noinput
manage.py sync_all
```

3.1.2 Change admin site name

```
SITE_HEADER = "Leonardo administration"

SITE_TITLE = "Leonardo site admin"
```

3.1.3 Apps, modules, themes ..

Leonardo has own specific app/module system. This system is same as Django, but provide some improvements, which makes time for installing and configuring new app shorter

```
APPS = ['leonardo']

# is same as

INSTALLED_APPS = ['leonardo']
```

But if configured via `APPS`, Leonardo tryies find `default` configuration in main descriptor of module. Descriptor may contains many various properties, which is safely merge into main settings. For full description see `modules`.

3.1.4 Leonardo

```
LEONARDO_MODULE_AUTO_INCLUDE = True
```

This option says please do not auto include leonardo modules.

```
LEONARDO_MEMOIZED = True
```

If set False is disabled any content cache.

For disable System Module which provide untested and unsecure features.

```
LEONARDO_SYSTEM_MODULE = True
```

3.1.5 Frontend Edit

```
LEONARDO_FRONTEND_EDITING = True
```

3.1.6 Horizon

Horizon has own `urls` finder, which provide capabilities for defining `dashboards`, `panels`.. in default state is included in main leonardo's urls, but you can turn off, but you must map external app to any `Page` which provide `horizon` namespace.

```
HORIZON_ENABLED = False
```

Note: Before this, please add external app `Horizon` to any `Page`, because may broke admin.

3.2 Live configuration

Live configuration is based on django-constance with few improvements.

- basic grouping via `CONSTANCE_CONFIG_GROUPS` which makes tabs for django admin
- access to config keys from standard django settings
- really live settings, set every value to django settings and respect the default value from them

```
class Default(object):

    optgroup = 'GA'

    apps = [
        'analytical',
        'leonardo_module_analytics',
    ]

    config = {
        'GOOGLE_ANALYTICS_PROPERTY_ID': ('xx-xxx-x', _('Google Site identificator')),
        'GOOGLE_ANALYTICS_SITE_SPEED': (False, _('analyze page speed')),
        'GOOGLE_ANALYTICS_ANONYMIZE_IP': (False, _('anonymize ip')),
    }

default = Default()
```

Note: Please be sure about keys in config, all is merged into one big dictionary which is used. Last wins.

All config keys will be namespaced under GA and available from standard django settings:

```
from django.conf import settings

settings.GOOGLE_ANALYTICS_PROPERTY_ID
-> xx-xxx-x
```

Warning: This may not work on some special environments like a sqlite. For stable usage in modules we recommend using via `constance` see below.

3.2.1 Backends

In default state is used database backend but redis is recommended.

Redis

installing via leonardo extras

```
pip install django-leonardo[redis]
```

set your configuration

```
APPS = ['constance.backends.redisd']

CONSTANCE_BACKEND = 'constance.backends.redisd.RedisBackend'

CONSTANCE_REDIS_CONNECTION = {
    'host': 'localhost',
    'port': 6379,
    'db': 0,
}
```

optionaly

```
CONSTANCE_REDIS_PREFIX = 'constance:myproject:'
```

for more configuration choices visit http://django-constance.readthedocs.org/en/latest/backends.html

3.2.2 Using in your code

```
from django.conf import settings

if settings.THE_ANSWER == 42:
    answer_the_question()
```

of via constance module

```
from constance import config

if config.THE_ANSWER == 42:
    answer_the_question()
```

3.2.3 Custom Fields

Sometimes is better to use custom choices fields or someting like this for this purposes could be used `LEONARDO_ADDITIONAL_FIELDS`:

LEONARDO_CONFIG = {

> **'MULTISITE_ENABLED': (False, _(** 'Enable multi site request processing')),
>
> **'SESSION_COOKIE_DOMAIN': ('', _(** ''If you set your session cookie domain to start with a "." character it will let you handle wildcard sub-domains and share a session cookie (login session) across multiple subdomains.'')),
>
> 'MY_SELECT_KEY': ('yes', 'select yes or no', 'yes_no_null_select'),

} LEONARDO_ADDITIONAL_FIELDS = {

> **'yes_no_null_select': ['django.forms.fields.ChoiceField',**
>
> > **{** 'widget': 'django.forms.Select', 'choices': (("—-", None), ("yes", "Yes"), ("no", "No"))
> >
> > }],

}

3.3 Storage / Media

In this time we have good integration of Django-Filer, which provides good base for us.

We support for standard scenarious:

- make, upload, delete, move folders and files
- import files (scan) into concrete folder via admin or command
- basic media entities

3.3.1 Configuation

put your configuation into your `local_settings.py`, these is defaults, you can also update only one concrete field, but you must import default from `leonardo.module.media.settings`

```
FILER_ENABLE_PERMISSIONS = True

FILER_STORAGES = {
    'public': {
        'main': {
            'ENGINE': 'filer.storage.PublicFileSystemStorage',
            'OPTIONS': {
                'location': '/path/to/media/filer',
                'base_url': '/smedia/filer/',
            },
            'UPLOAD_TO': 'filer.utils.generate_filename.randomized',
            'UPLOAD_TO_PREFIX': 'filer_public',
        },
        'thumbnails': {
            'ENGINE': 'filer.storage.PublicFileSystemStorage',
            'OPTIONS': {
                'location': '/path/to/media/filer_thumbnails',
                'base_url': '/smedia/filer_thumbnails/',
            },
        },
    },
    'private': {
        'main': {
            'ENGINE': 'filer.storage.PrivateFileSystemStorage',
            'OPTIONS': {
                'location': '/path/to/smedia/filer',
                'base_url': '/smedia/filer/',
            },
            'UPLOAD_TO': 'filer.utils.generate_filename.randomized',
            'UPLOAD_TO_PREFIX': 'filer_public',
        },
        'thumbnails': {
            'ENGINE': 'filer.storage.PrivateFileSystemStorage',
            'OPTIONS': {
                'location': '/path/to/smedia/filer_thumbnails',
                'base_url': '/smedia/filer_thumbnails/',
            },
        },
    },
}
```

3.3.2 Imports

```
manage.py import_files --path=/tmp/assets/images
manage.py import_files --path=/tmp/assets/news --folder=images
```

Note: via admin we support only relative(`MEDIA_ROOT`) scan

3.4 Migrations

Leonardo itself does not come with any migrations. It does not have to: Its core models haven't changed for several versions now. This does not mean migrations aren't supported. You are free to use either Django's builtin migrations support, or also South if you're stuck with Django versions older than 1.7.

3.4.1 Django's builtin migrations

- Create a new folder in your app with an empty `__init__.py` inside.
- Add the following configuration to your `settings.py`:

```
MIGRATION_MODULES = {
    'web': 'leonardo_site.migrations',
}
```

```
python manage.py makemigrations --noinput

python manage.py migrate --noinput
```

If you have database already created, redirect your migration and create empty migrations

add this to your `settings.py`

```
MIGRATION_MODULES = {
    'web': 'leonardo_site.migrations',
}
```

create empty migrations to new path

```
python manage.py makemigrations --empty web
```

For big apps we recommend separation of migrations per module, like this:

```
MIGRATION_MODULES = {
    'web': 'leonardo_site.migrations.web',
}
```

If you changed `LANGUAGES` Django check new migrations, which changed choices on translation of media models. For these purposes we recommend redirect affected apps:

```
MIGRATION_MODULES = {
    'web': 'leonardo_site.migrations.web',
    'media': 'leonardo_site.migrations.media',
}
```

Note: Don't forget to create corresponding directories.

You can also redirect migrations from any leonardo module. Just use `MIGRATIONS_MODULES` in the module descriptor something like this:

```
LEONARDO_MIGRATION_MODULES = {
    'web': 'my_module.migrations.web',
    'media': 'my_module.migrations.media',
}
```

With this, leonardo supports changing default location `leonardo_site` as project module.

3.5 Languages

For settings Langugages follow standard Django settings like this:

```
LANGUAGE_CODE = 'en'

LANGUAGES = (
    ('en', 'EN'),
    ('cs', 'CS'),
)
```

This is default settings which specify English as default Language

Note: location of these settings is `local_settings.py` or your Site settings file

for switching to Czech as default redefine:

```
LANGUAGE_CODE = 'cs'

LANGUAGES = (
    ('cs', 'CS'),
    ('en', 'EN'),
)
```

Warning: Ordering in the `LANGUAGES` is important for translations ! First must be default language.

Leonardo provides one management command for making messages and theirs compiling:

```
python manage.py update_translations
```

3.6 Search

Leonardo Search is only Haystack integration, which provide robust solution for this domain.

In default state Leonaro use this configuration

```
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'haystack.backends.whoosh_backend.WhooshEngine',
        'PATH': os.path.join(os.path.dirname(__file__), 'whoosh_index'),
    },
}
```

For other backends visit

- Sorl - http://django-haystack.readthedocs.org/en/latest/tutorial.html#solr
- ElasticSearch - http://django-haystack.readthedocs.org/en/latest/tutorial.html#elasticsearch

- Xapian - http://django-haystack.readthedocs.org/en/latest/tutorial.html#xapian

Warning: Don't forget rebuild indexes !

```
python manage.py rebuild_index --noinput
```

For other commands see doc or help.

Enjoy !

Themes, Templates and Color variations

3.7 Sync Themes

Sync widget themes

```
python manage.py sync_all
```

replace db from files (new version of core template etc..)

```
python manage.py sync_all -f
```

command by command

Run collectstatic

```
python manage.py collectstatic --noinput
```

After collectstatic create page themes

```
python manage.py sync_page_themes
```

load widget themes

```
python manage.py sync_widget_themes
```

3.8 Favicons

For enabling your favicon copy your favicon to `static/img/favicon.ico` anywhere in your installed app.

Extend

4.1 New Site

4.1.1 Django Template

Easiest way how you can create new Leonardo Site is our Django Site Template which lives here

https://github.com/django-leonardo/site-template

If you have installed Leonardo

```
django-admin startproject --template=https://github.com/django-leonardo/site-template/archive/master

cd myproject
```

- `local_settings` in your `PYTHONPATH` for all stuff
- or `settings/menu` .. in `conf`

Note: `leonardo_site` must be in the `PYTHONPATH`, you could use `pip install git+url.git#egg=leonardo_site` format

```
SITE_ID = 1
SITE_NAME = 'leonardo'
# or full domain
SITE_DOMAIN = 'www.leonardo.cz'

LANGUAGE_CODE = 'en'

RAVEN_CONFIG = {}

APPS = [
    'blog',
    'leonardo_site',  # our app
]
```

Leonardo template https://github.com/django-leonardo/site-template

If you have configured your database and other common stuff run

```
manage.py makemigrations --noinput
manage.py migrate --noinput
manage.py sync_all
```

4.1.2 Cookiecutter

```
pip install cookiecutter
git clone https://github.com/django-leonardo/cookiecutter-site.git cookiecutter-leonardo-site
cookiecutter cookiecutter-leonardo-site
project_name [leonardo-site]:
enter
repo_name [leonardo_site]:
enter
done.

export PYTHONPATH=/path/to/leonardo-site

manage.py makemigrations --noinput
manage.py migrate --noinput
manage.py sync_all
```

4.2 New Theme

Best example is live code, we have two base themes for you and lives under main github group

- AdminLTE - https://github.com/django-leonardo/leonardo-theme-adminlte
- Bootswatch - https://github.com/django-leonardo/leonardo-theme-bootswatch

As you can see theme must contains one template for page layout and optionaly base css for this layout and some color variations lives in `skins` directory.

Directory structure:

```
leonardo_theme_bootswatch
    |-- __init__.py
    |-- templates
        |-- base
            |-- page
                |-- bootswatch.html
    |-- static
        |-- themes
            |-- bootswatch
                |-- _variables.scss
                |-- cosmo
                    |-- _variables.scss
                    |-- _styles.scss
                    |-- scheme.scss
```

Required stuff for color sheme is `scheme.scss` which may contains something like this:

```
@import "_variables";
@import "../_styles";
@import "_styles";
```

Warning: Every skin must have `_variables` file which is dynamically appended to every widget scss file.

If we run

```
python manage.py sync_all
```

or any his variations Leonardo load base page templates into database and after this step tries find css in theme location. After that we have ready theme for our pages and also for editing via admin interface.

Leonardo automatically load these tested themes if is present. For their installation write this in your environment

For all supported themes simple do

```
pip install django-leonardo[themes]

python manage.py sync_all -f
```

sync themes

```
python manage.py sync_all -f
```

Solo AdminLTE

```
pip install leonardo_theme_adminlte

# or via main package

pip install django-leonardo[adminlte]
```

Note: Don't remmeber sync themes, which is described in the `web/themes`

For new theme is situation more complex. You have two options:

- create your Leonardo module descriptor and put your theme name into main `APPS`
- pur your theme name directly into main `INSTALLED_APPS`, but this is more hard way

For first option you must write simple leonardo module descriptor in `my_new_theme_name/__init__.py`

```
class Default(object):

    # define your specific apps
    apps = ['my_new_theme_name']

default = Default()
```

and add it to APPS in `local_settings.py`

```
APPS = [
    'my_new_theme_name'
]
```

That's it. Run `sync_all`.

New scaffold theme with using cookiecutter

```
pip install cookiecutter
git clone https://github.com/django-leonardo/cookiecutter-site.git cookiecutter-leonardo-site
cookiecutter cookiecutter-leonardo-site
```

4.3 Leonardo Module

Leonardo module is standard Django application with many additional posibilities. In Leonardo module just type your needs and develop your application.

```
pip install cookiecutter
git clone https://github.com/django-leonardo/cookiecutter-module.git cookiecutter-leonardo-module
cookiecutter cookiecutter-leonardo-module
```

start module directory structure:

```
leonardo_module_blog
    |-- __init__.py
    |-- settings.py
```

4.3.1 Application

As Django documentations says, you can define your apps in `apps.py` or anywhere, in Leonardo we use __init__.py for simplicity. But you can define it where you want.

Redirect configurations to any other location, like:

```
# Django stuff
default_app_config = 'leonardo_module_blog.apps.BlogConfig'
leonardo_module_conf = 'leonardo_module_blog.apps'
```

Required setup for Django Application are defined `AppConfig` which has basic attributes described in Django Documentation. For Leonardo Module are required one options with prefix `LEONARDO` or `default` attribute which specify Leonardo Module stuff.

```
from django.apps import AppConfig

default_app_config = 'leonardo_module_blog.BlogConfig'

class Default(object):

    optgroup = 'Blog'

    apps = [
        'leonardo_module_blog',
        'elephantblog',
        'leonardo_module_analytics',
    ]

    js_files = [
        'js/redactor.js'
    ]

    css_files = [
        'css/redactor.css'
    ]

    config = {
        'BLOG_PAGINATE_BY': (10, _('Blog Entries Pagination')),
        'DISQUS_COMMENTS': (False, _('Enable Disqus comments')),
        'DISQUS_SHORTNAME': ('michaelkuty', _('Disqus shortname identificator.')),

    }

    navigation_extensions = [
        'elephantblog.navigation_extensions.treeinfo',
    ]
    absolute_url_overrides = {
        'elephantblog.entry': 'leonardo_store.overrides.elephantblog_entry_url_app',
        'elephantblog.categorytranslation':
        'leonardo_store.overrides.elephantblog_categorytranslation_url_app',
    }


# standard django Application
class BlogConfig(AppConfig, Default):
    name = 'leonardo_module_blog'
    verbose_name = ("Blog")

default = Default()  # define module configuration
```

That's all.. Leonardo go throught every module defined in your `APPS` and merge all items to main settings file. Complete reference you can see below.

Note: Leonardo supports two syntax. One Pythonic way which is described upstair. `default` attribute which respect simple Python Object.

For some users are Python way unnecessarily complicated, for this people leonardo supports another config syntax:

```
LEONARDO_APPS = ['app1']

LEONARDO_ABSOLUTE_URL_OVERRIDES = {
    'elephantblog.entry': 'leonardo_store.overrides.elephantblog_entry_url_app',
    'elephantblog.categorytranslation':
    'leonardo_store.overrides.elephantblog_categorytranslation_url_app',
}
```

Note: Just use same keys with prefix and uppercase `LEONARDO_`

Tip: For all possibility settings keys see Module Reference

4.3.2 Settings

in the settings you may have something like this

```
BLOG_TITLE = 'name'

# whatever
```

As you expext every key from settings will be inported and merged into main settings file.

Warning: Be careful if you declare keys in the `module/settings.py`. Every key is imported without special merging process which may override your global settings ! It was designed only for module/app specific defaults.

4.3.3 Release

For releasing big amount of pip packages we use `PBR` which was developed for OpenStack and we have tunned version which lives here https://github.com/michaelkuty/pbr.

PBR can and does do a bunch of things for you:

- **Version**: Manage version number based on git revisions and tags
- **AUTHORS**: Generate AUTHORS file from git log
- **ChangeLog**: Generate ChangeLog from git log
- **Sphinx Autodoc**: Generate autodoc stub files for your whole module
- **Requirements**: Store your dependencies in a pip requirements file (install from vcs)
- **long_description**: Use your README file as a long_description
- **Smart find_packages**: Smartly find packages under your root package

With this tool is managing python module pretty simple. Add these lines to your `setup.py`:

```
import setuptools

# In python < 2.7.4, a lazy loading of package `pbr` will break
# setuptools if some other modules registered functions in `atexit`.
# solution from: http://bugs.python.org/issue15881#msg170215
try:
    import multiprocessing  # noqa
except ImportError:
    pass

setuptools.setup(
    setup_requires=['pbr'],
    pbr=True)
```

and write meta to `setup.cfg`

```
[metadata]
name = leonardo-team
summary = Team Application for Leonardo CMS or plain FeinCMS
description-file =
    README.rst
author = Michael Kuty
author-email = kutymichael@gmail.com
home-page = https://github.com/leonardo-modules/leonardo-team.git
classifier =
    Development Status :: 5 - Production/Stable
    Framework :: Django
    Intended Audience :: Developers
    License :: OSI Approved :: BSD License
    Operating System :: OS Independent
    Programming Language :: Python
    Programming Language :: Python :: 2.6
    Programming Language :: Python :: 2.7
    Programming Language :: Python :: 3
    Programming Language :: Python :: 3.3
    Programming Language :: Python :: 3.4
    Topic :: Software Development
    Topic :: Software Development :: Libraries :: Application Frameworks

[files]
packages =
    team
```

and run

```
python setup.py sdist register
```

PBR is GIT driven if you want add new version for release just create new tag like:

```
git tag v1.4
```

and then upload new release to pip:

```
python setup.py sdist upload
```

Note: Full documnetation of PBR lives there http://docs.openstack.org/developer/pbr/

4.4 Leonardo Descriptor Reference

Descriptor is Leonardo specific and is inspired from Openstack Horizon where is used for non invasive extend Dashboard extends. In the Leonardo we use same pattern, but with some additions.

Directory structure:

```
my_awesome_module
    |-- __init__.py
    |-- settings.py
    |-- urls.py
```

Warning: Leonardo include all settings and urls in root of module.

4.4.1 Descriptor reference

Leonardo

apps - leonardo modules or whatever:

```
apps = [
    'leonardo_module_blog',
    'elephantblog',
    'leonardo_module_analytics',
]
```

urls_conf url path to include

public if is set to `True` Leonardo does not decorate included url patters for required authentification

module_actions array of templates included in the frontend side bar

additional_fields dictionary of custom fields which could be used in config:

```
LEONARDO_ADDITIONAL_FIELDS = {
    'yes_no_null_select': ['django.forms.fields.ChoiceField',
                           {
                               'widget': 'django.forms.Select',
                               'choices': (("-----", None), ("yes", "Yes"), ("no", "No"))
                           }],
}
```

```
LEONARDO_CONFIG = {
    'MULTISITE_ENABLED': (False, _(
        'Enable multi site request processing')),
    'SESSION_COOKIE_DOMAIN': ('', _(
        '''If you set your session cookie domain to start with
        a "." character it will let you handle wildcard sub-domains
        and share a session cookie (login session) across multiple
        subdomains.''')),
    'MY_SELECT_KEY': ('yes', 'select yes or no', 'yes_no_null_select'),
}
```

FeinCMS

widgets - FeinCMS widgets:

```
widgets = [
        BlogCategoriesWidget,
        RecentBlogPostsWidget,
    ]
```

optgroup - menu group name for widgets:

```
optgroup = 'Blog'
```

plugins - FeinCMS 3rd party apps support:

```
plugins = [
    ('elephantblog.urls', 'Blog entries'),
]
```

page_extensions - FeinCMS page extensions

navigation_extensions - FeinCMS Page Extensions - will be imported before reofistering for proper load:

```
navigation_extensions = [
    'elephantblog.navigation_extensions.treeinfo',
]
```

Horizon

js_files - merged and added to main page footer:

```
js_files = [
    'js/redactor.js'
]
```

js_compress_files - already compressed static files:

```
js_files = [
    'js/redactor.min.js'
]
```

css_files linked in head as style:

```
css_files = [
    'css/redactor.css'
]
```

scss_files linked in head as scss style:

```
scss_files = [
    'scss/redactor.scss'
]
```

angular_modules Angular modules which will be loaded:

```
angular_modules = [
    'angular-carousel'
]
```

js_spec_files - Angular specific see https://github.com/openstack/horizon/blob/master/openstack_dashboard/enabled/_10_project.

Constance

config - dictionary of keys for `django-constance`:

```
config = {
    'BLOG_PAGINATE_BY': (10, _('Blog Entries Pagination')),
    'DISQUS_COMMENTS': (False, _('Enable Disqus comments')),
    'DISQUS_SHORTNAME': ('michaelkuty', _('Disqus shortname identificator.')),

}
```

Django

auth_backends - AUTHENTICATION_BACKENDS:

```
auth_backends = [
    'oscar.apps.customer.auth_backends.EmailBackend'
]
```

context_processors - Django Context Processors:

```
context_processors = [
    ...
    'oscar.apps.checkout.context_processors.checkout',
    'oscar.apps.customer.notifications.context_processors.notifications',
    ...
]
```

middlewares - Django Middlewares:

```
middlewares = [
    'oscar.apps.basket.middleware.BasketMiddleware',
]
```

migration_modules - allow override migration's location:

```
migration_modules = {
    'elephantblog': 'leonardo_module_blog.migrations',
}
```

absolute_url_overrides - model name and method wich would be imported for easy integrating 3rd party app:

```
    absolute_url_overrides = {
        'elephantblog.entry': 'leonardo_store.overrides.elephantblog_entry_url_app',
    }
```

4.4.2 Minimal example

your app directory structure:

```
leonardo_module_blog
    |-- __init__.py
    |-- settings.py
```

__init__.py

As Django documentations says, you can define your apps in `apps.py` or anywhere, in Leonardo we use __init__.py for simplicity. But you can define it where you want.

```
from django.apps import AppConfig

default_app_config = 'leonardo_module_blog.BlogConfig'

class Default(object):

    optgroup = 'Blog'

    apps = [
        'leonardo_module_blog',
        'elephantblog',
        'leonardo_module_analytics',
    ]

    js_files = [
        'js/redactor.js'
    ]

    css_files = [
        'css/redactor.css'
    ]

    config = {
        'BLOG_PAGINATE_BY': (10, _('Blog Entries Pagination')),
        'DISQUS_COMMENTS': (False, _('Enable Disqus comments')),
        'DISQUS_SHORTNAME': ('michaelkuty', _('Disqus shortname identificator.')),

    }

    navigation_extensions = [
        'elephantblog.navigation_extensions.treeinfo',
    ]

    absolute_url_overrides = {
        'elephantblog.entry': 'leonardo_store.overrides.elephantblog_entry_url_app',
        'elephantblog.categorytranslation':
        'leonardo_store.overrides.elephantblog_categorytranslation_url_app',
    }
```

```
# standard django Application
class BlogConfig(AppConfig, Default):
    name = 'leonardo_module_blog'
    verbose_name = ("Blog")

default = Default()  # inicialize
```

That’s all.. Leonardo go throught every module defined in your `APPS` and merge all items to main settings file. Complete reference you can see below.

settings.py

in the settings you may have something like this

```
BLOG_TITLE = 'name'

# whatever
```

As you expext every key from settings will be inported and merged into main settings file.

Warning: Be careful if you declare keys in the `module/settings.py`. Every key is imported without special merging process which may override your global settings ! It was designed only for module/app specific defaults.

CHAPTER 5

Components

5.1 Thumbnails

Sometimes is hard to manage all templates to use one thumbnail tag.

Leonardo has one thumbnail tag which lives under `thumbnail` templatetags.

This template tag combine `sorl` and `easy-thumbnails`. This templatetag can render different thumbnails in same template without syntax error.

this makes big advantages with supporting more backends for example we use `easy_thumbnails` and if we install `leonardo_module_eshop` or plain `Django oscar` which requires `sorl-thumbnail` as default thumbnail library we have problem. Leonardo support many variations thumbnail tags

5.1.1 Sorl

```
{% load thumbnail %}
{% thumbnail widget.image.file size format="PNG" as thumb %}
    <img src='{{ thumb.url }}' alt='my-image' />
{% endthumbnail %}
```

5.1.2 Easy-thumbnails

```
{% load thumbnail %}
<img src="{% thumbnail profile.photo 50x50 crop %}" alt="" />
```

5.1.3 Combined

```
{% load thumbnail %}

{% thumbnail widget.image.file size format="PNG" as thumb %}
    <img src='{{ thumb.url }}' alt='my-image' />
{% endthumbnail %}

<img src="{% thumbnail profile.photo 50x50 crop %}" alt="" />
```

For more examples and settings must follow appropriate pages. For Sorl http://sorl-thumbnail.readthedocs.org/en/latest/index.html and for Easy-Thumbnails https://github.com/SmileyChris/easy-thumbnails

5.2 Live configuration

Live configuration is based on django-constance with few improvements.

- basic grouping via `CONSTANCE_CONFIG_GROUPS` which makes tabs for django admin
- access to config keys from standard django settings
- really live settings, set every value to django settings and respect the default value from them

Live settings now supports these types:

- String
- Number
- Boolean
- Dict

```
class Default(object):

    optgroup = 'GA'

    apps = [
        'analytical',
        'leonardo_module_analytics',
    ]

    config = {
        'GOOGLE_ANALYTICS_PROPERTY_ID': ('xx-xxx-x', _('Google Site identificator')),
        'GOOGLE_ANALYTICS_SITE_SPEED': (False, _('analyze page speed')),
        'GOOGLE_ANALYTICS_ANONYMIZE_IP': (False, _('anonymize ip')),
        'Media Thumbnails': {
            'SIZES': ({
                'SMALL': '64x64',
                'MEDIUM': '265x265',
            }, 'Help Text')
        }
    }

default = Default()
```

Note: Please be sure about keys in config, all is merged into one big dictionary which is used. Last wins.

All config keys will be namespaced under GA and available from standard django settings:

```
from django.conf import settings

settings.GOOGLE_ANALYTICS_PROPERTY_ID
-> xx-xxx-x
```

Warning: This may not work on some special environments like a sqlite. For stable usage in modules we recommend using via `constance` see below.

5.2.1 Using in your code

```
from django.conf import settings

if settings.THE_ANSWER == 42:
    answer_the_question()
```

of via constance module

```
from constance import config

if config.THE_ANSWER == 42:
    answer_the_question()
```

5.3 Messagess

Drop-in replacement for django.contrib.messages which handles Leonardo's messaging needs (e.g. Celery tasks, AJAX communication, etc.).

for async messages install

```
pip install django-async-messages
```

from https://github.com/codeinthehole/django-async-messages

```
>>> from leonardo import messages
>>> barry = User.objects.get(username='barry')
>>> messages.debug(barry, "Barry was here")
>>> messages.info(barry, "Hi, Barry")
>>> messages.success(barry, "Barry, your report is ready")
>>> messages.warning(barry, "Barry, you didn't lock your session")
>>> messages.error(barry, "You are not Barry")
```

or standard request

```
>>> messages.error(request, "You are not Barry")
```

5.4 Modals Dialogs

Standard modals via views

`modal_size` - valid options `md`, `lg`, `sm`

```
from horizon import forms

class WidgetDeleteView(forms.ModalFormView):

    form_class = WidgetDeleteForm

        template_name = 'leonardo/common/modal.html'

    def get_context_data(self, **kwargs):
        context = super(WidgetDeleteView, self).get_context_data(**kwargs)

        context['url'] = self.request.build_absolute_uri()
        context['form_action'] = 'POST'
        context['modal_header'] = _('Create new Moon')
        context['title'] = _('Create new Moon')
        context['form_submit'] = _('Create')
        context['heading'] = self.get_header()
        context['help_text'] = _('Your awesome help text')
        context['modal_size'] = 'lg'

        return context
```

5.4.1 Lightboxes

For galleries you can use default Lightboxes for Bootstrap 3 see example below:

```
<a class="thumbnail" data-toggle="lightbox" data-title="{{ image.caption }}" data-footer="{{ image.de
  {% thumbnail file.file "320x200" crop="center" as thumbnail %}
  <img class="img-responsive" src="{{ thumbnail.url }}" alt="{{ category_file.default_alt_text }}" />
  {% endthumbnail %}
</a>
```

CHAPTER 6

Contribute

Feel free and contribute to Leonardo CMS ! Follow next steps

6.1 Contributors

6.1.1 How to contribute?

- Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug.
- Fork https://github.com/django-leonardo/django-leonardo.git on GitHub to start making your changes to the **develop** branch.
- Write a test which shows that the bug was fixed or that the feature works as expected.
- Make sure to add yourself to the contributors.
- Send a pull request

You can help further the development of **django-leonardo** by reporting bugs, submitting documentation, patches, with monetary or hardware donations.

6.1.2 Lead developers

- Aleš Komárek (mail@newt.cz) <newt.cz>
- Michael Kutý (mail@majklk.cz) <majklk.cz>

6.1.3 Contributors (in alphabetical order)

- No contribution

6.2 Development Environment

Simplest way is using our SaltStack Formula where you can comfortably specify sources for leonardo with plugins.

6.2.1 Prerequisites

Prerequisites are installed on Ubuntu 12.04 LTS or Raspian Wheezy 7.0 with:

```
$ sudo apt-get install python-pip python-dev python-setuptools git python-virtualenv libtiff5-dev lib
    libfreetype6-dev liblcms2-dev libwebp-dev tcl8.6-dev tk8.6-dev python-tk
```

Prerequisites are installed on Ubuntu 14.04 LTS with:

```
$ sudo apt-get install python-pip python-dev python-setuptools git python-virtualenv libtiff5-dev lib
    libfreetype6-dev liblcms2-dev libwebp-dev tcl8.6-dev tk8.6-dev python-tk
```

Prerequisites are installed on Fedora 20 with:

```
$ sudo yum install python-pip python-dev python-setuptools git python-virtualenv libtiff-devel libjpe
    lcms2-devel libwebp-devel tcl-devel tk-devel
```

Note: If you have problems with installation, please see SaltStack Formula where are all steps of installation.

usualy after successfuly installated prerequisites you can start with Leonardo

```
virtualenv -p /usr/bin/python2.7 /srv/leonardo/sites/mysite

git clone https://github.com/django-leonardo/django-leonardo.git -b develop /srv/leonardo/sites/mysit

vim /srv/leonardo/sites/mysite/local_settings.py
```

put your config to `local_settings.py`:

```
# -*- coding: utf-8 -*-

from __future__ import absolute_import

import sys
from os.path import join, dirname, abspath, normpath

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'HOST': '127.0.0.1',
        'NAME': 'leonardo_mysite',
        'PASSWORD': 'mysite_password',
        'USER': 'leonardo_mysite'
    }
}

CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
        'TIMEOUT': 120,
        'KEY_PREFIX': 'CACHE_MYSITE'
    }
}

SECRET_KEY = 'my_secret_key'

DEBUG = True
```

```
MEDIA_ROOT = '/srv/leonardo/sites/mysite/media/'
STATIC_ROOT = '/srv/leonardo/sites/mysite/static/'

TIME_ZONE = 'Europe/Prague'

LANGUAGE_CODE = 'en'

# your app here
APPS = [
    'blog',
    'forms',
    'news'
]

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'root': {
        'level': 'WARNING',
        'handlers': ['file'],
    },
    'filters': {
        'require_debug_false': {
            '()': 'django.utils.log.RequireDebugFalse'
        }
    },
    'formatters': {
        'verbose': {
            'format' : "[%(asctime)s] %(levelname)s [%(name)s:%(lineno)s] %(message)s",
            'datefmt' : "%d/%b/%Y %H:%M:%S"
        },
        'simple': {
            'format': '%(levelname)s %(message)s'
        },
    },
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'verbose'
        },
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': '/srv/leonardo/sites/mysite/leonardo_server.log',
            'formatter': 'verbose'
        },
        'mail_admins': {
            'level': 'ERROR',
            'filters': ['require_debug_false'],
            'class': 'django.utils.log.AdminEmailHandler',
            'formatter': 'simple',
        },
    },
    'loggers': {
        'django.request': {
            'handlers': ['mail_admins', 'file'],
            'level': 'DEBUG',
            'propagate': True,
        },
    }
}
```

Run

There are several options which you can use, see contrib directory in the repo https://github.com/django-leonardo/django-leonardo/tree/master/contrib

Django `runserver`

```
python /srv/leonardo/sites/mysite/leonardo/contrib/django/manage.py runserver 0.0.0.0:80
```

Tornado

```
python /srv/leonardo/sites/mysite/leonardo/contrib/tornado/server
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Bibliography

[Documentation] http://django-leonardo.readthedocs.org