cookiecutter-djangocms3-buildout Documentation

Release 1.7.0

Emencia

February 12, 2017

Contents

A Cookiecutter template to produce a structure for a Django CMS project.

CHAPTER 1

Features

- Many Django apps pre-configured and ready to work (see *Available mods*);
- Django project is based on:
  - Django CMS;
  - Django Blog Zinnia;
  - Django CKEditor;
  - And many other... All of them are allready configured and ready to work;
- buildout ensure deployment on various environments;
- Web design integration is made with SASS (>=3.2.x) using libraries Foundation and Bourbon;
- Some pre configuration are ready for uwsgi, nginx and monit;

Emencia company uses this tool for web projects along with our techniques and procedures.

Goal is to automatically create and initialize a complete project structure so you don't lose time assembling all parts and components.

CHAPTER 2

Table of contents

2.1 Install

There is no need to install anything but Cookiecutter, when it's done you can use any Cookiecutter templates directly from a repository (local, hosted on github, etc..):

```
pip install cookiecutter
```

It's done.

2.1.1 Created projects requirements

Although the project template don't need anything but Cookiecutter, a created project will requires pip, virtualenv and some libraries to install it.

There is the **required Basic** development libraries and some optional other ones for some tasks. Note that package name can differ depending from your system.

Basic

- `python-dev`;
- `gettext`;
- `gcc`;
- `make`;
- `libjpeg`;
- `zlib`;
- `libfreetype`;

For Postgresql driver (psycopg2)

- `libpq`;

For Mysql driver

- `libmysqlclient-dev`;

For M2Crypto

- `swig`;

For Graphviz

- `graphviz`;
- `libgraphviz-dev`;
- `graphviz-dev`;

The method to install system packages depends on your plateform:

- On **Linux** systems you will install them with your package system like `apt`, see Pillow documentation for Linux;
- On **MacOSX**, the recommended way is to use `brew` utility, see Pillow documentation for OSX;
- **Windows** system is not supported, you probably can install needed stuff but with some works on your own;

Note: All created projects are shipped with a README file that should contains all necessary details to build it and use it. This will be a simplified procedure with a **Makefile** command to launch the buildout process.

2.2 Usage

Firstly, it will produce a new project structure then you will have to deploy it locally and finally enjoy it (webdesign integration, development, etc..).

2.2.1 Produced project structure

Just invoke the Cookiecutter template to create a new project:

```
cookiecutter https://github.com/emencia/cookiecutter-djangocms3-buildout
```

You will be prompted to anwser to some inputs about your project and other inputs to enable some applications:

project_name [Default:*Project name*]

Project name, should be unique into your repository host.

repo_name [Default:A slug created from project name]

This will be used as the repository name.

repo_username [Default:*emencia*]

Your username to use to push the repository.

repo_host [Default:*github.com*]

The hostname of the repository host.

secret_key [Default:A random key]

The secret key to use in Django settings, you should let the default value (a randomly generated string).

enable_accounts [Default:*no*]

Enable the accounts component.

enable_contact_form [Default:*yes*]

Enable a basic contact form with an optional captcha field.

enable_porticus [Default:*yes*]

Enable Porticus application.

enable_slideshows [Default:*yes*]

Enable Slideshows application.

enable_zinnia [Default:*no*]

Enable Zinnia application.

enable_multiple_languages [Default:*no*]

Enable CMS internationalization.

enable_https [Default:*yes*]

Use a dedicated nginx configuration to enable https only through *let's encrypt*.

deploy_user [Default:*django*]

System user where project will be deployed to (in its home directory).

deploy_host [Default: Empty]

Host where to to deploy project in integration mode (using ssh).

prod_host [Default: Empty]

Host where to to deploy project in production mode (using ssh).

changelog_mail_from [Default: Empty]

Email 'from' adress use to send deployment informations.

changelog_mail_to [Default: Empty]

Email 'to' adress use to send deployment informations.

Note: Usually you won't need to care about fields starting with `repo_` or `deploy_` and just let their default values.

Take care that question about applications require a strict `yes` value to enable them, all other value are considered as a `no`.

Once structure has been created, a GIT repository will be initialized with the right remote url and a first commit, ready to push.

2.2.2 Deploy produced project

When project has been created, you need to install it to use it. Do it in your project directory:

```
make install
```

When it's finished, active the virtual environment:

```
source bin/active
```

You can then use the project on the development server:

```
django-instance runserver 0.0.0.0:8001
```

Note: `0.0.0.0` is some sort of alias that mean "bind this server on my ip", so if your local ip is "192.168.0.42", the server will be reachable in your browser with the url `http://192.168.0.42:8001/`.

Note: Note the `:8001` that mean "bind the server on this port", this is a required part when you specify an IP. Commonly you can't bind on the port 80 so allways prefer to use a port starting from *8001*.

Note: If you don't know your local IP, you can use `127.0.0.1` that is an internal alias to mean "my own network card", but this IP cannot be reached from other computers.

The first required action is the creation of a CMS page for the home page and also you should fill-in the site name and its domain under `Administration > Sites > Sites > Add site`.

2.3 Projects structure

Projects are created with the many components that are available for use. These components are called **mods** and these mods are already installed and ready to use. You can enable or disable them, as needed.

2.3.1 django-instance

This is the command installed to replace the old `manage.py` script in Django. `django-instance` is located into the `project/bin` directory and is aware of installed eggs from buildout.

2.3.2 Git ignore

Project embeds many `.gitignore` files to avoid to commit some files into your repository.

Aim is to never commit files created from buildout (installed packages, app development sources, etc..), compiled static files, project media files and database.

2.3.3 Makefile

Project embeds a `Makefile` file that contains some usefull commands to build your project.

`install` to proceed to a new install of this project. Use clean command before if you want to reset a current install

`install-dev` to proceed to a new install of this project with additional sources for development

`install-foundation` to install (or re-install) Foundation sources

`clean` to clean your local repository from all stuff created by buildout and instance usage

`delpyc` to remove all `*.pyc` files, this is recursive from the current directory

`assets` to minify all assets and collect static files

`scss` to compile all SCSS stuffs with compass

`syncf5` to update staticfiles with Foundation sources (use this is you upgrade Foundation sources)

`tar_data` to dump applications datas to json files then put them in a tarball

`import_db` to import dumped datas, you should empty the database before

`reload` to reload uwsgi instance (for integration and production only)

2.3.4 SASS

Produced project contain a SASS structure ready to use and contains Foundation and Bourbon libraries. You should find a structure like this:

```
sass/
-- bourbon
-- foundation5
-- scss
```

The `scss` directory is where live your SASS sources to compile to CSS. SASS sources and libraries must be compatible with any compiler which implement SASS >= 3.2.x references.

Included HTML templates was done using Foundation, so you will need to change them if you plan to use another SASS framework.

We strongly advise you to use Boussole to avoid messing with some Ruby or Node.js installation. Compass 1.x is still supported. In the SASS sources directory a default configuration is available for both of them.

Warning: Compass include also a SASS framework. It's most important feature is *vendor prefix* for many CSS3 properties but this is not needed anymore with recent browsers.
If you use still use Compass compiler, **never use any Compass mixin or stuff** in your SASS sources, it will break compatibility for people using another compiler.
For CSS3 properties requiring vendor prefix, we include Bourbon framework instead. If you are not sure about vendor prefix needs, search in Can I Use database.

2.3.5 Adding application

If you plan to integrate a new app into a project, always use the buildout system to ensure its portability (pip requirements file is not used in our system).

To do this, just open and edit the `buildout.cfg` file to add the new egg name to be installed. For more details, read the buildout documentation.

Also it is always preferable to use the mods system to configure new added apps and keep the `settings.py` only for Django owned settings.

Note: Project contains a `version.cfg` file that define exact version to use for listed eggs in `buildout.cfg`. All existing mods have their eggs defined in this file, if you need to enable a mod that you skipped during project creation, just find its egg name in `version.cfg` and add it to the eggs in `buildout.cfg`.

2.3.6 How the Mods work

The advantage of centralizing app configurations in their mods is to safely gather together them distinctly from the Django basic settings (like SMTP config, database access, middlewares, etc..). Furthermore it is easier to enable or disable apps.

To create a new mods:

- Create a directory in `project/mods_available/` that contains at least one empty `__init__.py` and a `settings.py` to enable the app (using `settings.INSTALLED_APPS`) in the project and potentially its settings and urls;

- The `settings.py` and `urls.py` files in this directory will be executed automatically by the project (the system loads them after the project ones so that a mods can overwrite the project initial settings and urls);
- N.B. With the Django `runserver` command, a change to these files does not reload the project instance; you need to relaunch it yourself manually;

To enable a new mods, you need to create its symbolic link (**a relative path** to the available mod) in `project/mods_enabled`. To disable it, simply delete the symbolic link.

Since Django 1.8, every template settings are contained in their backend entry in `settings.TEMPLATES`. We actually assume to only use the default Django template backend in our project. So mods will be able to manipulate template settings using the default backend that will be the index 0 of the backends list:

```
TEMPLATES[0]['OPTIONS']['context_processors'] = ....
```

Trying to use the old template settings will result in an error.

2.3.7 Available mods

accounts

Enable Django registration and everything you need to allow users to register and to connect/disconnect. Sample views and forms are include so it can be easily used.

It includes:

- A view for the login and one for the logout;
- All the views for the registration request (request, confirmation, etc.);
- A view to ask for the reinitialization of a password;
- Email sending;

In the `skeleton.html` template, a partial HTML code is commented. Uncomment it to display the *logout* button when the user is connected.

The registration process consists in sending an email (sender/destination emails have to be configured in settings) with the registration request to an administrator responsible for accepting them (or not). Once validated, an email is sent to the user to confirm his registration by way of a link. Once this step has been completed, the user can connect.

Also, note that this app extend the user model with a profile model.

This profile is naive because it implement some comon additional fields for sample but you may not need all of them, if you change it you will need to do some changes also in registration view, forms and email senders.

Note: Included forms and templates depends on *crispy_forms* mod.

---

admin_style

Enable djangocms-admin-style to enhance the administration interface. Also enable django-admin-shortcuts.

admin-style is an enhancement for Django admin that have been developed by Django CMS team.

assets

Enable django-assets to combine and minify your *assets* (CSS, JS).

There are two used minification libraries:

- yuicompressor for CSS;
- googleclosure for Javascript;

Both of them requires the installation of Java (OpenJDK installed on most Linux systems is sufficient), they are installed through some Python meta package from Pypi.

In general, this mod is required. If you do not intend to use it, you will need to modify the project's default templates to remove all of its occurrences.

Assets are defined in `project/assets.py` and some apps can defined their own `asset.py` file but our main file does not use them.

Our `assets.py` file is divised in three parts :

- BASE BUNDLES: Only for app bundle like Foundation Javascript files, Modernizr files, etc..;
- MAIN AVAILABLE BUNDLES: Where you defined main bundles for the frontend, use app bundles previously defined;
- ENABLE NEEDED BUNDLE: Leading bundles you effectively want to use and expose in your templates. Bundle that are not defined here will not be reachable from templates and won't be minified;

autobreadcrumbs

Enable autobreadcrumbs to add automatic breadcrumbs building in templates and applications.

ckeditor

Enable and define customization for the CKEditor editor. It is enabled by default and used by Django CKEditor in the *cms* mod, and also in *zinnia*.

Note that DjangoCMS use it's own app named "djangocms_text_ckeditor", a djangocms plugin to use CKEditor (4.x).

But Zinnia (and some other generic app) use "django_ckeditor" that ship the same ckeditor but without cms addons.

This mod contains configuration for all of them.

And some useful patches/fixes :

- the codemirror plugin that is missing from the ckeditor's django apps;
- A system to use the "template" plugin (see views.EditorTemplatesListView for more usage details);
- Some overriding to have content preview and editor more near to Foundation;

cms

Django CMS allows for the creation and management of the content pages that constitute your site's tree structure. By default, this component enables the use of *filebrowser*, Django CKEditor and emencia-cms-snippet (a clone of the snippets' plugin with a few improvements).

By default it is configured to use only one language. See its `urls.py` to find out how to enable the management of multiple languages.

codemirror

Enable Django Codemirror to apply the editor with syntax highlighting in your forms (or other content).

It is used by the snippet's CMS plugin.

contact_form

A simple contact form that is more of a standard template than a full-blown application. You can modify it according to your requirements in its `apps/contact_form/` directory. Its HTML rendering is managed by *crispy_forms* based on a customized layout.

Note: Depends on *recaptcha* and *crispy_forms* mods.

---

crispy_forms

Enable the use of django-crispy-forms and crispy-forms-foundation.

crispy_forms is used to manage the HTML rendering of the forms in a finer and easier fashion than with the simple Django form API.

crispy-forms-foundation is a supplement to implement the rendering with the structure (tags, styles, etc.) used in Foundation.

datadownloader

Add **'_django-datadownloader'_** to your project to manage data (media/db).

debug_toolbar

Add django-debug-toolbar to your project to insert a tab on all of your project's HTML pages, which will allow you to track the information on each page, such as the template generation path, the query arguments received, the number of SQL queries submitted, etc.

This component can only be used in a development or integration environment and is always disabled during production.

Note that its use extends the response time of your pages and can provokes some bugs (see the warning at end) so for the time being, this mods is disabled. Enable it locally for your needs but never commit its enabled mod and remember trying to disable it when you have a strange bug.

> **Warning:** Never enable this mod before the first database install or a syncdb, else it will result in errors about some table that don't exist (like "django_site").

emencia_utils

Group together some common and various utilities from `project.utils`.

filebrowser

Add Django Filebrowser to your project so you can use a centralized interface to manage the uploaded files to be used with other components (*cms*, *zinnia*, etc.).

The version used is a special version called *no grappelli* that can be used outside of the *django-grapelli* environment.

Filebrowser manage files with a nice interface to centralize them and also manage image resizing versions (original, small, medium, etc..), you can edit these versions or add new ones in the settings.

Note: Don't try to use other resizing app like sorl-thumbnails or easy-thumbnails, they will not work with Image fields managed with Filebrowser.

---

filer

Mod for django-filer and its DjangoCMS plugin

Only enable it for specific usage because this can be painful to manage files when Filebrowser and django-filer are enabled in the same project.

flatpages

Enable the use of Django flatpages app in your project. Once it has been enabled, go to the `urls.py` in this mod to configure the *map* of the urls to be used.

google_tools

Add django-google-tools to your project to manage the tags for *Google Analytics* and *Google Site Verification* from the site administration location.

Note: The project is filled with a custom template `project/templates/googletools/analytics_code.html` to use Google Universal Analytics, remove it to return to the old Google Analytics.

---

icomoon

Django Icomoon help you to manage your webfonts with Icomoon service. It won't work with a webfont not generated on Icomoon site because it depends on a JSON manifest file (you could make it yourself but it's a little bit complicated).

This mod can handle many webfonts if you need, you just have to define them in the mod settings, at least one webfont is required.

Once one or more webfonts are defined, Django Icomoon can help you to automatically deploy them in your project from downloaded Zip on Icomoon using a command line `django-instance icomoon_deploy`.

Also when deployed and the webfonts are loaded in your templates, you can visualize every icons from a gallery located at `/icomoon/`.

logentry

Enable django-logentry-admin to browse all admin log entries at contrary to default Django admin behavior that only display the last entries.

pdb

Add Django PDB to your project for more precise debugging with breakpoints.

N.B. Neither `django_pdb` nor `pdb` are installed by buildout. You must install them manually, for example with pip, in your development environment so you do not disrupt the installation of projects being integrated or in production. You must also add the required breakpoints yourself.

See the the django-pdb Readme for more usage details.

Note: Make sure to put django_pdb after any conflicting apps in INSTALLED_APPS so that they have priority.

So with the automatic loading system for the mods, you should enable it with a name like "zpdb", to ensure that it is loaded at the end of the loading loop.

---

porticus

Add Django Porticus to your project to manage file galleries.

There is a DjangoCMS plugin for Porticus, it is not enabled by default, you will have to uncomment it in the mod settings.

recaptcha

Enable the Django reCaptcha module to integrate a field of the *captcha* type via the Service reCaptcha. This integration uses a special template and CSS to make it *responsive*.

Note: If you do in fact use this module, go to its mods setting file (or that of your environment) to fill in the public key and the private key to be used to transmit the data required.

By default, these keys are filled in with a *fake* value and the captcha's form field therefore sends back a silent error (a message is inserted into the form without creating a Python *Exception*).

---

sendfile

Enable django-sendfile that is somewhat like a helper around the **X-SENDFILE headers**, a technic to process some requests before let them pass to the webserver.

Commonly used to check for permissions rights to download some private files before let the webserver to process the request. So the webapp can execute some code on a request without to carry the file to download (than could be a big issue with some very big files).

django-sendfile dependancy in the buildout config is commented by default, so first you will need to uncomment its line to install it, before enabling the mod. Then you will need to create the directory to store the protected medias, because if you store them in the common media directory, they will public to everyone.

This directory must be in the project directory, then its name can defined in the `PROTECTED_MEDIAS_DIRNAME` mod setting, default is to use `protected_medias` and so you should create the `project/protected_medias` directory.

Your webserver need to support this technic, no matter on a recent nginx as it is allready embeded in, on Apache you will need to install the Apache module XSendfile (it should be availabe on your distribution packages) and enable it in the virtualhost config (or the global one if you want), see the Apache module documentation for more details.

Then remember to update your virtualhost config with the needed directive, use the Apache config file builded from buildout.

The nginx config template allready embed a rule to manage `project/protected_medias` with sendfile, but it is commented by default, so you will need to uncomment it before to launch buildout again to build the nginx config file.

Note: By default, the mod use the django-sendfile's backend for development that is named `sendfile.backends.development`. For production, you will need to use the right backend for your web-server (like `sendfile.backends.nginx`).

Finally you will need to implement it in your code as this will require a custom view to download the file, see the django-sendfile documentation for details about this. But this is almost easy, you just need to use the `sendfile.sendfile` method to return the right Response within your view.

site_metas

Enable a module in `settings.TEMPLATE_CONTEXT_PROCESSORS` to show a few variables linked to Django sites app in the context of the project views template.

Common context available variables are:

- `SITE.name`: Current *Site* entry name;
- `SITE.domain`: Current *Site* entry domain;
- `SITE.web_url`: The Current *Site* entry domain prefixed with the http protocol like `http://mydomain.com`. If HTTPS is enabled 'https' will be used instead of 'http';

Some projects can change this to add some other variables, you can see for them in `project.utils.context_processors.get_site_metas`.

sitemap

This mod use the Django's Sitemap framework to publish the `sitemap.xml` for various apps. The default config contains ressources for DjangoCMS, Zinnia, staticpages, contact form and Porticus but only ressource for DjangoCMS is enabled.

Uncomment ressources or add new app ressources for your needs (see the Django documentation for more details).

slideshows

Enable the emencia-django-slideshows app to manage slide animations (slider, carousel, etc.). This was initially provided for *Foundation Orbit* and *Royal Slider*, but can be used with other libraries if needed.

staticpages

This mod uses emencia-django-staticpages to use static pages with a direct to template process, it replace the deprecated mod *prototype*.

thumbnails

Mod for easy-thumbnails a library to help for making thumbnails on the fly (or not).

Generally **this is not recommended**, because by default we allready enable Filebrowser that allready ships a thumbnail system.

urlsmap

django-urls-map is a tiny Django app to embed a simple management command that will display the url map of your project.

xiti

Mod to define Django-xiti settings to load Xiti HTML code into templates

Since Xiti usage is not common, this mod is not installed or enabled on default install, you will need to enable it's egg in buildout, enable its mod and finally update `marketing_tags.html` to load it.

zinnia

Django Blog Zinnia allows for the management of a blog in your project. It is well integrated into the *cms* component but can also be used independently.

2.4 Common topics around project usage

Additionally to Django, a created project is based on many other tools you should know, here are their topics.

2.4.1 Boussole

Previously we were using Compass that was a pain to manage with Ruby environment.

Boussole is full Python solution to build your CSS stylesheets from SASS sources.

You can install it in your virtualenv if you need a specific version or install globally at your system level.

2.4.2 Compass

Compass is a Ruby tool used to compile SASS sources in **CSS**.

A recent install of Ruby and Compass is required first for this purpose.

2.4.3 Webfonts

Often, we use webfonts to display icons instead of images because this is more flexible to use (can take any size without to re-upload it) and results on less files. It's also more *CSS friendly*.

We use Icomoon service to build webfont because we can centralize their sources and the service generate a clean ZIP archive containing all needed stuff (all font kind, icon manifest, sample css, etc..).

Within our project We manage it through Django Icomoon to deploy webfont updates (using the downloaded ZIP) and to display an icon gallery.

Note: Django Icomoon usage is a new feature (see History for details), it may not be allready configured in your project if too old. But you can easily add it to, it should be compatible from Django '1.4.x' to '1.8.x'.

Just download the webfont ZIP from your Icomoon project, put it in your Django project and use the command line (adjust zip file path if needed):

```
django-instance icomoon_deploy Default icomoon.zip
```

Font files will be deployed to their directory in statics (defined in mod settings) then a SCSS file will be generated so you can directly recompile them to build your CSS.

When it's done you can reach the gallery on:

```
/icomoon/
```

Warning: You need to be authenticated to view the gallery.

Note: There is allready a default webfont installed in your project with some default used icons like those ones required for **Slick.js** plugin.

2.4.4 Assets management

Why

In the past, assets management was painful with some projects, because their includes was often divided in many different templates. This was causing issues to update some library or retrieve some code.

Often it resulted also in pages loading dozen of asset files and sometime much more. This was really a bad behavior because it slowed pages loading and added useless performance charge on the web server.

This is why we use an **asset manager** called django-assets which is a subproject of webassets. Firstly read the webassets documentation to understand how is working its **Bundle** system. Then you can read the django-assets that is only related about Django usage with the settings, templatetags, etc..

How it works

Asset managers generally perform two tasks :

- Regroup some kind of files together, like regrouping all Javascript files in an unique file;
- Minimize the file weight with removing useless white spaces to have the code on unique line;

Some asset manager implement this with their own file processor, some other like webassets are just "glue" between the files and another dedicated *compiler* like yuicompressor.

Environments

Asset management is really useful within integration or production environments and so when developing, the manager is generally disabled and the files are never compiled, you can verify this with looking at your page's source code.

make assets

Project have a `make assets` command that is useful **on integration and production environment** to deploy and update your assets in the `static/` directory. In fact **this command is always required in these environments** when you deploy a new update. Also you should never use it on development environment because it can cause you many troubles.

What does this command :

1. Remove some previous minified assets;
2. Collecting all static files from your project and installed apps to your `settings.STATIC_ROOT` directory;
3. Use django-assets to *compile* all defined bundles using previously collected files;
4. Re-collecting static files again to collect the compiled bundle files;

Static files directories

In your `settings.py` file you should see :

```
STATIC_ROOT = join(PROJECT_PATH, 'static')
```

It define the *front* static file directory. But **never put yourself a file in this directory**, it is **reserved** for collected files in **integration and production environment** only.

All static files sources will go in the `project/webapp_statics` directory, it is defined in the *assets* mod:

```
ASSETS_ROOT = join(PROJECT_PATH, 'webapp_statics/')
STATICFILES_DIRS += (ASSETS_ROOT,)
```

This way, we allways have separated directories for the sources and the compiled files. This is required to never commit compiled files and avoid conflicts between development and production environments.

The rule

Never, ever, put CSS stylesheets in your templates, NEVER. You can forget them and they will be deployed in production and forgeted, this can be painful for other developers that coming after you. So **always add CSS stylesheets by the way of SCSS sources** using Compass.

For Javascript code this is different, sometime we need to generate some code using Django templates for some specific cases. But if you use a same Javascript code in more than one template (using inheriting or so), you must move the code to a Javascript file.

Developers should never have to search in templates to change some CSS or Javascript code that is used in more than one page.

2.4.5 Developing application

Sometimes, you will need to develop some new app package or improve them without to embed them within the project.

You have two choices to do that:

- Use `develop` buildout variable to simply add your app to the developped apps, your app have to exists at the root of buildout project;
- Use `vcs-extend-develop` buildout variable to define a repository URL to the package sources;

Even they have the same base name *develop*, these two ways are differents:

- The first one simply add a symbolic link to the package in your Python install without to manage it as an installed eggs, it will be accessible as a Python module installed in the Python virtual environment. This method does not require that your app have a repository or have been published on PyPi;
- The second one install the targeted package from a given repository instead of a downloaded package from PyPi, it act like an installed eggs but from which you can edit the source and publish to the repository. And so your app name have to be defined in the buildout's egg variable, buildout will see it in `vcs-extend-develop` and will not try to install it from PyPi but from the given repository url;

In all ways, your apps is allways a full package structure that mean this is not a simple Python module, but its package structure containing stuff like `README` file and `setup.py` at the base of the directory then the Python module containing the code. Trying to use a simple Python module as a develop app will not work.

Which one to use and when

- If you want to **develop a new package**, it's often much faster to create its package directory structure at the root of your buildout project then use it within `develop`. You would move it to `vcs-extend-develop` when you have published it;
- If you want to **develop an allready published package**, you will use `vcs-extend-develop` with its repository url, this so you will be able to edit it, commit changes then publish it;

Most of Emencia's apps are allready setted within `vcs-extend-develop` in the buildout config for development environment (`development.cfg`) but disabled, just uncomment the needed one.

Take care, an Egg that is installed from a repository url is validated on its version number if defined in the `versions.cfg`, and so if your develop egg contains a version number less than the one defined in `versions.cfg`, buildout will try to get the most recent version from PyPi, so allways manage the app version number.

2.4.6 PO-Projects

> **Warning:** This is on the way to be deprecated since the PO-project server future is not known.

It aims to ease PO translations management between developpers and translation managers.

The PO-Projects client is pre-configured in all created projects but disabled by default. When enabled, its config file is automatically generated (in `po_projects.cfg`), don't edit this file because it will be regenerated each time buildout is used.

The principe is that **developpers and translators does not have anymore to directly exchange PO files**. The developpers update the PO to the translation project on PO-Project webservice, translators update translations on PO-Project service frontend and developpers can get updated PO from the webservice.

To use it, you will have first to enable it in the buildout config, to install the client package, fill the webservice access and buildout part. Then when it's done, you have to create a project on PO-Project webservice using its frontend, then each required language for translation using the same locale names that the ones defined in the project settings.

There is only two available actions from the client :

Push action The `push` action role is to send updated PO (from Django extracts) from the project to the PO-Project webservice.

Technically, the client will archive the locale directory into a tarball then send it to the webservice, that will use it to update its stored PO for each defined locales.

Common way is (from the root of your project):

```
cd project
django-instance makemessages -a
cd ..
po_projects push
```

Pull action The `pull` action role is to get the updated translations from the webservice and install into the project.

Technically, the client will download a tarball of the latest locale translations from the webservice and deploy it to your project, note that it will totally overwrite the project's locale directory.

Common way is (from the root of your project):

```
po_projects pull
```

Then reload your webserver.

Note that the client does not manage your repository, each time you change your PO files (from Django `makemessages` action or `pull` client action) you still have to commit them.

2.4.7 Dr Dump

Dr Dump is an utility to help you to dump and load datas from your Django project's apps. It does not have any command line interface, just a buildout recipe (emencia-recipe-drdump) that will generate some bash scripts (`datadump` and `dataload`) in your `bin` directory so you can use them directly to dump your data into a `dumps` directory.

If the recipe is enabled in your buildout config (this is the default behavior), its bash scripts will be generated again each time you invoke a buildout.

Buildout will probably remove your dumps directory each time it re-install Dr Dump and Dr Dump itself will overwrite your dumped data files each time you invoke it dump script. So remember backup your dumps before doing this.

Note that Dr Dump can only manage app that it allready know in the used map, if you have some other packaged app or project's app that is not defined in the map you want to use, you have two choices :

- Ask to a repository manager of Dr Dump to add your apps, for some *exotic* or uncommon apps it will probably be refused;
- Download the map from the repository, embed it in your buildout project and give its path into the `dependancies_map` recipe variable so it will use it.

The second one is the most easy and flexible, but you will have to manage yourself the map to keep it up-to-date with the original one.

2.5 History

Versions come from git tags.

2.5.1 Version 1.7.0 - 2017/02/12

- Fixed new pep error E305;
- Excluded migrations folder from Flake8 check;
- Use same SSL certification folder on dev host;
- Added Fabric script for install/deploy production version;
- Added project CHANGELOG file and build first release number with `jinja2-time`;

- Use a `.flake8` file for Flake8 settings
- Fixed bug in generated project `__init__.py` file that was causing trailing whitespace;
- Fixed cookieconsent template to use only simple quote `'` to surround variable strings, close #75;
- Remove Sphinx from requirement in development.cfg, close #76;
- Added new setting `ENABLE_STATICPAGES` to enable staticpages ressources, default is true excepted for production, close #77;
- Upgraded to `django-datadownloader==0.2.3`, close #78;
- Removed forgotten royalslider images, close #79;
- Change every occurrences of transition path `{% load static from staticfiles %}` to the recent one `{% load static %}`, close #80;
- Dropped yuicompressor usage in profit of rCSSmin filter, close #81 and close #56;
- Relaxed `six` and `pyparsing`, close #84;
- Upgraded to `crispy-forms-foundation==0.5.5`;
- Removed unused `pages/3_cols.html` template and updated CMS settings to enable page template `pages/free.html`;

2.5.2 Version 1.6.0 - 2016/12/17

- Now require `cookiecutter==1.4.0` (although for now this should not break for >=1.1.0,<1.4.0);
- Fixed typo on `secret_key` template part in `cookiecutter.json`, close #73;
- Updated project Readme;
- Updated install documentation about required devel libraries;
- Upgraded to `django==1.8.17`;

2.5.3 Version 1.5.0 - 2016/11/29

- Added files to use Continuous Integration (*CI*) with Jenkins/Fabric;
- Cleaned code with flake8 and add a rule to run flake8 into Makefile;
- Added `django-datadownloader`;
- Added initial migration for `project.contact_form`;
- Upgraded to `django==1.8.16`, close #72;

2.5.4 Version 1.4.4 - 2016/10/06

- Upgraded to `django==1.8.15`, close #68;
- Fixed icomoon mod settings for `sass` dir new path, close #69;
- Upgraded to `django-debug-toolbar==1.5` to fix a bug with sqlquery panel, close #70
- Added makefile action `livehtml` for `sphinx-autobuild` usage (have to install it before, not in dependancies);

- Added new settings `MARKETINGTAGS_ENABLED` (default value depend on `DEBUG` setting) to determine marketing tags display from `skeleton.html`. Also added an additional context processor to expose this setting to default template context, close #71;

2.5.5 Version 1.4.3 - 2016/09/24

- Fix minor typo on CookieConsent message quotes, close #66;
- Disabled `PO-Projects-client` egg and buildout part again since nap is breaked, close #67;
- Upgraded to `django-assets==0.12`;

2.5.6 Version 1.4.2 - 2016/09/04

- Removed `emencia-cookie-law` in profit of a Javascript library Cookie Consent which works better with webapp caches;
- Fixed `js/jquery/button-dropdown-trick.js` which contained unecessary Foundation init;

2.5.7 Version 1.4.1 - 2016/08/20

- Fixed `cookiecutter.json` so the settings files are rendered correctly with Jinja variable from cookiecutter during project generation, close #64;

2.5.8 Version 1.4.0 - 2016/08/05

- Upgraded to `django==1.8.14`, close #63;
- Upgraded to `django-google-tools==1.1.0`, close #60;
- Enable `mptt` through Zinnia settings, so we can use `recursetree` tag in `categorie_tree.html` zinnia template, close #57;
- Restore `PO-Projects-client` installation, close #62;
- Add some Javascript code in `app.js` to implement clickable slider from `emencia-django-slideshows`, close #58;
- Added a template `pagination.html` for paginate object with generic class based view `ViewList`;

2.5.9 Version 1.3.0 - 2016/06/12

- Added usage of googleclosure for Javascript minification instead of `yuicompressor` (caused many issues with recent scripts because yui is not maintained anymore against recent Ecmascript versions). Related to issue #50;
  - `closure` is installed through a Python meta package alike `yuicompressor`;
  - `yuicompressor` is still used for CSS minification until we find something as stable;
  - Added Makefile action `delminifiedassets` to clean previous minified files from leading assets bundles;
- Added a note about Ckeditor configurable sanitizer in its mod settings. Related to issue #46;

2.5.10 Version 1.2.2 - 2016/05/29

- Freezed to `djangocms-text-ckeditor==2.9.3`, close #55;
- Upgraded to `porticus==1.1.0`, close #54;
- Upgraded to `cmsplugin-porticus==0.4.1`, close #54;
- Upgraded to `cmsplugin-zinnia==0.8.1`, close #53;
- Added "CurrentMediaQuery" plugin and enable it only when Django debug mode is on;

2.5.11 Version 1.2.1 - 2016/05/17

- Dropped assets (Javascript, SASS and CSS) for RoyalSlider, Masonry and MMenu since they are not really used anymore;
- Changed `sass/scss/addons/_type.scss` to override Foundation types for titles instead of using extend;
- Close #48:
  - Cleaning `js/app.js`: removed unused code, moved sample plugins code to their own file within `js/jquery/`;
  - Fix broken Ckeditor within cms text plugin introduced since Image-swapper plugin has been moved to its own file (`js/jquery/jquery.image-swapper.js`);

2.5.12 Version 1.2.0 - 2016/05/02

Move to full libsass support but stay compatible with "Compass 1.x".

This is related to issue #43

- Moved `compass` directory to `sass` directory with a new structure;
  - Divided addons files;
  - Added Bourbon 4.2.6;
  - Foundation5 SASS sources now lives in sass directory;
  - Keep a config file for Compass support;
- Removed Foundation5 sources directory, now we only ship SASS and Javascripts sources in their respective location;
- Updated Makefile action `syncf5` to synchronize SASS sources to the `sass` directory;
- flags stylesheet is not supported for now because it stand Compass sprites;
- admin_styles stylesheet is not supported for now;
- Updated documentation;
- Dropped `admin_tools` mod that is not supported anymore;
- Fixed a bug with a wrong import path for icomoon fonts;
- Upgrade to django-icomoon 0.4.0, close #47;

2.5.13 Version 1.1.1 - 2016/05/02

- Add option to use https within nginx conf;

2.5.14 Version 1.1.0 - 2016/04/19

- Upgraded to `django==1.8.12`;
- Upgraded to `django-icomoon==0.4.0`;
- Upgraded to `django-xiti==0.1.1`;

2.5.15 Version 1.0.0 - 2016/03/19

- Upgraded dependencies versions for upgrade to `Django==1.8`;
  - `django==1.8.11`;
  - `psycopg2==2.6.1`;
  - `Pillow==3.1.1`;
  - `django-mptt==0.7.4`;
  - `django-cms==3.2.3`;
  - `django-registration-redux==1.4`;
  - `djangocms-admin-style==1.1.0`;
  - `django-admin-tools==0.7.2`;
  - `django-filebrowser-no-grappelli==3.6.1`;
  - `django-assets==0.11`;
  - `django-recaptcha==1.0.5`;
  - `django-debug-toolbar==1.4`;
  - `django-extensions==1.6.1`;
  - `django-filer==1.1.1`;
  - `cmsplugin-filer==1.0.1`;
  - `django-icomoon==0.3.1`;
  - `django-sendfile==0.3.10`;
  - `easy-thumbnails==1.5`;
  - `django-contrib-comments==1.6.2`;
  - `django-blog-zinnia==0.16`;
  - `django-tagging==0.4.1`;
  - `django-taggit==0.18.0`;
  - `sorl-thumbnail==12.2`;
- Removed all occurences to `socialaggregator` that is not supported anymore;
- Updated project settings and mods settings to use the new `TEMPLATE` setting that contain all template backends settings;

- Added empty `TEXT_ADDITIONAL_ATTRIBUTES` setting for ckeditor;
- Some minor changes and cleaning in mods settings;
- Added mod for `autobreadcrumbs`;
- Updated `djangocms_admin_style` Sass and CSS stylesheets to the app version 1.1.0;
- Patched them for Filebrowser and also for a bug regression with libsass 3.3.3;
- Although these Sass stylesheets are in compass directory, they can only be compiled with libsass;
- Upgraded to `django-crispy-forms==1.6.0` to remove some warnings from django checks;

2.5.16 Version 0.9.3 - 2015/12/19

- Upgraded to `django-cms==3.1.4`;
- Upgraded to `django-admin-shortcuts==1.2.6`;
- Upgraded to `djangocms-admin-style==0.2.8`;
- Updated `djangocms-admin-style` SCSS source and recompile them again, it should definitively close issue #39;
- Removed `compass/Gemfile` because it cause too many issues when switching between rvm gemset (like to compile the main scss then the admin one);

2.5.17 Version 0.9.2 - 2015/12/17

Upgrade to buildout 2.5.0 and dependancies:

- Removed `bootstrap.py`, now we just install buildout throught pip;
- Upgraded to `setuptools>=19.1`;
- Upgraded to `pip>=7.1.2`;
- Upgraded to `buildout==2.5.0`, close #41;
- Upgraded to `zc.recipe.egg==2.0.3`;
- Upgraded to `buildout.recipe.uwsgi==0.1.1`;
- Upgraded to `collective.recipe.cmd==0.11`;
- Upgraded to `collective.recipe.template==1.13`;
- Upgraded to `djangorecipe==2.1.2`;
- Updated Makefile `install` action for theses changes;
- Updated `[uwsgi]` buildout part since `buildout.recipe.uwsgi==0.1.1` deprecate option prefix `xml-` in profit of `config-`;
- Added `pip-selfcheck.json`, `gestus.cfg` and `po_projects.cfg` to Makefile `clean` action;

For now we are relaxing again `setuptools` and `pip` to a knowed working version or better. We may fix a version again in future if we encounter some bug.

2.5.18 Version 0.9.1 - 2015/12/13

- Added Javascript library jquery-smartresize for **Debounced and Throttled Resize Events for jQuery**. Not enabled by default. This close #42;

2.5.19 Version 0.9.0 - 2015/12/13

Goal of this version was to port structure, code and components to `Django==1.7`.

Many Django apps have been upgraded and some mods settings have been updated.

There is too much changes to write them all here, see the dedicated document Porting to Django 1.7 history for full details.

2.5.20 Version 0.8.2 - 2015/10/30

- Fixed usage of template context variable for `DEBUG` setting, seems it's not exposed in context as uppercase since a long time (if even been), it's lowercase now;
- Fixed Ckeditor custom `styles.js` not loaded from mod, close #35;
- Use staticfiles template tag instead of STATIC_URL in our shipped templates, close #36;
- Fixed wrong gitignore that caused uncommited foundation5 sources when pushing created new projects to their repository (will need to watch for this gignore changes when eventually update foundation sources from last their version), close #38;
- Updated to `emencia-cookie-law==0.2.3`;
- Added `django-xiti==0.1.0` structure (template, mod, etc..) but not installed or enabled on defaut install;

2.5.21 Version 0.8.1 - 2015/10/22

- Fixed missing `__init__.py` in `project/utils/templatetags`, close #34;
- Update to `zinnia-wysiwyg-ckeditor==1.2` to get rid of `django-ckeditor-updated` dependancy and now stands only on `django-ckeditor`. Note that we don't go to `zinnia-wysiwyg-ckeditor==1.3` because it depends on `django-ckeditor=5.x` that we didn't audit yet;

2.5.22 Version 0.8.0 - 2015/10/18

- Updated Foundation to `5.5.3` version, this require now Compass 1.x install to compile, close #22;
- Updated Makefile for some Foundation install strategy changes;
- Updated SCSS to fit to Foundation changes;
- Updated to `django-icomoon==0.3.0`;
- Updated documentation for new methodology with webfont since `django-icomoon` usage;

2.5.23 Version 0.7.6 - 2015/10/01

- Added and enabled mod for `emencia-cookie-law`, close #32;
- Added and enabled mod for `django-icomoon`, close #31;
- Updated documentation, close #33
- Fixed `django-crispy-forms` mod settings for last release, updated to `crispy-forms-foundation==0.5.3`, #29;
- Added `reload` action to the Makefile, to restart the uwsgi instance on integration or production environment;

2.5.24 Version 0.7.3 - 2015/08/31

- Updated docs to add tips about *RVM Gemsets*;
- Fixed `django-reversion==1.8.7` for issue #27;
- Fixed *sitemap* mod `urls.py`, close #28;

2.5.25 Version 0.7.2 - 2015/06/13

- Added some cleaning when using 'make assets' command;
- Updated some scss, Enabled default icomoon webfont;
- Updated some docs;

2.5.26 Version 0.7.1 - 2015/06/06

- Fix some included html templates to use `<h1>` instead of `<h2>`, although Django apps templates probably all use `<h2>` again, so we will need to override them;

2.5.27 Version 0.7.0 - 2015/06/06

- Use `fonts_dir` setting in compass config, close #13
- Use *lazy protocole prefix* to load googlefont, close #12;
- Remove `<h1>` usage in topbar for a better semantic (`<h1>` should not be identical to `<title>`), **WARNING: now all cms page must define their own h1, also other app template have to define the right h1**;
- Get back our CMS snippet plugin, temporary using our fork as a develop source, close #19;
- Upgrade `django-admin-style` to `0.2.7`, close #18;
- Fix to `djangocms_text_ckeditor==2.4.3`, close #16;
- Include Slick.js, close #17;
- Remove Foundation Orbit usage because it is deprecated and Slick.js works better;
- `project/assets.py` is now processed by cookiecutter+Jinja so we can disable assets from user choices like for socialaggregator Javascript library;
- Reorganize SCSS sources:
  - `components/` directory is for page parts or specific Django apps layout;

  - `vendor/` directory contains all SCSS for included library (like mmenu, royalslider, etc..);
  - `utils/` directory contains all utils stuff like mixins, basic addons, Foundation patches, etc..;
  - Added Flexbox support;
- Remove interchange template for slideshows;
- Cleaning `app.js` since Orbit is not used anymore;

2.5.28 Version 0.6.6 - 2015/05/16

- Enforce `django-tagging==0.3.4` (to avoid a bug with django<=1.7);
- Review and update `assets.py`, close #10;
- Some assets cleanup, close #9;
  - Added missing default images for *Royal Slider*;
  - Removed Foundation3 Javascript stuff;
  - Cleaning main frontend script `app.js`;
  - Added MegaMenu stuff;
- Big update on `contact_form` app:
  - Fix print message on template;
  - Reorganise admin view;
  - Use `django-import-export` for exporting contact datas;
  - Don't print captcha on form when `settings.DEBUG` is `True`;

2.5.29 Version 0.6.5 - 2015/05/03

- Cleaning documentations;
- Restored doc stuff to automatically build mod documentations;
- Updated to `django-cms==3.0.13`;
- Enforce `django-contrib-comments==1.5.0` (to avoid a bug with django<=1.7);
- Integrated `django-logentry-admin` as a default enabled mod, close #8;
- Fixed doc config to get the right version number from git tags;

2.5.30 Version 0.6.1 - 2015/04/20

- Added cookiecutter context in `project/__init__.py` file;

2.5.31 Version 0.6.0 - 2015/04/19

- Better documentation;

2.5.32 Version 0.5.0 - 2015/04/17

- Enabled cms translation and some settings from cookiecutter context, close #4;

2.5.33 Version 0.4.0 - 2015/04/16

- Removed unused variables in `cookiecutter.json`;
- Changed ignored files from jinja to target some files to use as templates;
- Changed template for `skeleton.html` to remove occurences to not enabled apps;
- Added cookiecutter context usage to remove unused sitemap parts, close #5;
- Changed buildout.cfg to be more flexible without some enabled apps;

2.5.34 Version 0.3.0 - 2015/04/15

- Added Git repo initialization in the post generation hook;
- Added a message at the end of the post generation hook to display some help;
- Changed some variables from `cookiecutter.json` for repository infos;

2.5.35 Version 0.2.0 - 2015/04/13

- Added post generation hook to enable mods after install;
- Use cookiecutter context to remove eggs in `buildout.cfg` egg list;

2.5.36 Version 0.1.0 - 2015/04/12

- First version started from emencia_paste_djangocms_3 structure version `1.4.0`;
- Not ready to be used yet, it misses some things for now;