# Epaster Documentation

*Release 2.3.1*

David THENON

April 11, 2015

# Contents

**1 Introduction** **1**

**2 Structure** **3**

**3 Table of contents** **5**

3.1 Install . . . . . 5
3.2 Usage . . . . . 6
3.3 DjangoCMS 2.x paste . . . . . 9
3.4 DjangoCMS 3.x paste . . . . . 17
3.5 Common topics around Epaster . . . . . 29
3.6 Development . . . . . 33
3.7 History . . . . . 34

CHAPTER 1

# Introduction

Emencia uses the Epaster tool for web projects along with our techniques and procedures. It's mostly based on Python Paste and buildout to allow for the distribution of projects easy to install anywhere.

Its goal is to automatically create and initialize the project's structure so you don't lose time assembling the different parts.

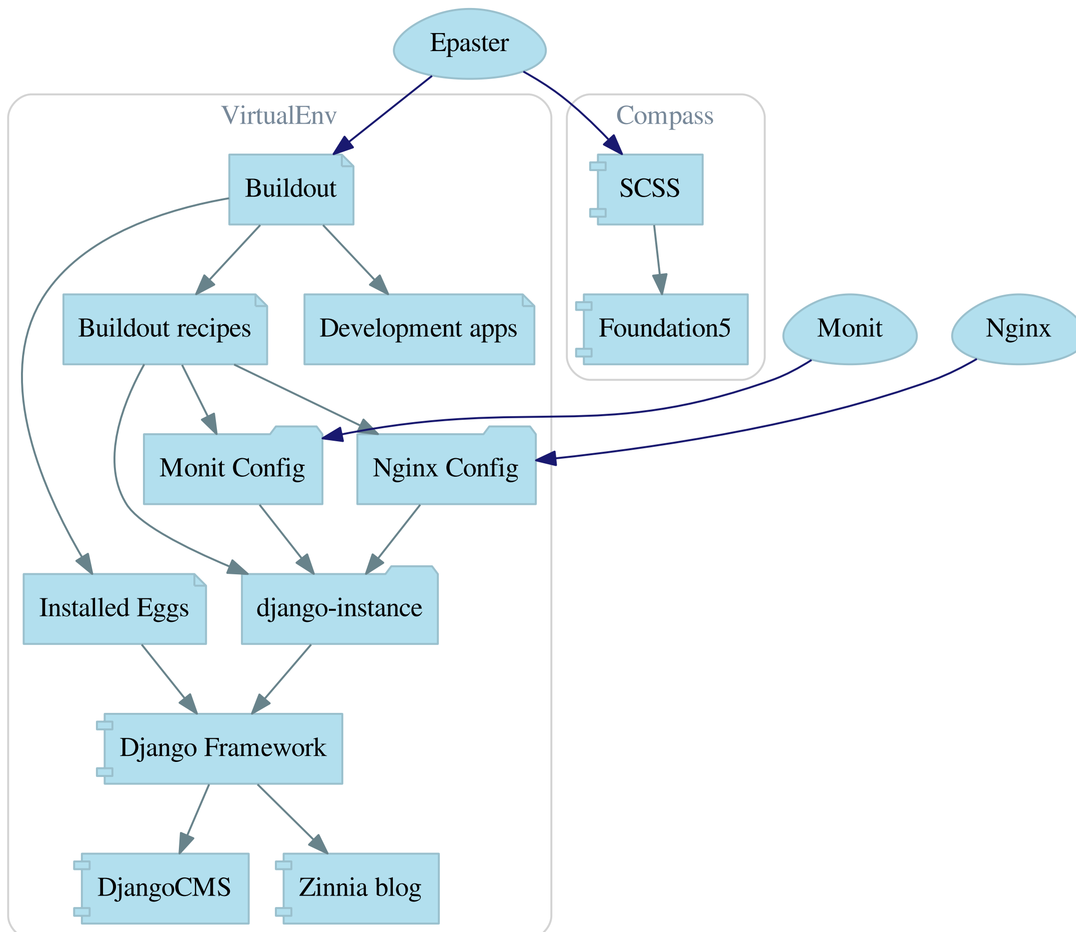
Epaster is not really a package, just a buildout project to assemble some apps to develop Python Paste templates (called a *paste*). In theory, you should be able to install these paste just with virtualenv and pip, but Epaster assemble all our paste in a unique buildout project.

For now, it is only used to build Django projects through some paste packages.

# CHAPTER 2

# Structure

Finally, Epaster will build you a project that is designed to be use with some software and components, below you can find a simple diagram to resume their interaction.



CHAPTER 3

# Table of contents

## 3.1 Install

This install procedure is designed for a virtual Python environment. The Epaster tool will not be installed in your system's Python environment to avoid conflicts or the crashing of Python modules in your system.

It is possible to install Epaster in your system's Python environment but you will need to skip the virtualenv stage, which may create a risk.

### 3.1.1 Requirements

The Python virtualenv module is required and must be installed on your system. We recommend you install it directly from pip to ensure you install a more recent version that the one in your system package.

A few devel libraries are required to correctly compile some modules within your buildout project :

- Python
- libpq (for **psycopg2**)
- python (for **psycopg2**)
- libjpeg (for **Pillow**)
- zlib (for **Pillow**)
- libfreetype (for **Pillow**)

Note that **psycopg2** is only required if you plan to use a PostgreSQL database instead of the default **sqlite3** database.

If you plan to build the documentation (in `docs` directory) you will have to install graphviz before on your system.

### 3.1.2 Procedure

When the required elements are installed, you will need to retrieve **Epaster** from its Github repository, install it and activate it:

```
git clone https://github.com/emencia/Epaster.git
make install
source bin/activate
```

This will downloads all dependencies and install them in the virtual environment. If an error occurs, the buildout process will stop and print out the problem. You can correct it and relaunch the buildout process that will continue from the previous job.

If the behavior seems uncertain, you can clean all the files installed and the directory using the dedicated Makefile feature:

```
make clean
```

When the buildout process is successfully completed, **Epaster** is ready and you can use it to create new projects.

Additionaly if you have to make some development on Epaster (and/or edit its documentation and rebuild it) or its Paste templates, you will have to install its development environment, after the basic install just do:

```
buildout -c development.cfg
```

### 3.1.3 Global config

You can set a global config where the default option will be used with Buildout. A common method is to create this global config with these lines:

```
[buildout]
download-cache = /home/django/.buildout-cache
```

The path defined in `download-cache` will be used to store downloaded packages. This is a cache to avoid re-downloading these packages every time you launch buildout. Note that you have to create this path beforehand or an error will occur in the buildout.

## 3.2 Usage

**Epaster** can only really be used within its virtual environment, so you must remember to enable it first:

```
cd /home/emencia/epaster
source bin/active
```

Then you can go the path where you want to create your project.

### 3.2.1 List of available project types

The following command:

```
paster create --list-templates
```

will display a list of available project types which you can create:

```
Available templates:
  basic_package:  A basic setuptools-enabled package
  django:         Django project
  paste_deploy:   A web application deployed through paste.deploy
```

This is just a sample, your install may have different paste.

### 3.2.2 Create a new project

The **Epaster** tool is an interactive command. When launched, some questions will be asked for the selection of components and options to be used within the project:

```
paster create -t django myproject
```

### 3.2.3 Install a new project

Once the project has been created by Buildout, it is autonomous of **Epaster** and you can move it wherever you want. This is the process we recommend (i.e., do not keep it under the Epaster tree).

So, for a newly created project called `myproject`, you will have to enter it in its directory and just execute the automatic install command from Makefile:

```
make install
```

This will install the virtual environment and all required packages using the default config `buildout.cfg`. When it's finished, active the virtual environment:

```
source bin/active
```

Then if you need to use a specific config, execute it as follows:

```
buildout -c production.cfg
```

Generally, the database type used is **sqlite3**, stored in a `database.sqlite3` file at the root directory of your project.

### 3.2.4 Makefile actions

A Makefile is shipped within a project to include some useful maintenance command actions:

- `help`: display this help list;
- `install`: to proceed with a new install of this project. Use clean command before if you want to reset a current install;
- `clean`: to clean your local repository of all the buildout and instance usage elements;
- `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 elements with compass;

It is only used from its location as follows.

You can use it with the following syntax:

```
make ACTION
```

Where `ACTION` is the command action to use, as follows:

```
make install
```

### 3.2.5 PO-Projects

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 it because it will be regenerated each time buildout is used.

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

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. The compile PO (`*.mo` files) are lost during this action and so each time you use this action you will have to recompile them.

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

```
po_projects pull
```

And probably 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.

### 3.2.6 Gestus

The Gestus client is pre-configured in all created projects, its config file is automatically generated (in `gestus.cfg`), don't edit it because it will be regenerated each time buildout is used.

You can register your environment with the following command :

```
gestus register
```

Remember this should only be used in integration or production environment and you will have to fill a correct accounts in the `EXTRANET` part.

### 3.2.7 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.

## 3.3 DjangoCMS 2.x paste

DjangoCMS 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, but they are not all enabled. You can enable or disable them, as needed.

It is always preferable to use the mods system to install new apps. You should never install a new app with pip. If you plan to integrate it into the project, always use the buildout system. Just open and edit the `buildout.cfg` file to add the new egg to be installed. For more details, read the buildout documentation.

**This paste is not really maintained anymore, you should prefer to see for the DjangoCMS 3.x version instead.**

### 3.3.1 Links

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

### 3.3.2 Paste

This paste will appear with the name `djangocms-2` in the paster templates list (with the `paster create --list-templates` command).

To use this paste to create a new project you will do something like :

```
paster create -t djangocms-2 myproject
```

### 3.3.3 Django

#### django-instance

This is the command installed to replace the `manage.py` script in Django. `django-instance` is aware of the installed eggs.

#### Paste template version

In your projects, you can find from which Paste template they have been builded in the 'project/__init__.py' file where you should find the used package name and its version.

Note that previously (before the Epaster version 1.8), this file was containing the Epaster version, not the Paste template one, since the package didn't exists yet.

#### How the Mods work

The advantage of centralizing app configurations in their mods is the project's `settings.py` and `urls.py` are gathered together in its configuration (cache, smtp, paths, BDD access, etc.). Furthermore, it is easier to enable or disable the apps.

To create a new mods, create a directory in `$PROJECT/mods_avalaible/` that contains at least one empty `__init__.py` and a `settings.py` to build the app in the project and potentially its settings. 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's initial settings and urls). N.B. With Django's `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**) in `$PROJECT/mods_enabled`. To disable it, simply delete the symbolic link.

### 3.3.4 Compass

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

By default, a Django project has its SCSS sources in the `compass/scss/` directory. The CSS Foundation framework is used as the database.

A recent install of Ruby and Compass is required first for this purpose (see RVM if your system installation is not up to date).

Once installed, you can then compile the sources on demand. Simply go to the `compass/` directory and launch this command:

```
compass compile
```

When you are working uninterruptedly on the sources, you can simply launch the following command:

```
compass watch
```

Compass will monitor the directory of sources and recompile the modified sources automatically.

By default the `compass/config.rb` configuration file (the equivalent of *settings.py*' in Django) is used. If needed, you can create another one and specify it to Compass in its command (for more details, see the documentation).

#### Foundation

This project embeds Foundation 5 sources installed from the Foundation app so you can update it from the sources if needed (and if you have installed the Foundation cli, see its documentation for more details). If you update it, you need to synchronize the updated sources in the project's static files using a command in the Makefile:

```
make syncf5
```

**You only have to do this when you want to synchronize the project's Foundation sources from the latest Foundation release. Commonly this is reserved for Epaster developers.**

This will update the Javascript sources in the static files, but make sure that it cleans the directory first. Never put your files in the `project/webapp_statics/js/foundation5` directory or they will be deleted. Be aware that the sources update will give you some file prefixed with a dot like `.gitignore`, you must rename all of them like this `+dot+gitignore`, yep the dot character have to be renamed to `+dot+`, else it will cause troubles with GIT and Epaster. There is a python script named `fix_dotted_filename.py` in the source directory, use it to automatically apply this renaming.

For the Foundation SCSS sources, no action is required; they are imported directly into the compass config.

The project also embeds Foundation 3 sources (they are used for some components in Django administration) but you don't have to worry about them.

#### RVM

rvm is somewhat like what virtualenv is to Python: a virtual environment. The difference is that it is intended for the parallel installation of a number of different versions of **Ruby** without mixing the gems (the **Ruby** application packages). In our scenario, it allows you to install a recent version of **Ruby** without affecting your system installation.

This is not required, just an usefull cheat to know when developing on a server with an old distribution.

### 3.3.5 Installation and initial use

Once your project has been created with this epaster template, you need to install it to use it. The process is simple. 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
```

You will then be able to access it at the following url (where `127.0.0.1` will be the server's IP address if you work on a remote machine) : `http://127.0.0.1:8001/`

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

### 3.3.6 Available mods

#### accounts

Enable Django registration and everything you need to allow users to request registration and to connect/disconnect. The views and forms are added so this part can be 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.

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 (to be configured in the 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.

#### admin_tools

Enable django-admin-tools to enhance the administration interface. This enables three widgets to customize certain elements. filebrowser is used, so if your project has not enabled it, you need to remove the occurrences of these widgets.

#### assets

Enable django-assets to combine and minify your *assets* (CSS, JS). The minification library used, *yuicompressor*, requires the installation of Java (the OpenJDK installed by default on most Linux systems is sufficient).

In general, this component 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.

#### ckeditor

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

Use "djangocms_text_ckeditor", a djangocms plugin to use CKEditor (4.x) instead of the default one

This mod contains some tricks to enable "django-filebrowser" usage with "image" plugin from CKEditor.

And some contained patches/fixes :

- the codemirror plugin that is not included in djangocms-text-ckeditor;
- Some missed images for the "showblocks" plugin;
- A system to use the "template" plugin (see views.EditorTemplatesListView for more usage details);
- Some patch/overwrites 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.

By default, it uses the recaptcha 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.

#### 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 mysterious bugs (like with syncdb or zinnia) so for the time being, this mods is disabled. So enable it locally for your needs, but never commit its enabled mod and remember to disable it when you have a strange bug.

#### 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.

### 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.

### pdb

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

N.B. Neither `django_pdb` nor `pdb` are installed by the 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:** django-pdb should be put at the end of settings.INSTALLED_APPS :

"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 assure 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*.

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*).

### 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.

#### socialaggregator

Enable the emencia-django-socialaggregator app to manage social contents.

This app require some API key settings to be filled to work correctly.

#### staticpages

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

#### urlsmap

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

#### zinnia

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

At the time of installation, an automatic patch (that can be viewed in the `patches/` directory) is applied to it to implement the use of ckeditor, which is enabled by default in its settings.

### 3.3.7 Changelogs

#### Version 1.9.8 - 2015/01/28

- Fix webassets bug: since we use Bundle names with version placeholder, webassets needed a manifest file to know what version to use in its templatetags. So now a `webassets.manifest` file is created in `project/webapp_statics` directory and will be copied to `project/static` dir when assets are deployed;

This will be the last maintenance release, don't expect any other update for this package.

### Version 1.9.7 - 2015/01/20

Changing default behavior of *Asset bundles* in `project/assets.py` so now bundle urls will be like `/static/screen.acefe50.css` instead of old behavior `/static/screen.min.css?acefe50` that was causing issue with old proxies caches (see webassets documentation);

You can safely backport this change to your old projects, this should be transparent to your install and won't require any server change.

### Version 1.9.6.1 - 2014/12/26

- Fix a damned bug with `bootstrap.py` that was forcing to upgrade to `setuptools=0.8` that seems to results with bad parsing on some constraints like the one from django-cms for `django-mptt==0.5.2,==0.6,==0.6.1` that was causing a buildout fail on conflict version. This has been fixed with updating to the last `bootstrap.py` and use its command line arguments to fix versions for `zc.buildout` and `setuptools` in the Makefile;

### Version 1.9.6 - 2014/11/17

- Mount 500 and 404 page view in urls.py when debug mode is activated;

### Version 1.9.5 - 2014/11/07

- Update to `zc.buildout==2.2.5`;
- Update to `buildout.recipe.uwsgi==0.0.24`;
- Update to `collective.recipe.cmd==0.9`;
- Update to `collective.recipe.template==1.11`;
- Update to `djangorecipe==1.10`;
- Update to `porticus==0.8.1`;
- Add package `cmsplugin-porticus==0.1.2` in buildout config;
- Remove dependancy for `zc.buildout` and `zc.recipe.egg`;

### Version 1.9.4 - 2014/11/02

Update mods doc

### Version 1.9.3 - 2014/11/01

Fix some app versions in version.cfg

### Version 1.9.2 - 2014/09/31

Following repository renaming (*emencia-paste-djangocms-2* to *emencia_paste_djangocms_2*) for a workaround with 'gp.vcsdevelop'

#### Version 1.9.1 - 2014/09/31

Fix paste template and setup

#### Version 1.9 - 2014/09/31

Renaming repository to *emencia-paste-djangocms-2* to follow Epaster new structure.

#### Version 1.8.2 - 2014/09/27

Add mods documentations taken from Epaster documentation.

#### Version 1.8 - 2014/09/26

First release as *emencia_paste_django* started from `Epaster==1.8`

## 3.4 DjangoCMS 3.x paste

DjangoCMS 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, but they are not all enabled. You can enable or disable them, as needed.

It is always preferable to use the mods system to install new apps. You should never install a new app with pip. If you plan to integrate it into the project, always use the buildout system. Just open and edit the `buildout.cfg` file to add the new egg to be installed. For more details, read the buildout documentation.

### 3.4.1 Links

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

### 3.4.2 Paste

This paste will appear with the name `djangocms-3` in the paster templates list (with the `paster create --list-templates` command).

To use this paste to create a new project you will do something like :

```
paster create -t djangocms-3 myproject
```

### 3.4.3 Django

#### django-instance

This is the command installed to replace the `manage.py` script in Django. `django-instance` is aware of the installed eggs.

#### Paste template version

In your projects, you can find from which Paste template they have been builded in the `project/__init__.py` file where you should find the used package name and its version. So you can easily see the version doing something like :

```
cat project/__init__.py
```

Note that previously (before the Epaster version 1.8), this file was containing the Epaster version, not the Paste template one, since the package didn't exists yet.

#### How the Mods work

The advantage of centralizing app configurations in their mods is the project's `settings.py` and `urls.py` are gathered together in its configuration (cache, smtp, paths, BDD access, etc.). Furthermore, it is easier to enable or disable the apps.

To create a new mods, create a directory in `$PROJECT/mods_avalaible/` that contains at least one empty `__init__.py` and a `settings.py` to build the app in the project and potentially its settings. 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's initial settings and urls). N.B. With Django's `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**) in `$PROJECT/mods_enabled`. To disable it, simply delete the symbolic link.

### 3.4.4 Installation and initial use

Once your project has been created with this epaster template, you need to install it to use it. The process is simple. 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 (because they have also this alias linked to their own network card).

---

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

### 3.4.5 Available mods

#### accounts

Enable Django registration and everything you need to allow users to request registration and to connect/disconnect. The views and forms are added so this part can be 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.

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 (to be configured in the 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 use a dummy profile model linked to User object. This profile is dummy because it implement fields for sample but you may not need all of them or you can even may not need about a Profile model, the User object could be enough for your needs. So before to use the syncdb, be sure to watch for the model to change it, then apply your changes to `forms.RegistrationFormAccounts`, `views.RegistrationView` and eventually templates.

#### admin_style

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

*admin-style* better fit with DjangoCMS than admin_tools.

> **Warning:** This mod cannot live with admin_tools, you have to choose only one of them.

#### admin_tools

Enable django-admin-tools to enhance the administration interface. This enables three widgets to customize certain elements. filebrowser is used, so if your project has not enabled it, you need to remove the occurrences of these widgets.

> **Warning:** This mod cannot live with admin_style, you have to choose only one of them.

#### assets

Enable django-assets to combine and minify your *assets* (CSS, JS). The minification library used, *yuicompressor*, requires the installation of Java (the OpenJDK installed by default on most Linux systems is sufficient).

In general, this component 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 `asset.py` file is divised in three parts :

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

### 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.

By default, it uses the recaptcha 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.

#### 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 painful to manage files with Filebrowser and django-filer 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.

### pdb

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

N.B. Neither `django_pdb` nor `pdb` are installed by the 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:** django-pdb should be put at the end of settings.INSTALLED_APPS :

"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 assure 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*.

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 webserver (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.

#### socialaggregator

Enable the emencia-django-socialaggregator app to manage social contents.

---

**Note:** This app require some API key settings to be filled to work correctly.

---

#### 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.

#### 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.

### 3.4.6 Changelogs

#### version 1.4.0 - 2015/04/12

- Enforce python2.7 usage into Makefile (to avoid a bug with MacOSX);
- Update to `django==1.6.11`;
- Update to `django-cms==3.0.12`;
- Enable a default robots.txt in default and integration environments so development sites won't never be referenced;
- Enforce to `mptt==0.6.1` to avoid a but third tier apps (like django-tagging) that accept superior versions not compatible with cms;

#### version 1.3.8 - 2015/02/27

- Add conf for sentry tracking in production env;
- Fix bug into Makefile template;

#### Version 1.3.7 - 2015/02/26

- Fix Makefile's 'install' action so this will works on all systems (OSX included) with a shell and Python2;

### Version 1.3.6 - 2015/02/25

- Fix rst typo into README file;
- Remove project's apps locale dirs, close #7;
- Fix missing `django_comments` in `settings.INSTALLED_APPS`, required by zinnia else it cause a bug on some admin views, close #9;
- Update to `djangocms==3.0.10`;
- Update to `crispy-forms-foundation==0.4.1`;
- Update to `djangocms-admin-style==0.2.5`;

### Version 1.3.5 - 2015/02/06

- Use new options `dump_other_apps` and `exclude_apps` from emencia-recipe-drdump/drdump packages;
- Add 2 new commands in makefile for export/import project data (database + media)

### Version 1.3.4 - 2015/02/03

- Force Python2.x usage in virtual environment from the Makefile because actually a lot of used apps can't works with Python3 and some distributions allready use Python3 as the default Python interpreter;

### Version 1.3.3 - 2015/01/29

- Use get_civility_display into contact_form app's email template rather civility;

### Version 1.3.2 - 2015/01/28

- Comment settings.ADMINS so we are not sending anymore Django's mail alerts to @dummy.com..;
- Fix webassets bug: since we use Bundle names with version placeholder, webassets needed a manifest file to know what version to use in its templatetags. So now a `webassets.manifest` file is created in `project/webapp_statics` directory and will be copied to `project/static` dir when assets are deployed;

### Version 1.3.1 - 2015/01/28

- Fix a bug in project/contact_form/cms_app that was using the wrong hook name;
- Remove sample patch for Django and unknown locales because since 1.6, Django does not care about known or unknown locale;
- Disable 'sitemap.xml' mapping to a static files in the nginx config since we have a mod to generate it automatically from enabled apps;

### Version 1.3.0 - 2015/01/28

- Update to `django-filer==0.9.9` to fix a bug with `setuptools>=7` (this should permits soon to remove freezing to setuptools==7 and pip==1.5.x);
- Remove "syncf5" action in Makefile because now it resides in a Makefile into foundation5's sources;

### Version 1.2.9 - 2015/01/20

Changing default behavior of *Asset bundles* in `project/assets.py` so now bundle urls will be like `/static/screen.acefe50.css` instead of old behavior `/static/screen.min.css?acefe50` that was causing issue with old proxies caches (see webassets documentation);

You can safely backport this change to your old projects, this should be transparent to your install and won't require any server change.

### Version 1.2.8 - 2015/01/14

- Update to `django==1.6.10`;
- Update to `django-cms==3.0.9`;
- Fix default slideshow template with a bad html id;
- Add a Makefile in foundation5 sources, move syncf5 action into it and add a syncjquery to fix compressed jquery in foundation5 vendor sources that was causing issue with compressed assets;
- Add CMS apphook sample for contact_form;

### Version 1.2.7 - 2015/01/06

- Update to `django==1.6.9`;
- Update to `django-cms==3.0.7`;
- Update to `Pillow==2.7.0`;
- In buildout config, remove the old patch hack to add unsupported locales from Django, since Django 1.6 does not care anymore;

### Version 1.2.6 - 2014/12/26

- Fix a damned bug with `bootstrap.py` that was forcing to upgrade to `setuptools=0.8` that seems to results with bad parsing on some constraints like the one from django-cms for `django-mptt==0.5.2,==0.6,==0.6.1` that was causing a buildout fail on conflict version. This has been fixed with updating to the last `bootstrap.py` and use its command line arguments to fix versions for `zc.buildout` and `setuptools` in the Makefile;

### Version 1.2.5 - 2014/12/25

- Add config for emencia-recipe-drdump recipe for Dr Dump;

### Version 1.2.4 - 2014/12/19

- Add Foundation's *kitchen sink* in a staticpage within `project/templates/prototypes/foundation5.html` and mounted on `/prototypes/foundation5.html`;
- Add template tag library named `utils_addons` in `project/utils/templatetags/`;
- Add `split` filter in `utils_addons` template tag library;
- Add nginx conf for admin with timeout and max body size increase;

### Version 1.2.3 - 2014/12/02

- Improve `sitemap` mod, more modular and usefull;
- Add `filer` and `thumbnails` mod, ususally not used in our projects but it could be usefull for some specific goals;
- Fix contact_form app that was missing its `sitemap.py` file;
- Update to `crispy-forms-foundation==0.4`;
- DjangoCMS page templates has moved from `project/templates/cms` to `project/templates/pages`, following a recommandation from DjangoCMS' documentation;
- Add `menu/menu_sidenav.html` and `pages/2_cols.autonav.html` templates to have a template with deep menu for current root page;
- Update to `porticus==0.9.6`;
- Update to `emencia-django-slideshows==0.9.4`;

### Version 1.2.2 - 2014/11/24

- Add `sendfile` mod;
- Add *client_max_body_size* sample directive usage in nginx template (but commented);
- Add commented location */protected_medias* to demonstrate sendfile mod usage within nginx template;

### Version 1.2.1 - 2014/11/24

- Update to Foundation 5.4.7;

### Version 1.2 - 2014/11/19

- Refactoring Template code to open a new way for a much modular behavior, should not break anything;

### Version 1.1.3 - 2014/11/17

- Mount 500 and 404 page view in urls.py when debug mode is activated;

### Version 1.1.2 - 2014/11/16

- Fix a bug with symlinks that was not packaged and so was missing from the installed egg, this close #1, thanks to @ilanouh;
- Add missing gitignore rule to ignore debug_toolbar mod (it must never be installed from the start because it causes issues with cms and the syncdb command);

### Version 1.1.1 - 2014/11/07

- Update to `zc.buildout==2.2.5`;
- Update to `buildout.recipe.uwsgi==0.0.24`;
- Update to `collective.recipe.cmd==0.9`;
- Update to `collective.recipe.template==1.11`;
- Update to `djangorecipe==1.10`;
- Update to `porticus==0.9.5`;
- Add package `cmsplugin-porticus==0.2` in buildout config;
- Remove dependancy for `zc.buildout` and `zc.recipe.egg`;

### Version 1.1 - 2014/11/03

- Update to `zc.buildout==2.2.4` to fix a bug introduced in 2.2.3;
- Update to last `bootstrap.py` script;
- Remove Foundation3 sources, CSS and bundles, they are not used anymore;
- Move ckeditor and minimalist CSS to common SCSS sources with Foundation5;
- Update Compass README;
- Correct admin_style Compass config;
- Add 'ar' country to the CSS flags;
- Recompile all CSS in project's webapp_statics;
- Changing `assets.py` to use nested bundles, so we can separate app bundles (foundation, royalslider, etc..) from the main bundles where we load the app bundles;
- Main frontend's CSS & JS bundles are now called `main.css` and `main.js` not anymore `app.***` (yes we use the old Foundation3 ones that have been removed);

### Version 1.0.4 - 2014/11/03

Update mods doc

### Version 1.0.3 - 2014/11/03

Fix some app versions in version.cfg, fix app.js to use socialaggregator only if its lib is loaded.

### Version 1.0.2 - 2014/11/03

Remove all enabled mods because it's the template responsability to enabled them or not.

### Version 1.0.1 - 2014/11/03

Following repository renaming for a workaround with 'gp.vcsdevelop'.

#### Version 1.0 - 2014/11/03

First commit started from emencia-paste-djangocms-2 == 1.9.1 and merged with buildout_cms3 repository, bump to 1.0

## 3.5 Common topics around Epaster

There are some topics around some things we use in Epaster environnment.

Note that these topics are based on **emencia_paste_djangocms_3**, but generally they will apply also to **emencia_paste_djangocms_2**.

### 3.5.1 Compass

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

By default, a Django project has its SCSS sources in the `compass/scss/` directory. The CSS Foundation framework is used as the database.

A recent install of Ruby and Compass is required first for this purpose (see RVM if your system installation is not up to date).

Once installed, you can then compile the sources on demand. Simply go to the `compass/` directory and launch this command:

```
compass compile
```

When you are working uninterruptedly on the sources, you can simply launch the following command:

```
compass watch
```

Compass will monitor the directory of sources and recompile the modified sources automatically.

By default the `compass/config.rb` configuration file (the equivalent of *settings.py*‘ in Django) is used. If needed, you can create another one and specify it to Compass in its command (for more details, see the documentation).

#### RVM

rvm is somewhat like what virtualenv is to Python: a virtual environment. The difference is that it is intended for the parallel installation of a number of different versions of **Ruby** without mixing the gems (the **Ruby** application packages). In our scenario, it allows you to install a recent version of **Ruby** without affecting your system installation.

This is not required, just an usefull cheat to know when developing on a server with an old distribution.

### 3.5.2 Webfonts

Often, we use webfonts to display icons instead of images, because a webfont is more flexible to use (it can take any size without to re-upload it) and more light on file size. It is also more *CSS friendly*.

Commonly we use icomoon that is a service to pack a selected set of webfonts to a ZIP archive that you can use to easily embed it in your project.

The first thing is to go on icomoon, create a webfont project and select the needed item from fonts. Then you have a webfont project, you have to download it as a ZIP archive and open it when it’s done.

When you open the archive, you should something like that :

```
icomoon/
-- demo-files
|   -- demo.css
|   -- demo.js
-- demo.html
-- fonts
|   -- icomoon.eot
|   -- icomoon.svg
|   -- icomoon.ttf
|   -- icomoon.woff
-- Read Me.txt
-- selection.json
-- style.css
```

What we need here is the `fonts` directory because it contains the font we need to put in our project assets, and the `style.css` file that contain the icons class name *map*.

So for a site project named **site_sample** generated from Epaster, first you will copy the fonts directory in `project/webapp_statics` into your project, there should allready be a `fonts` directory so overwrite it.

Now open the `style.css` from the archive, it should look like this :

```
@font-face {
        font-family: 'icomoon';
        src:url('fonts/icomoon.eot?n45w4u');
        src:url('fonts/icomoon.eot?#iefixn45w4u') format('embedded-opentype'),
                url('fonts/icomoon.woff?n45w4u') format('woff'),
                url('fonts/icomoon.ttf?n45w4u') format('truetype'),
                url('fonts/icomoon.svg?n45w4u#icomoon') format('svg');
        font-weight: normal;
        font-style: normal;
}
[class^="icon-"], [class*=" icon-"] {
        font-family: 'icomoon';
        speak: none;
        font-style: normal;
        font-weight: normal;
        font-variant: normal;
        text-transform: none;
        line-height: 1;

        /* Better Font Rendering =========== */
        -webkit-font-smoothing: antialiased;
        -moz-osx-font-smoothing: grayscale;
}


.icon-left:before {
        content: "\e622";
}
.icon-right:before {
        content: "\e623";
}
.icon-play:before {
        content: "\e62b";
}
```

Not that there are two parts, the first with `@font-face` and `[class^="icon-"], [class*=" icon-"]`, and the second part with some icon class names. Don't mind about the first part, we allready define it in our SCSS

component, just copy the whole second part with all class names for your icons.

Then you will have to fill the class names used in the SCSS components `compass/scss/components/_icomoon.scss` in your project, search for this pattern at the end of the file :

```
// Icon list
/*
*
* HERE GOES THE ICONS FROM THE style.css bundled in the icomoon archive
*
*/
```

And put the pasted icon class names after this pattern.

Finally in `compass/scss/app.scss` search for the line containing `@import "components/icomoon";` and uncomment it, now you can compile your SCSS and the webfont icons will be available from your `app.css` file.

### 3.5.3 Assets management

#### Why

In the past, assets management was painful with some projects, because their includes was often divided in many different templates. This causing issues like to update some library or retrieve effective code that was working on some template by inherit.

Also, this often results in pages loading dozen of asset files and sometime much more. This is a really bad behavior because it slows page loading and add useless performance charge on the web server.

This is why we use an **asset manager** within Epaster: 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

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

What does this command :

1. Collecting all static files from your project and installed apps to your `settings.STATIC_ROOT` directory;
2. Use django-assets to *compile* all defined bundles using previously collected files;
3. 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 conflict between development and production.

### The rule

Never, ever, put CSS stylesheets in your templates, NEVER. You can forget it, this will go in production and forgeted for a long time, 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.

## 3.5.4 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.

## 3.6 Development

For Epaster (or its pastes) development, you will need to install the Epaster repository, then use the development environnment :

```
git clone https://github.com/emencia/Epaster.git
make install
source bin/activate
buildout -c development.cfg
```

It will requires that you have installed the GIT client because the development environnment installs the pastes sources in the `src/` directory. This is where you will work.

### 3.6.1 Documentation

The documentation sources lives in the `docs/` as a Sphinx project, Sphinx is installed within the development environnment.

Note that for each paste the *mods* documentation is automatically builded from the paste. For example with the `emencia_paste_djangocms_3` paste, a `docs/emencia_paste_djangocms_3.rst` file is builded from its source code.

The paste's mods documentation is not builded from the common `make html` Sphinx command, but with the dedicated command : `make grab`.

The *grab* action will take it's base document from `project/mods_available/__init__.py` docstring then replace the directive `.. document-mods::` with the grabbed mods documentation.

The grabbed mods documentation itself is taken from each mod living in `project/mods_available/` using their `__init__.py` docstring, then they are assembled as an unique string that will replace the `.. document-mods::` directive.

**Beware** that all these `__init__.py` **docstrings must be valid RST syntax** else it will break the documentation building.

Finally to ease the documentation building, when you did lot of changes in the mods documentation, just use the following command to rebuild their docs then build the whole project documentation :

```
make all
```

This command assemble the `make grab` and `make html` commands.

### 3.6.2 Symlinks

You can't include symlinks into your paste templates, because Distribute ignore them, they won't be packaged and so won't be available in the paste's installed egg.

If you need to create some symlinks in the projects to build, you will have to do it in the paste template in `templates.py`. The `emencia_paste_djangocms_3` paste has generic way to do this, just append a tuple to the list `emencia_paste_djangocms_3.templates.Django.symlink_list` where the tuple contains the *target* (a relative path to the file/directory to link to) and the symlink file to create (an absolute path into the project to build).

### 3.6.3 Foundation updates

> **Warning:** To do this, you must have a strong knowing of Foundation sources structure
> You only have to do this when you want to synchronize the project's Foundation sources from the latest Foundation release. This is reserved for Epaster maintainers.

This project embeds Foundation 5 sources installed from the Foundation app so you can update it from the sources if needed (and if you have installed the Foundation cli, see its documentation for more details). to update these sources go into its directory and use it Makefile action:

```
make update
```

> **Warning:** Never manually put your files in the `project/webapp_statics/js/foundation5` directory or they will be deleted.

Then you have to checkup that directories structure has not changed, if it so you must fix the `syncf5` and `syncjquery` actions. When it's done, do:

```
make syncf5
```

For the SCSS sources, no action is required; they are imported directly into the compass config.

## 3.7 History

### 3.7.1 Changelog

#### Version 2.3.1 - 2015/04/12

- Enforce python2.7 usage into Makefile (to avoid a bug with MacOSX);
- Update to `emencia_paste_djangocms_3==1.4.0`;

### Version 2.3.0 - 2015/02/27

- Update to `emencia_paste_djangocms_3==1.3.8`;

### Version 2.2.9 - 2015/02/26

- Fix Makefile's 'install' action so this will works on all systems (OSX included) with a shell and Python2;
- Update to `emencia_paste_djangocms_3==1.3.7`;

### Version 2.2.8 - 2015/02/25

- Update to `emencia_paste_djangocms_3==1.3.6`;

### Version 2.2.7 - 2015/02/07

- Update to `emencia_paste_djangocms_3==1.3.5`;

### Version 2.2.6 - 2015/02/03

- Force Python2.x usage in virtual environment from the Makefile because actually a lot of used apps can't works with Python3 and some distributions allready use Python3 as the default Python interpreter;
- Update to `emencia_paste_djangocms_3==1.3.4`;

### Version 2.2.5 - 2015/01/29

- Update to `emencia_paste_djangocms_3==1.3.3`;

### Version 2.2.4 - 2015/01/28

- Update to `emencia_paste_djangocms_2==1.9.8`;
- Update to `emencia_paste_djangocms_3==1.3.2`;

### Version 2.2.3 - 2015/01/28

- Update to `emencia_paste_djangocms_3==1.3.1`;

### Version 2.2.2 - 2015/01/28

- Update to `emencia_paste_djangocms_3==1.3.0`;

### Version 2.2.1 - 2015/01/20

- Update to `emencia_paste_djangocms_2==1.9.7`;
- Update to `emencia_paste_djangocms_3==1.2.9`;

### Version 2.2.0 - 2015/01/14

- Update to `emencia_paste_djangocms_3==1.2.8`;

### Version 2.1.9 - 2015/01/06

- Update to `emencia_paste_djangocms_3==1.2.7`;

### Version 2.1.8.1 - 2014/12/26

- Fix: Forgotted to update `bootstrap.py` in previous version;

### Version 2.1.8 - 2014/12/26

- Update to `emencia_paste_djangocms_2==1.9.6.1`;
- Update to `emencia_paste_djangocms_3==1.2.6`;
- Backport fix from them to the Epaster Makefile to avoid any bugs;

### Version 2.1.7 - 2014/12/25

- Update to `emencia_paste_djangocms_3==1.2.5`;
- Update documentation to add some informations about *Dr Dump* in 'Usage' document;

### Version 2.1.6 - 2014/12/19

- Update to `emencia_paste_djangocms_3==1.2.4`;
- Update documentation to rename the tips section as the topics section, then improve it a little bit;

### Version 2.1.5 - 2014/12/01

- Update to `emencia_paste_djangocms_3==1.2.3`;
- Update documentation to add a *Tips* section;

### Version 2.1.4 - 2014/11/25

- Update to `emencia_paste_djangocms_3==1.2.2`;
- Fix "version.cfg";
- Update documentation;

### Version 2.1.3 - 2014/11/17

- Update to `emencia_paste_djangocms_2==1.9.6`;
- Update to `emencia_paste_djangocms_3==1.1.3`;

### Version 2.1.2 - 2014/11/16

- Update to `emencia_paste_djangocms_3==1.1.2`;
- Add "Development" notes in the docs;
- Update documentation;

### Version 2.1.1 - 2014/11/07

- Update to `zc.buildout==2.2.5`;
- Update to `emencia_paste_djangocms_2==1.9.5`;
- Update to `emencia_paste_djangocms_3==1.1.1`;
- Update documentation;

### Version 2.1 - 2014/11/03

- Update to `zc.buildout==2.2.4` to fix a bug introduced in 2.2.3;
- Update to last `bootstrap.py` script;
- Update to `emencia_paste_djangocms_3==1.1`;

### Version 2.0 - 2014/11/02

- Implement new pastes for djangocms 2.x and 3.x
- Update doc to fit to the new structure

### Version 1.8.2 - 2014/09/27

- Update docs to get the mods documentation directly from their docstring (in their __init__.py);
- Add eggedpy build part;

### Version 1.8.1 - 2014/09/26

- Add Development environnment, close #2;
- Try to fix 'Doc compile fail on rtd', fix #1;

### Version 1.8 - 2014/09/25

First public release on Github, there has been some changes to split Epaster from its Django project template, the template and its sources now resides in its own package named "emencia-paste-django". Both of them starts from the 1.8 version for history purpose.

### Version 1.7 - 2014/09/24

- Fix nginx template;
- Moving common apps from 'apps' dir to 'project';
- Some minor changes before going public on Github;
- This is the last version from our internal and private repository before Epaster goes public on Github, previous changelog is keeped here for history although you can't access to these previous versions;

### Version 1.6 - 2014/08/02

- Update to Foundation 5.3.3;
- Improve documentation by using Sphinx theme Bootstrap with 'yeti' bootswatch theme and add History page;
- Add a structure diagram in introduction (warning this will require to install graphviz on your system);

### Version 1.5 - 2014/07/28

- Update to Foundation 5.3.1;
- Update README for last changes and to use the version from `git describe --tags`;

### Version 1.4 - 2014/07/27

- Update to last Gestus & Po-projects clients;
- Add emencia-django-staticpages package and 'staticpages' mod to replace 'prototypes' mod;
- Add 'sitemap' mod;
- Fix Gestus config with Jinja2 template syntax;
- Use now a template recipe that use jinja and improve the nginx conf;