Epaster Documentation

Release 2.0.1

David THENON

November 03, 2014

Contents

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.

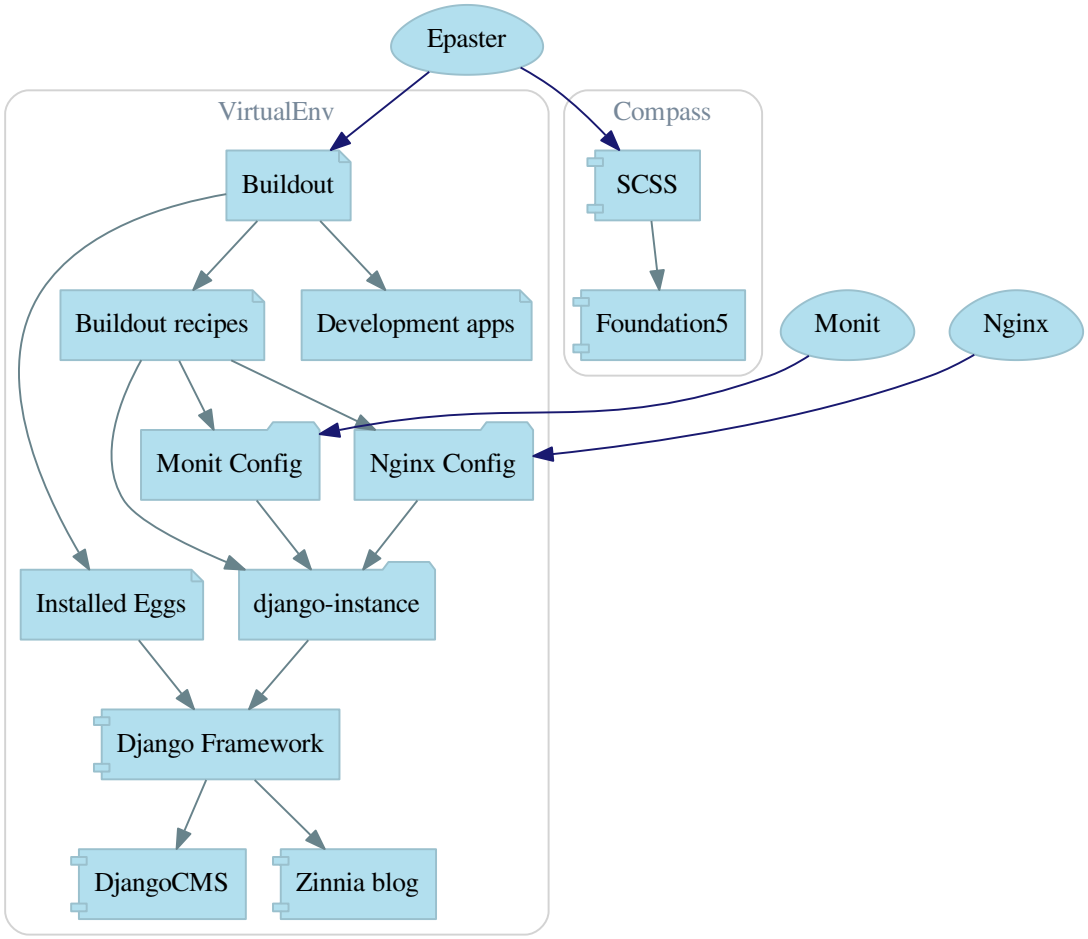
Epaster is not really a package, just a buildout project to assemble some apps to develop Python Paste templates (called a *paste*). In theory, you should be able to install these paste just with virtualenv and pip, but Epaster assemble all our paste in a unique buildout project.

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 git@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;
- `syncf5`: to synchronize required Javascript files from foundation5 sources dir to the project static files;

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 Gestus

The Gestus client is embedded in all created projects, its config is automatically generated (in `gestus.cfg`). 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.

3.3 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.

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.

3.4 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.

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 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.

3.4.5 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
```

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`.

3.4.6 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.

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.

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 use "django_ckeditor" that ship the same ckeditor but without cms addons.

This mod contains configuration for all of them.

And some contained patches/fixes :

- the codemirror plugin that is not included in any of the 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.

---

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.

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.

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*.

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.

3.5 History

3.5.1 Changelog

Version 2.1 - 03/11/2014

- 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`;

emencia_paste_djangocms_3 version 1.1

- 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 2.0 - 02/11/2014

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

Version 1.8.2 - 27/09/2014

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

Version 1.8.1 - 26/09/2014

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

Version 1.8 - 25/09/2014

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 - 24/09/2014

- 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 - 08/02/2014

- 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 - 07/28/2014

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

Version 1.4 - 07/27/2014

- 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;