Généralités sur l'application

Contexte du projet/Outils

Cette application a été redéveloppée à partir d'un prototype (v0) utilisant jext/geoext/openlayers/php et htaccess pour l'authentification. Il a été proposé de faire évoluer ce prototype afin de faciliter sa mise en ligne et les développements futurs. Tous les outils utilisés dans le projet sont gratuits et opensource.

La version (v1) proposée utilise:

- le framework php Symfony2:] basé sur le standard de Modèle Vue Controller MVC >Les « distributions » Symfony2 sont des applications entièrement fonctionnelles qui incluent les bibliothèques du coeur de Symfony2, une sélection de bundles utiles, une arborescence pratique et une configuration par défaut. Une distribution Symfony2 est un squelette d'application qui peut être immédiatement utilisé pour commencer à développer. > Les bundles ont chacun un rôle précis. Ces bundles sont activables en fonction du besoin et permettent ensemble de déployer une application web complète, en passant par l'authentification, la sécurité, l'administration de données, la construction dynamique de pages, la mise en place de web services, ... * la librairie FOSUserBundle qui permet d'implémenter un système d'authentification dynamique ([[http://sonata-project.org|SONATA gérant l'interface de connexion et faisant appel à la librairie FOS pour la gestion des utilisateurs).

> Le système d'autorisation donne la possibilité de gérer une liste de permissions (rôles) attribuables aux utilisateurs et/ou aux groupes. Ces permissions définissent l'accès à des fonctionnalités à tous les niveaux de l'application. **Symfony2** propose de hiérarchiser ces permissions ce qui offre une grande flexibilité dans la gestion des droits d'accès

- le moteur de template Twig

> Twig est le moteur de template suggéré par Symfony2. Il permet de séparer la présentation du reste de l'application. Twig est responsable du rendu d'une page HTML.

- le framework CSS/JS Bootstrap permet, entre autres choses, que le rendu de l'interface s'adapte à différents moniteurs (responsive web design). L'extension Less est utilisée pour compiler le CSS.
- l'ORM **Doctrine 2**

> L'ORM est utilisé pour gérer le modèle de l'application. ORM pour object-relational mapping est une technique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé. Il apporte une couche d'abstraction à la base de données qui en simplifie grandement le requêtage. L'ORM donne aussi la possibilité de changer facilement de base de données.

- La librairie **Python Fabric**

> Frabric est une librairie utilisée en ligne de commande pour automatiser les routines de l'administrateur (commandes présentes dans le fabfile).

- et toujours **geoext**, **javascript**... l'utilisation de jext à été abandonnée au profit de php.

Par ailleurs l'appel aux différentes librairies javascript et css se fait à travers des CDN:] (Content Delivery Network). Cela permet de ne pas avoir à installer en dur les librairies, il suffit de copier le lien dans le code de l'application (en revanche certaines fonctionnalités de l'application ne sont pas disponible en mode déconnecté). D'autres outils ont été utilisées par les développeurs tout au long du projet: - [[http://www.redmine.org/|Redmine application web libre de gestion de projet
- Git logiciel de gestion de version décentralisé
- Composer gestionnaire de dépendances pour php (gère les librairies serveur)
- CBNJS C'est un content delivery network qui gère les librairies javascript. Il est appellé dans un fichier de template à travers des liens url.

> Un content delivery network (CDN) est constitué d'ordinateurs reliés en réseau à travers Internet et qui coopèrent afin de mettre à disposition du contenu ou des données (généralement du contenu multimédia volumineux) à des utilisateurs.

La version (v2) proposée:

Elle utilise les mêmes librairies/outil que la V1. La partie SIFLore (NewSIFLoreBundle) a évolué. La partie SIVeg a été intégrée (NewSIVegBundle).

Arborescence du projet sous Symfony

- \app –> concerne l'application globale. Par exemple le fichier de configuration globale. Rassemble également les caches (calculés par twig) et les log (en fonction des canaux)
- \app\config –> toutes les configurations de Symfony

> C'est dans app/config/config.yml que sont stockés par exemple les paramètres de la base de donnée (connexions à la base de donnée). Il existe un fichier config par environnement.
>
> Il y a aussi les fichiers routing.yml (c'est là où on liste les url relatifs pour accéder aux ressources, par exemple l'url de la page siflore et c'est là aussi que l'on va gérer les permaliens)
>
> Fichier security.yml avec 3 notions: group/user/role (permissions). Dans security.yml on trouve aussi les firewall (par exemple on peut mettre le compte admin derrière un firewall afin qu'on soit toujours redirigé vers la page d'authentification si on copie l'url). Le fichier parameters gère le mdp de la BD principale. Les fichiers parameters.dist et CBN.dist sont des fichiers exemples. Le fichier security gère la hiérarchie des rôles et permissions.

- bin –>binaries, ce sont les executables. Ancien répertoire qui n'est plus utilisé (gardé pour assurer une compatibilité descendante). Vide dans notre cas.
- src –> sources du projet.

> Tout ce qui a été développé dans le projet, ce qui est spécifique, par exemple le javascript

- src\Fcbn : développements spécifiques organisés en Bundles :
- src\Fcbn\NewSIFloreBundle : SIFlore, interface du SIFlore http://siflore.fcbn.fr
- src\Fcbn\NewSIVegBundle : SIVeg, accessible a travers l'interface du SIFlore http://siflore.fcbn.fr/siveg
- src\Fcbn\SIFloreBundle : Toute première maquette du SIFlore qui n'est plus accessible à travers la plateforme (point d'entré commenté dans le fichier routing.yml)
- src\Fcbn\...\Controller : gestion des contrôleurs de requête (modèle MVC) – côté serveur
- src\Fcbn\...\Fixtures : gestion des couches de données
- src\Fcbn\...\Entity : description des modèles de table de la base de données utilisés par doctrine pour alimenter la bdd propre à l'application(ex: MapLayers.php contient les informations des couches cartographique de fond)
- src\Fcbn\...\Service : regroupes les classes avec un but précis – côté serveur
- src\Application : Bundles Sonata (Bundles spécifique orienté CMS)
- vendor –>librairies utilisées par le projet (ex: fos, sonata, swiftmailer, symphony, doctrine...) ATTENTION : NE JAMAIS MODIFIER VENDOR DIRECTEMENT
- web –> contient les fichiers assets globaux qui concernent tout le projet(css, javascript...) par ex: bootstrap (NB : tous les assets installés de src/ sont copiés dans web/ ils sont rafraichis avec la commande php app/console assetic:dump)

SIFlore et SIVeg

Philosophie générale

SIFlore et SIVeg sont contenus dans deux bundles distincts au sein du dossier src (NewSIFloreBundle et NewSIVegBundle). L'url principal http://siflore.fcbn.fr renvoi vers l'index.html.twig de SIFlore (grâce au routing.yml). Dans la page d'accueil de SIFlore un lien est fait vers l'interface SIVeg qui possède son propre index.html.twig.

Les deux outils font appel à deux bases de données:

- une base de donnée propre à l'application qui est générée par Doctrine et qui contient les tables FOS pour la gestion des utilisateurs ainsi que la table maplayers qui contient les id, label et liens vers les fonds cartos WMS utilisés par l'application
- une base de donnée siflore_national_v3 qui est externe à l'application (le bundle SIFlore utilise les schémas observation, observation_reun et exploitation, tandis que le bundle siveg utilise les schémas observation_veg et exploitation_veg).

Paramètres (URL/permaliens)

!{height: 200px; float: right}parameters.png!

Les paramètres du formulaire principal sont passés dans l'URL:

Paramètre	**Variable**	**Valeurs**
Taxon	cd_ref	
Région	r	**metro**, reun

Paramètre	Variable	Valeurs
Taxon Options	to	**1**, 2
Synthèse Options	so	**1**, 2, 3

Les paramètres sont les mêmes pour les deux bundles: SIFlore et SIVeg (pour des raisons pratiques de développement le paramètre Taxon s'intitule aussi comme ça pour les syntaxons) Les valeurs par défaut (en gras) sont utilisées si le paramètre n'est pas spécifié.

Exemple: http://siflore.fcbn.fr/?cd_ref=79319&to=2 Veut dire: Abies Alba Mill., 1768 en Métropole - Uniquement le nom choisi - Synthèse maille 10km

Templates (Affichage)

Les “templates”:http://symfony.com/fr/doc/current/book/templating.html contiennent le squelette du HTML qui sera utilisé pour générer le rendu envoyé au navigateur. Ceux utilisés par SIFlore sont contenus dans source:src/Fcbn/NewSIFloreBundle/Resources/views/Default et sont organisés comme suit:

- Page principale (template principal)

source:src/Fcbn/NewSIFloreBundle/Resources/views/Default/index.html.twig (Flore) et source:src/Fcbn/NewSIVegBundle/Resources/views/Default/index.html.twig (Végétation)

Elle fait appel aux 4 templates suivants:

- **Panneau de gauche** contenant le formulaire principal et les photos source:src/Fcbn/NewSIFloreBundle/Resources/views/Default/leftpanel.html.twig (Flore) et source:src/Fcbn/NewSIVegBundle/views/Default/leftpanel.html.twig (Végétation)
- **Panneau de droite** contenant les layers et la légende source:src/Fcbn/NewSIFloreBundle/Resources/views/Default/rightpanel.html.twig (Flore) et source:src/Fcbn/NewSIVegBundle/views/Default/rightpanel.html.twig (Végétation)
- **Onglet pour l'insertion de nouveaux commentaires** source:src/Fcbn/NewSIFloreBundle/Resources/views/Default/newcommentform.html.twig (Flore) et source:src/Fcbn/NewSIVegBundle/views/Default/newcommentform.html.twig (Végétation)
- **Onglet Notices** source:src/Fcbn/NewSIFloreBundle/Resources/views/Default/utilisation.html.twig (Flore) et source:src/Fcbn/NewSIVegBundle/views/Default/utilisation.html.twig (Végétation)

Les templates contiennent aussi les librairies javascript qui sont chargées à partir de CDN dans le template principal de SIFlore et celui de SIVeg

Tableaux (Synthèse, Observations, Commentaires)

Les tableaux de synthèse, observations et commentaires sont entièrement générés en Javascript

- Synthèses et Observations par source:/src/Fcbn/NewSIFloreBundle/assets/js/sifloregrids.js
- Commentaires par source:/src/Fcbn/NewSIFloreBundle/assets/js/siflorecom.js

Synthèses & Observations

Les 3 tableaux qui concernent ces données sont configurables dans des fichiers de description yml:

- Synthèse Mailles dans source:src/Fcbn/NewSIFloreBundle/Resources/config/mailles.yml
- Synthèse Taxons dans source:src/Fcbn/NewSIFloreBundle/Resources/config/taxons.yml
- Observations dans source:src/Fcbn/NewSIFloreBundle/Resources/config/observations.yml

Options

Ces fichiers décrivent le comportement de tous les champs. Les options et leurs comportements respectifs sont les suivants:

option	**comportement**	**valeur par défaut**
maille_display	Affiche le champ quand on est en mode maille (5 ou 10km)	false
commune_display	Affiche le champ en mode commune	false
label	Libellé à afficher en haut de la colonne	index du champ en remplaçant les @_@ par des espaces
field_name	Partie du SELECT qui sera utilisée pour récupérer le champ en SQL	index du champ
header_classes	Classes CSS à appliquer au @th@ de la colonne	
body_classes	Classes CSS à appliquer aux @td@ de la colonne	
roles	Rôles requis pour voir ce champ	
not_roles	Rôles exclus pour voir ce champ	
metro_display	Affiché pour la France (non pris en compte dans les observations)	false
reunion_display	Affiché pour La Réunion (non pris en compte dans les observations)	false

Examples

```
        id_flore_fcbn:
            maille_display: true
            commune_display: true
            header_classes: hidden
            body_classes: key hidden
```

Le champ sera toujours récupéré mais caché par CSS. Pour l'afficher, il suffirait de définir:

```
        id_flore_fcbn:
            maille_display: true
            commune_display: true
            body_classes: key
```

Le titre de la colonne serait “id flore fcbn” car aucun @label@ n'est défini (l'index du champ *id_flore_fcbn* est utilisé et les @_@ sont remplacés par des espaces).

```
        date_premiere_obs:
            label: Date prèmiere obs.
            maille_display: true
            commune_display: true
            metro_display: true
            roles: [IS_AUTHENTICATED_FULLY]
        date_premiere_obs_year:
            field_name: to_char(date_premiere_obs,'YYYY')
            label: Date prèmiere obs.
            maille_display: true
            commune_display: true
            metro_display: true
            not_roles: [IS_AUTHENTICATED_FULLY]
```

Ici on définit 2 fois la même donnée mais pour différencier 2 comportements. Ainsi les utilisateurs non connectés verront le champ @date_premiere_obs_year@ (juste l'année) et les autres verront @date_premiere_obs@ (date complète)

Requêtes (en base de donnée)

Les requêtes vers les données SIFlore sont stockées dans le fichier source:/src/Fcbn/NewSIFloreBundle/Services/QueryService.php

Elles utilisent la connexion *siflore* (paramétrée dans config.yml) avec des “requêtes préparées”:http://www.php.net/manual/fr/pdo.prepared-statements.php.

Le fichier est un “service”:http://symfony.com/fr/doc/current/book/service_container.html Symfony contenant une fonction par requête.

Chaque fonction prend les paramètres voulus et renvoie du HTML ou du JSON selon le besoin.

L’accès à ces requêtes est sécurisé dans source:/src/Fcbn/NewSIFloreBundle/Controller/QueryController.php Par exemple c'est ici qu'on interdit l’accès aux commentaires. Attention, ça ne conditionne pas l'affichage (onglets, boutons, …) l'affichage est géré dans les templates. La partie **SELECT** de chaque requête qui concerne une grid (Synthèses & Observations) est construite à partir de la configuration correspondante

Layers

Les layers sont stockés dans la base de donnée du portail. La table s’appelle @maplayers@. On peut y configurer notamment:

Champ	Description
treeNodeChild	Nom de la section (Fonds de cartes, INPN, ...)
label	Le libellé affiché
zone	La zone géographique pour laquelle le layer doit apparaître

Le modèle de la table est défini dans le fichier source:/src/Fcbn/NewSIFloreBundle/Entity/MapLayers.php

L'appel et le paramétrage des fonds cartographiques est fait dans le fichier source:/src/Fcbn/NewSIFloreBundle/Resources/public/js/sifloremaps.js

Fixtures (Layers d'origine)

Le projet utilise des “fixtures”: pour charger les layers d'origine La commande pour les installer est la suivante:

```
$ php app/console doctrine:fixtures:load --append
```

Ensuite quand on modifie un fichier contenant des fixtures (en cours de dev par exemple si on modifie .\Fcbn\NewSIFloreBundle\DataFixtures\ORM\LoadMapLayersData.php)

```
$ php app/console doctrine:fixtures:load
```

Système

Mise en place d'une machine virtuelle

Si vous n'avez pas de machine qui fonctionne sous linux, il vous faudra tout d'abord installer un machine virtuelle pour installer l'application en local. Pour ce faire, nous avons installé “VirtualBox”:https://www.virtualbox.org/, puis créé une machine virtuelle Ubuntu (“marche à suivre”:http://doc.ubuntu-fr.org/virtualbox). La création de la machine virtuelle se fait en 3 étapes :

1. Définir dans VirtualBox les paramètres de cette machine virtuelle (mémoire allouée, espace disque...),
2. pointer vers l'image Ubuntu que l'on souhaite installer,
3. Lancement de la machine virtuelle - Suivre les étape d'installation d'Ubuntu,
4. Redémarrer. Attention, l'image Ubuntu ayant permis l'installation n'est pas toujours “éjecter” automatique. Il peut être nécessaire de supprimer le lien vers cet image dans les paramètres de VirtualBox (cf. étape 2)

Il est possible que la machine virtuelle ne propose pas une fenêtre adaptée à votre écran notamment. Pour accéder au mode plein écran, il suffit d'installer les “additions invités”:http://www.commentcamarche.net/faq/7576-virtualbox-installer-les-additions-client-dans-ubuntu

Authentification & Autorisation

Login/Logout

Les actions de connexion/déconnexion redirigent l'utilisateur vers l'adresse précédente. Si par exemple on se déconnecte à partir de SIFlore, la même page se rafraîchira avec les permissions du grand public. Si on crée un utilisateur CBN et qu'on se reconnecte tout de suite avec, on aura une erreur d'accès non autorisé.

Rôles

Les différences de comportement dans l'interface sont gérées par des permissions (appelées "roles":http://symfony.com/fr/doc/current/book/security.html#les-roles dans Symfony) Par exemple pour accéder à la gestion des utilisateurs, le rôle **SONATA_USER_ADMIN_USER** est requis.

Les utilisateurs connectés se voient automatiquement attribués le rôle **IS_AUTHENTICATED_FULLY** (comportement Symfony par défaut).
C'est ce rôle qu'on utilise pour savoir si on doit afficher certains éléments de SI Flore.
Exemple pour savoir si on doit afficher **Bryophytes DHFF / conv. Berne** et **Bryophytes hors DHFF / conv. Berne** ou seulement **Bryophytes**, le contrôle est fait ici source:/src/Fcbn/NewSIFloreBundle/Resources/views/Default/leftpanel.html.twig#L18

Groupes

Pour regrouper ces permissions dans des profils (CBN, Admin, ...) on utilise des groupes (Concept apporté par "FOSUserBundle":https://github.com/FriendsOfSymfony/FOSUserBundle)

Affectation multiple

Pour affecter un groupe à une selection d'utilisateur il faut pouvoir lancer les commandes SQL suivantes. Par exemple pour affecter le groupe 6 à tous les utilisateurs n'ayant pas de groupe:

```
fcbn=> CREATE TEMPORARY TABLE tmp_group AS
SELECT usr.id AS user_id, 6 AS group_id
FROM fos_user_user AS usr
LEFT JOIN fos_user_user_group AS ugp
  ON (usr.id = ugp.user_id AND ugp.group_id = 6)
WHERE ugp.user_id IS NULL;
fcbn=> INSERT INTO fos_user_user_group SELECT * FROM tmp_group;
```

Bases de données

Elles sont réglées dans le fichier source:/app/config/config.yml

- **default**

La base de donnée utilisée pour le portail (utilisateurs, groupes, layers, ...) Elle est réglée dans **/app/config/parameters.yml**

- **siflore**

Pointe vers les données PostgreSQL de SIFlore

- **photos**

Base MySQL contenant les photos

Réplication des données

Pour copier les données d'un serveur à un autre, par exemple de la pre-prod vers la prod:

À partir de la pre-prod copier les données dans un fichier

```
$ pg_dump -U fcbn -h localhost -C fcbn --data-only > machin.sql
```

Copier le fichier par ssh

```
$ scp -P 10022 machin.sql siflore.fcbn.fr:
```

À partir de la prod et du repertoire du projet, supprimer la base

```
~/fcbn$ php app/console doctrine:database:drop --force
```

Recréer la base et les tables

```
~/fcbn$ php app/console doctrine:database:create
~/fcbn$ php app/console doctrine:schema:create
```

Dumper les données dans la base

```
$ psql -h localhost -U siflore -f machin.sql
```

Configuration des mails

- Pour l'envoi des mails de réinitialisation de mot de passe, Symfony a besoin d'un destinataire.

Celui-ci est réglé dans source:app/config/config.yml dans la section @fos_user.from_email@

- Par défaut, Symfony utilise le SMTP local, donc il doit y en avoir un d'installé sur la machine (“exemple Postfix”:https://www.digitalocean.com/community/articles/how-to-install-and-setup-postfix-on-ubuntu-12-04)
- Quand un utilisateur demande un nouveau mot de passe, la date est stokée dans la table des utilisateurs ce qui l’empêche de faire une 2eme demande
- Pour mettre à 0 les dates de demande

```
UPDATE fos_user_user SET password_requested_at = NULL;
```

Installation de l'application en local

Requiert d'installer

1. “git”:http://fr.wikipedia.org/wiki/Git
2. “postgresql”:http://www.postgresqlfr.org/

(NB: pour un déploiement en local sous linux on peut monter une machine virtuelle linux avec “virtual box”:https://www.virtualbox.org)

```
$ sudo apt-get install git postgresql postgis
```

Il est aussi nécessaire d'avoir “apache2” (on installe pour cela le package lamp-server)

```
$ sudo apt-get install lamp-server^
```

Les étapes pour déployer le projet sont les suivantes (à faire à partir d'un terminal):

Récupérer le projet

Au préalable la clé ssh doit avoir été envoyée à l'admin.

Générer une clé SSH

Si aucune clé n'a encore été créée

```
$ ssh-keygen
```

Laisser toutes les valeurs par defaut (Enter à chaque fois). Afficher le contenu de la clé publique:

```
$ cd
$ cat .ssh/id_rsa.pub
```

Copier/coller le resultat dans un mail pour l'admin

Cloner

La récupération du projet se fait en clonant le repository avec git

```
$ git clone fcbn@dev.masao.eu:fcbn.git
```

Si le message suivant aparait

> Agent admitted failure to sign using the key.

Vous devrez ajouter votre clé à l'agent SSH

```
$ ssh-add
```

Racine du projet

À partir d'ici, lancer toutes les commandes à la racine du projet:

```
$ cd fcbn
```

Créer la base de donnée et l'utilisateur

Exemple avec PostgreSQL

```
$ sudo -u postgres createuser -d -R -P fcbn
$ sudo -u postgres createdb -O fcbn fcbn
```

L'utilisateur ne devrait pas avoir besoin d'être "superuser". La 2eme ligne crée la base et l'affecte à l'utilisateur fcbn

Installer les extensions php

```
$ sudo apt-get install php5-cli php5-intl php5-gd php5-json php5-pgsql
```

- **php-cli** est un interpréteur php en ligne de commande.
- **php5-intl** Module php d'internationalisation dont dépend le projet. Il permet "d'internationaliser" l'application, qui permettra, plus tard, par exemple de traduire les libellés et de formater les dates en fonction du pays.
- **php5-gd** librairie qui permet de créer/modifier des images (requise depuis l'ajout des crédits sur les photos).
- **php5-json** Convertit les données pour Javascript
- **php5-pgsql** module pour les connexions à la base de données postgresql directement depuis les scripts php. Cela inclu aussi le module pdo_pgsql pour un usage avec l'extension PHP data object

Récupérer composer

À la racine du projet

```
$ curl -s https://getcomposer.org/installer | php
```

Si @curl@ n'est pas présent, l'installer:

```
$ sudo apt-get install curl
```

Installer composer dans le bin, (exemple sous Debian)

> « Composer » est un fichier PHAR exécutable:

```
$ sudo mv composer.phar /usr/local/bin/composer
```

```
$ alias composer='/usr/local/bin/composer/composer.phar'
```

Ici on a déplacé l'executable composer.phar vers /usr/local/bin/composer ce qui permet de le lancer de n'importe où. Ensuite on souhaite déployer les dépendances du projet. On doit donc se positionner à la racine du projet dans laquelle se trouve le fichier composer.json qui contient les dépendances du projet

Installer les dépendances (dont Symfony)

```
$ composer install
```

À la fin de l'installation spécifier les réglages de base de données utilisés précédemment (database_driver: pdo_pgsql, database_host:localhost, db: fcbn, user: fcbn, pass: fcbn, laisser le reste par défaut)

Configurer le vhost

Utiliser l'éditeur de texte nano et recopier dedans le vhost indiqué plus bas

```
$ sudo nano /etc/apache2/sites-available/fcbn.conf
```

Si @nano@ n'est pas present, l'installer:

```
$ sudo apt-get install nano
```

Coller le vhost ci-dessous dans nano

```
<VirtualHost *:80>
```

```
    ServerAdmin cestmoil@admin.net

    DocumentRoot "/var/www/fcbn/"
    <Directory /var/www/fcbn/>
        Options Indexes FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from All
        <IfModule mod_rewrite.c>
            RewriteEngine On
            RewriteCond %{REQUEST_FILENAME} !-f
            RewriteRule ^(.*)$ /app_dev.php [QSA,L]
        </IfModule>
    </Directory>
</VirtualHost>
```

Enregistrer le fichier (qui s'appelle fcbn) en faisant Ctrl+O puis Ctrl+X pour quitter

Les droits du fichier doivent être les mêmes que ceux déjà présents, sinon:

```
$ sudo chmod 644 /etc/apache2/sites-available/fcbn.conf
$ sudo chown root:root /etc/apache2/sites-available/fcbn.conf
```

Nommer le serveur

Requis pour une première installation

Le nom du serveur “localhost” n'est pas réglé explicitement par défaut Le spécifier de cette manière:

```
$ echo "ServerName localhost" | sudo tee -a /etc/apache2/apache2.conf
```

Activer le module @rewrite@

@rewrite@ permet de modifier les chemins de requête HTTP en temps réel

```
$ sudo a2enmod rewrite
```

pour activer la nouvelle configuration

```
$ service apache2 restart
```

Lier le vhost

Faire pointer @/var/www/fcbn/@ sur le repertoire contenant @app.php@ et @app_dev.php@

```
$ sudo ln -s ~moi/fcbn/web/ /var/www/fcbn
```

Remplacer @~moi/fcbn/web/@ par le chemin d'installation de l'application (en y incluant le répertoire @web@)

Activer le vhost

```
 $ sudo a2ensite fcbn.conf
```

Redémarrer le serveur

```
$ sudo service apache2 restart
```

Configurer le cache et log

“voir la doc officielle en anglais”:http://symfony.com/doc/master/book/installation.html#configuration-and-setup
“voir la doc officielle en français”:http://symfony.com/fr/doc/current/book/installation.html
Préférer la méthode “setfacl”

Si la gestion des ACL n'est pas présente, l'installer:

```
 $ sudo apt-get install acl
```

Déployer la base

Toujours à la racine du projet

```
$ php app/console doctrine:schema:create
```

Créer le superuser

```
$ php app/console fos:user:create --super-admin admin cestmoil@admin.net
pass
```

Vérifier l'installation

```
$ php app/check.php
```

Et enfin charger les fixtures

Désactiver le vhost default si présent (dans /etc/apache2/sites-available):

```
$ sudo a2dissite 000-default.conf
```

ou

```
$ sudo a2dissite default
```

Installer les assets

Les ressources publiques (JS, CSS, images) doivent se trouver dans le répertoire des assets (par defaut /web/bundles)

```
$ php app/console assets:install --symlink
```

Modifications de l'application: les étapes

Logiciels requis

Fabric : est une librairie python et un outil en ligne de commande pour faciliter l'utilisation d'application de déploiement en SSH ou des tâches d'aministration système. C'est donc un outil qui permet d'écrire, en Python, un script de déploiement, c'est à dire d'automatiser la mise en ligne d'une nouvelle version de site.

```
$ sudo apt-get install fabric
```

Le fichier source:fabfile.py contient donc quelques enchainements de commandes d'administration.

Voici un petit blog qui explique bien ce qu'on peut faire avec Fabric: ici

Architecture de mise en production

L'application en locale est poussée depuis les machines de développement(git push) vers un repository (site de dépôt, actuellement hébergé chez MASAO). Elle peut aussi être tirée (git pull) du repository vers les machines de dev.

Ensuite elle peut passer du repository à la pre-prod (hébergée actuellement chez MASAO) ou du repository à la prod (hébergée actuellement sur le serveur de la FCBN)

Opération de mise en production

Tout d'abord il faut paramétrer le .git/config
C'est dans ce fichier qu'est indiqué le chemin du repository. On paramètre ce fichier en:

- changeant l'url avec le nom d'utilisateur autorisé à se connecter au repository (deamnde faite à l'administrateur en lui fournisant la clé ssh de la machine après l'avoir générée avec ssh-keygen

```
$sudo nano .git/config
```

- puis indiquer

```
url = fcbn@dev.masao.eu:fcbn.git
```

- Puis configurer vos informations pour git

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

La modification est réalisée en local. Une fois la modification réalisée on peut vérifier ce qui a été fait.
La commande

```
$ git status
```

Indique les fichiers modifiés.

```
$ git diff
```

Donne le détail des différences entre fichiers

```
$ git reset --hard
```

Permet d'annuler la dernière mise à jour (revenir à l'état du dernier commit).

```
$ git pull
```

Permet de récupérer en local la dernière version de l'application
Enfin la commande

```
$cat .gitignore
```

Permet de savoir quels fichiers ne sont pas gérés par git.

Vider les caches

–puis on vide les caches des assets

```
sudo php app/console assetic:dump --env=preprod
```

–puis on vide le cache php

```
sudo php app/console cache:clear --env=preprod
```

```
sudo chown -R www-data /home/preprod/siflore
```

Utilisation de Fabric:

On liste les fonctions de fabric de la façon suivante:

```
$ fab -l
```

Il est possible de pousser sur le repository uniquement la dernière modification en enchainant les commandes

```
$ fab commit
```

Le commit se contente de versionner les dernières modifs: après le lancement du commit, il est demandé de commenter la modification. Dans la mesure où git est couplée à redmine, cela permet d'historiser les modifications dans le dépôt de redmine. et

```
$ git push
```

Pour pousser la dernière version sur le repository

Pour déployer l'application directement en production (utiliser avec parcimonie), enchainer les commandes suivantes:

```
$ fab commit
```

et

```
$ git push
```

et

```
$ fab prod deploy
```