Rapport du site projet: CMS-Yungi

# Analyse

## Concept de base

Je veux réaliser un site web de type CMS. Ce site sera surtout orienté catalogue de vente, type e-commerce. Cependant tous les concepts du e-commerce ne seront pas traités dans cette version.

Un des concepts repris de Drupal est le concept des modèles de pages. Je m’explique vous pouvez avoir différents *types* de pages, ce qui signifie qu’il faut pouvoir définir des champs de saisies depuis une interface d’administration.

Exemple d’un modèle de page :

* + - La page de type catalogue se définit de la sorte :
      1. Un champ texte pour le nom du produit [obligatoire]
      2. Un champ texte pour la référence du produit [facultatif]
      3. Un champ bracelet
      4. Un champ mouvement
      5. Une zone de texte
      6. Des pièces jointes :
         1. Des Images [galerie]
         2. Des vidéos
         3. Un document PDF => Fiche produit à la volée
      7. Un prix + gestion des monnaies étrangères
      8. […]

Ce site tournera aussi avec un moteur de templates, ce qui me forcera aussi d’une certaine manière à séparer un maximum les processus métiers de l’affichage.

Pour une question de lisibilité ainsi que de maintenabilité je vais regrouper toutes les intelligences d’affichage et métier dans des classes et par conséquent utiliser la POO de PHP.

Le contenu doit être stocké dans une base de données, cependant la partie admin peut être stockée dans des fichiers.

## Drupal

### Modèles de documents

Un concept très intéressant de Drupal, il permet de construire ces formulaires de saisie pour des types de pages que l’utilisateur défini. Ce qui offre une souplesse maximale à l’utilisateur.

### Affichage via des "templates"

Le contenu est indépendant du contenant. Je m’explique on met en page des containers, puis on y injecte du contenu. Voici le procédé du templating. Ce qui implique que le contenu doit être structuré. Pour ce faire il existe plusieurs technologie mais le concept est important ont doit récupérer un tableau de données.

### Tout tient dans des catégories

Dans Drupal un concept important est le concept de liens et d’organisation du contenu. On dit qu’il est fondé sur un principe de « taxonomie ». Ce concept est relativement compliqué mais il repose sur un système de classification avec des pondérations. Ce qui signifie concrètement qu’un contenu n’appartient pas à une page mais à une catégorie qui elle s’affiche indépendamment de la page. Ensuite il existe aussi une hiérarchie entre les différentes catégories.

### Console d'administration

Il faut que je m’inspire pour ce projet de la console d’administration de Drupal. Qui pour moi est vraiment trop complexe pour mon projet mais me paraît cohérente dans son interface. De plus il faut que je me penche sur leur système de modèles.

### Expérience utilisateurs

Nous allons essayer d’utiliser un maximum de technologies actuelles afin d’améliorer l’expérience de l’utilisateur avec notre plate-forme. Ce qui sous-entend que le JavaScript sera de la partie. Il faut aussi donc que je garde cette idée à l’esprit dans ma conception.

## Vivamea

### Logique d'affichage

La logique d’affichage de Vivamea repose entièrement sur le moteur de template Smarty. Etant donné que je l’ai déjà utilisé dans la maintenance de Vivamea, j’ai opté pour ce moteur. De plus après quelles que recherches sur Internet il est vraiment bien positionné. Il permet de gérer le cach, et surtout de séparer la logique métier de l’affichage.

### Procédé Commercial

Chez Vivaema, j’ai pu aussi voir comment il fallait découper son affichage afin d’offrir des zones d’affichage supplémentaires pour des publicités par exemple.

Le principe de White Labels, c’est-à-dire de pouvoir modifier le layout d’un site sans en changer l’intelligence, permet de faire des partenariats commerciaux.

## Partis pris

### Librairie JS: JQuery

Pour la librairie JavaScript je vais opter pour JQuery, elle est très bien documentée, me paraît parfaitement adaptée et une grosse communauté utilise cette librairie. Ce qui signifie que les informations se trouveront plus facilement.

### Moteur de template PHP: Smarty

Est un moteur complet, et je l’ai déjà utilisé, pas dans toute sa puissance mais son procédé ne m’est pas inconnu.

### Contenus stockés dans du MySql: JSON

Pour la structure du contenu, étant donné que j’ai besoin d’un contenu structuré, j’ai choisi le format JSON (JavaScript Object Notation). Ce format a été choisi parce qu’il offre de nombreux avantages.

#### JSON vs XML

Il est nettement plus économique en syntaxe que du XML. Et par conséquent plus léger que du XML.

#### Les prises en charges natives

Javascript prend en charge nativement les objets JSON et PHP les transforment en tableaux PHP sans aucuns soucis.

#### JSON est tendance

C’est un format qui a le vent en poupe. On parle maintenant de base NoSQL, ces bases qui seraient l’avenir (Google et Facebook entre autre ne travaillent pratiquement qu’avec ces systèmes) intègrent aussi nativement le JSON.

#### AJAX

JSON est un format couramment utilisé avec les requêtes AJAX. On peut générer du XML cependant ce format n’est pas pris nativement en charge par PHP, ce qui n’est pas le cas de JSON… En effet on peut transformer un tableau PHP en JSON par une simple fonction.

### Accès aux données: PDO

Nous allons utiliser PDO afin d’accéder aux données. Ce qui me permet de faire un code d’accès aux données indépendant du SGBD. Elle joue le rôle d’interface, permettant au programmeur de coder sans trop se soucier du déploiement en divers environnements ni de se soucier de la suite.

PDO gère aussi les exceptions, ce qui permet d’avoir un script plus stable. De plus à partir de PHP 6, se sera le système par défaut activer pour accéder aux données, donc la librairie a de l’avenir devant elle.

### Versionning: Totoise SVN & Code.Google.com

Durant le projet toutes les évolutions seront commitées. Ce qui permet d’avoir des versions et de revenir sur des changements le cas échéant.

Comme Repository en ligne j’ai choisi Google Code, qui m’offre un serveur SVN en line gratuitement… enfin à voir.

# Conception

## Intelligence d'affichage

### Ce que nous voulons

* + - * 1. Un site piloté essentiellement par les CSS et du JavaScript.
        2. Séparé un maximum l’intelligence métier de l’affichage. Ce qui permet de faire soit des « white labels », soit des déclinaisons le plus facilement possibles.
        3. Pour ce faire j’ai défini les types principaux d’affichages possibles :

Une page, se rapportant à un seul contenu

Un formulaire, composé de champs (texte, textarea, password, select, checkbox, radios, bouton et autres reset ou submit.

Une matrice permettant l’affichage de plusieurs éléments.

Un type menu, permettant de générer des listes.

* + - * 1. Une composition qui nous permet facilement de jongler avec les différentes demandes d’affichages.

### Squelette de base du site

#### Type Page

#### Type Formulaire

##### Type input

##### Type checkbox

##### Type radio

##### Type select

##### Type textarea

#### Type Matrice

#### Type Menu

#### Theming – gestion des thèmes

#### Classique

#### Tradiluxe à venir

#### Arborescence

## Conception de la BDD

### MCD

### Raisonnements intermédiaires

### MLD

## Quels que diagrammes de traitements

### Création d'un modèle de page

### Ajout d'un produit dans le catalogue

### Gestion d'un membre

## Intelligence Métiers

### Diagramme des classes

### Classes de gestion du site

#### Artworks

#### Categories

#### DataBase

#### Debug

#### Documents

#### Files

#### Form

#### Images

#### InfosWebsite

#### Languages

#### Matrice

#### Menu

#### Pages

#### Scripts

#### SmartyYungi

#### String

#### Upload

#### User

#### Videos

#### Vues

# Réalistation

## Google Analytics

## Google Maps

## Présentation "Maquette"

# Tests