

ModuloDocs

Création d'une base documentaire interne



Table des matières

1. Présentation	3
1.1. Objet du document	4
1.2. L'entreprise	4
1.3. The project	4
1.4. Le projet	4
1.4. Les acteurs du projet	5
2. Définition des besoins	6
2.1. Analyse de l'existant	6
2.2. Enjeux et objectifs	7
2.3. Cas d'utilisation	7
2.4. Le public ciblé	9
2.5. Les contraintes de réalisation	9
3. Réponse à la demande	10
3.1. Spécifications et fonctionnalités	10
3.2. Styletile	10
3.3. Wireframes	10
3.4. Maquettes	13
4. Réalisation	16
4.1. Environnements	16
4.2. Technologies utilisées	17
4.3. Arborescences	18
4.4. Base de données	21
5. Architecture	24
5.1. Le routeur	25
5.2. Le DAO	26
5.3. Le MVC	29
6. Sécurité	52
6.1. Tests	52
6.2. Sécurité	54
7. Gestion du projet	58
7.1. Livrables	58
7.2. Planning	58
7.3. Déploiement	62
8. Conclusion	63
8.1. Bilan	63
8.2. Améliorations	64
8.3. Comparatif maquette / site final	64
9. Annexes	66
9.1. Table des illustrations	66

0. Compétences

Maquetter une application :

3.2. Styletile	10
3.3. Wireframes	10
3.4. Maquettes	13

Développer une interface utilisateur :

5.3. Le MVC	29
-----------------------------	----

Développer des composants d'accès aux données :

5.2. Le DAO	26
-----------------------------	----

Développer des pages web en lien avec une base de données :

4.4. Base de données	21
--------------------------------------	----

Concevoir une base de données :

4.4. Base de données	21
--------------------------------------	----

Mettre en place une base de données :

4.4. Base de données	21
--------------------------------------	----

Développer des composants dans le langage d'une base de données :

5.2. Le DAO	26
-----------------------------	----

Utiliser l'anglais dans son activité professionnelle en informatique :

1.3. The project	4
----------------------------------	---

Concevoir une application :

2. Définition des besoins	6
3. Réponse à la demande	10

Collaborer à la gestion d'un projet informatique :

2. Définition des besoins	6
7. Gestion du projet	58

Développer des composants métier :

5. Architecture	24
---------------------------------	----

Construire une application organisée en couches :

5. Architecture	24
---------------------------------	----

Préparer et exécuter les plans de tests d'une application :

6.1. Tests	52
----------------------------	----

1. Présentation

1.1. Objet du document

Ce document a pour objectif de présenter une vue détaillée de la mise en place du projet ModuloDocs. On y retrouvera toutes les étapes de réalisation, de l'expression des besoins au déploiement du site.

1.2. L'entreprise

Modulo+ est une agence web créée en 2005, spécialisée dans le développement de sites et applications web pour les PME (extranet / intranet). Leur démarche de travail est de proposer des solutions informatiques sur mesure, adaptées aux besoins de l'entreprise et de ses utilisateurs.

Elle emploie actuellement trois personnes, Samuel Gallard (chef de projet et développeur back-end), Régis Oliviero (chef de projet et développeur front-end) et Mireille Campourcy (développeuse web).

1.3. The project

Modulo+ is a web agency created in 2005, specialized in websites and web applications development for small and medium-sized businesses (extranet and intranet).

Currently, the necessary documentary resources to set up projects aren't centralized. They are on each employee's computer, but there is no follow-up of the resources use, nor any sharing or documentation on an intern network.

To optimize the projects set-up and the technical researches, Modulo+ wants to have an intern application to improve its inner working and to prepare its future development plan.

This application will be called ModuloDocs. It will be developed in PHP and may content technical documentation, pictures, snippets and commented links.

It will be useful to transmit documentation to new arrivals, and to follow the resources use (e.g. pictures bought from data banks). The developers will also have a follow-up of the changes they made to their system (e.g. adding modules). It will also be useful to test new technologies, and eventually implement them on the actual system.

1.4. Le projet

Actuellement, les ressources documentaires nécessaires lors de la mise en place des projets de Modulo+ ne sont pas centralisées. Elles sont sur les postes de chaque employé, mais il y a peu de suivi d'utilisation d'une ressource ni de mise en commun ou de documentation des outils sur un réseau interne.

Pour optimiser la mise en place des projets et la recherche d'informations techniques, Modulo+ souhaite mettre en place une application interne pour améliorer son fonctionnement interne et préparer son futur développement.

Cette application sera appelée ModuloDocs. Elle sera développée en PHP, et pourra contenir de la documentation technique, des images, des morceaux de code source, et des liens commentés.

Elle permettra de transmettre de la documentation à de nouveaux arrivants, et de mieux suivre l'utilisation des ressources, notamment lors de l'achat d'images et de templates. Elle pourra également servir à tester des technologies récentes, pour éventuellement les implémenter dans le système actuel.

1.5. Les acteurs du projet

La MOA (Maîtrise d'ouvrage)

L'entreprise Modulo+, sous la responsabilité de Samuel Gallard, chef de projet et développeur back-office et de Régis Oliviero, chef de projet et développeur front-office

La MOE (Maîtrise d'œuvre)

Mireille Campourcy, développeuse web

2. Définition des besoins

2.1. Analyse de l'existant

Actuellement, le site de Modulo+ est composé d'un intranet¹, d'un front² et d'un back-office³. Le passage de l'intranet au back-office se fait via un menu déroulant. Tous les sites créés par l'entreprise sont sur ce modèle. ModuloDocs viendra s'y greffer, et apparaîtra dans le menu déroulant de l'intranet et du back-office.

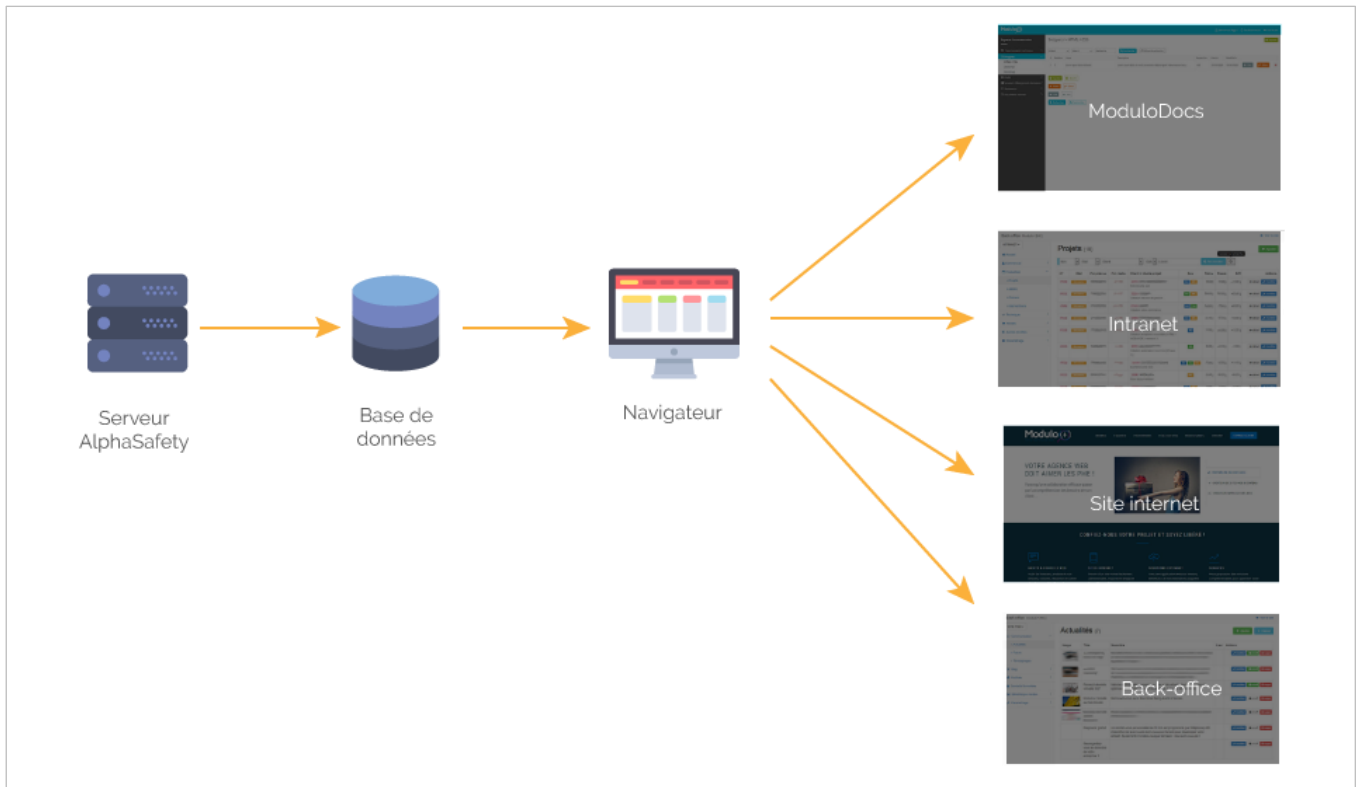


Figure 1 – Structure du site www.Modulo+.com

Tous les intranets créés par l'entreprise, y compris celui à usage interne, ont la même charte graphique.

L'entreprise travaille avec le framework⁴ CSS Bootstrap 3, aussi bien pour les intranets que les sites web, ce qui permet d'avoir des sites en responsive design⁵.

¹ Intranet : réseau informatique utilisé à l'intérieur d'une entreprise

² Front-office : partie d'un système informatique accessible aux utilisateurs finaux ou aux clients

³ Back-office : la partie d'un système informatique qui n'est pas accessible aux utilisateurs finaux ou aux clients

⁴ Framework : ensemble de classes d'objet, utilisables pour créer des applications informatiques.

⁵ Responsive design : un design qui s'adapte à tous les appareils (ordinateurs, portables, tablettes...)

2.2. Enjeux et objectifs

Objectif n°1

Améliorer l'efficacité de l'équipe technique lors de la mise en place d'un projet en optimisant la mise à disposition des ressources documentaires.

Enjeux

- Avoir une meilleure visibilité des différentes ressources
- Gagner du temps lors de la phase de recherche de ressources

Bénéfices attendus

La mise en place d'un projet se fera plus rapidement, et la recherche de données concernant d'anciens projets sera plus efficace. Le risque d'avoir une même ressource dupliquée sera largement minimisé.

Objectif n°2

Assurer la transmission des informations liées à l'utilisation des outils.

Enjeux

- Avoir une meilleure visibilité des différentes ressources
- Avoir une documentation technique des outils développés, pour s'y référer pendant le développement d'une application

Bénéfices attendus

La formation de personnes tierces sur les applications internes ou externes utilisées par les développeurs sera facilitée.

2.3. Cas d'utilisation

ModuloDocs se composera principalement de documents, qui pourront être rangés en catégories et sous-catégories. Il contiendra une médiathèque dans laquelle il sera possible d'importer des images et des documents type .pdf et .doc. Ces médias pourront être associés à des documents.

La connexion à ModuloDocs se fera en amont, comme pour l'intranet. Un utilisateur n'aura qu'un seul rôle, celui d'administrateur.

L'utilisateur, une fois connecté, pourra consulter, créer, éditer ou supprimer une catégorie. Idem pour les documents : il pourra en consulter un, en créer, en éditer ou en supprimer. Enfin, il pourra créer, consulter ou supprimer un média.

Il pourra associer un document à une catégorie ou un média, ce qui inclut que le document, la catégorie et le média existent.

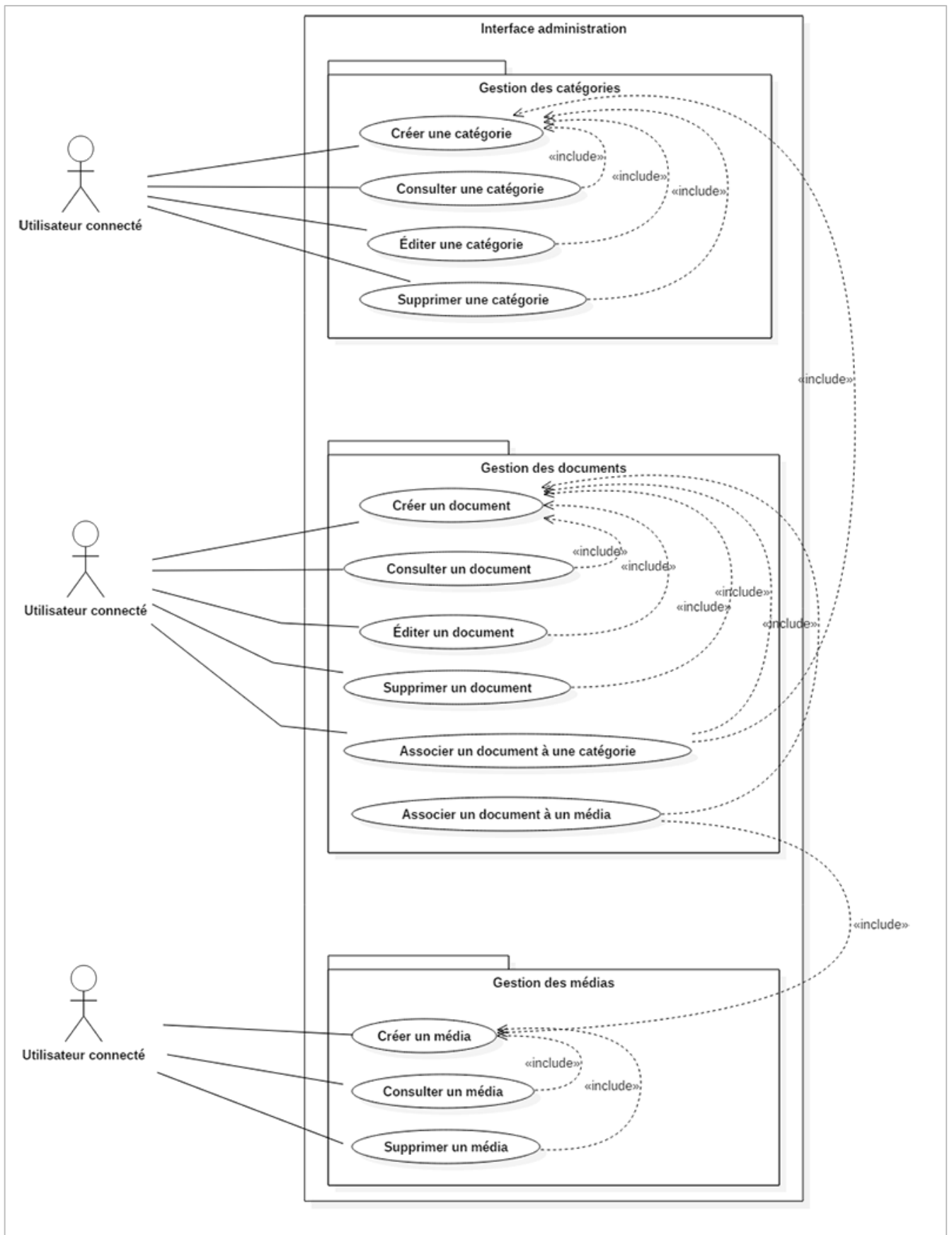


Figure 2 - Schéma des cas d'utilisation

2.4. Le public ciblé

Les utilisateurs de ModuloDocs seront les employés de Modulo+. Ce sont des développeurs, qui ont l'habitude d'utiliser l'intranet et qui entreront eux-mêmes les ressources dans ModuloDocs.

2.5. Les contraintes de réalisation

ModuloDocs sera développé avec le framework CSS Bootstrap 3 (CSS 3, HTML 5), en PHP.

Le site doit être compatible pour tous les navigateurs (> IE9).

Il sera basé sur la charte graphique de l'intranet actuel, qui peut être modernisée. Si la charte graphique convient, elle sera ensuite appliquée aux autres intranets existants. Les changements ne doivent donc pas dérouter les utilisateurs qui ont l'habitude d'utiliser l'intranet.

La développeuse chargée du projet étant amenée à travailler sur d'autres projets en parallèle, la mise en œuvre de celui-ci peut prendre du retard.

3. Réponse à la demande

3.1. Spécifications et fonctionnalités

ModuloDocs se composera principalement de documents, qui auront un titre, un sous-titre, deux zones de textes et un lien externe. Les documents pourront être rangés en catégories et sous-catégories. Ils auront une option « A la une » pour les mettre en avant sur la page d'accueil. Le lien externe devra apparaître sur la liste des documents.

Les catégories auront un nom et une icône, qui apparaîtra dans le menu. Les sous-catégories auront un nom et une catégorie parente, mais pas d'icône. Il sera possible de modifier leur ordre d'apparition dans le menu.

ModuloDocs contiendra une médiathèque dans laquelle il sera possible d'importer des images et des documents. Ces médias pourront être associés à des documents. Les formats importables dans la médiathèque seront jpg, png, doc et pdf. Ils devront faire moins de 100Mo.

3.2. Styletile


Les modifications se sont principalement faites sur les couleurs pour moderniser la charte graphique et faire ressortir les informations importantes. Le gris et le bleu rappellent les couleurs du logo, et les couleurs vert et orange rendent le tout moins sombre et plus énergique.



Figure 3 - StyleTile

3.3. Wireframes

ModuloDocs étant basé sur l'intranet existant, sa structure sera identique pour ne pas modifier les habitudes des utilisateurs : un menu sur la gauche, les documents affichés sous forme de liste sur la droite avec des boutons pour les différentes actions possibles.



Catégories
Sous-catégorie
Sous-catégorie
Catégories
Sous-catégorie
Sous-catégorie
Catégories
Sous-catégorie
Sous-catégorie

Bienvenue, Monsieur Durand [Voir le site >](#)

Accueil

[Créer un document](#)

Bienvenue

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis pretium condimentum lacinia. Maecenas dictum libero luctus, dapibus lorem id, commodo justo. Morbi sed nisi sodales, condimentum dui sed, porta leo. Sed nunc mi, ornare eget sagittis eu, faucibus vitae mauris. Ut consectetur ultricies elementum.


Documents à la une

id	Titre	Description	Créé le	Modifié le	
1	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail Éditer Supprimer
2	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail Éditer Supprimer
3	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail Éditer Supprimer

Derniers documents publiés

id	Titre	Description	Créé le	Modifié le	
1	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail Éditer Supprimer
2	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail Éditer Supprimer
3	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail Éditer Supprimer

Figure 4 - Affichage de la page d'accueil



Catégories
Sous-catégorie
Sous-catégorie
Catégories
Sous-catégorie
Sous-catégorie
Catégories
Sous-catégorie
Sous-catégorie

Bienvenue, Monsieur Durand
[Voir le site >](#)


[Accueil](#) > Catégorie

Créer un document

Catégorie

id	Titre	Description	Créé le	Modifié le			
1	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail	Éditer	Supprimer
2	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail	Éditer	Supprimer
3	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail	Éditer	Supprimer

Figure 5 - Affichage d'une catégorie



Catégories
Sous-catégorie
Sous-catégorie
Catégories
Sous-catégorie
Sous-catégorie
Catégories
Sous-catégorie
Sous-catégorie

Bienvenue, Monsieur Durand
[Voir le site >](#)

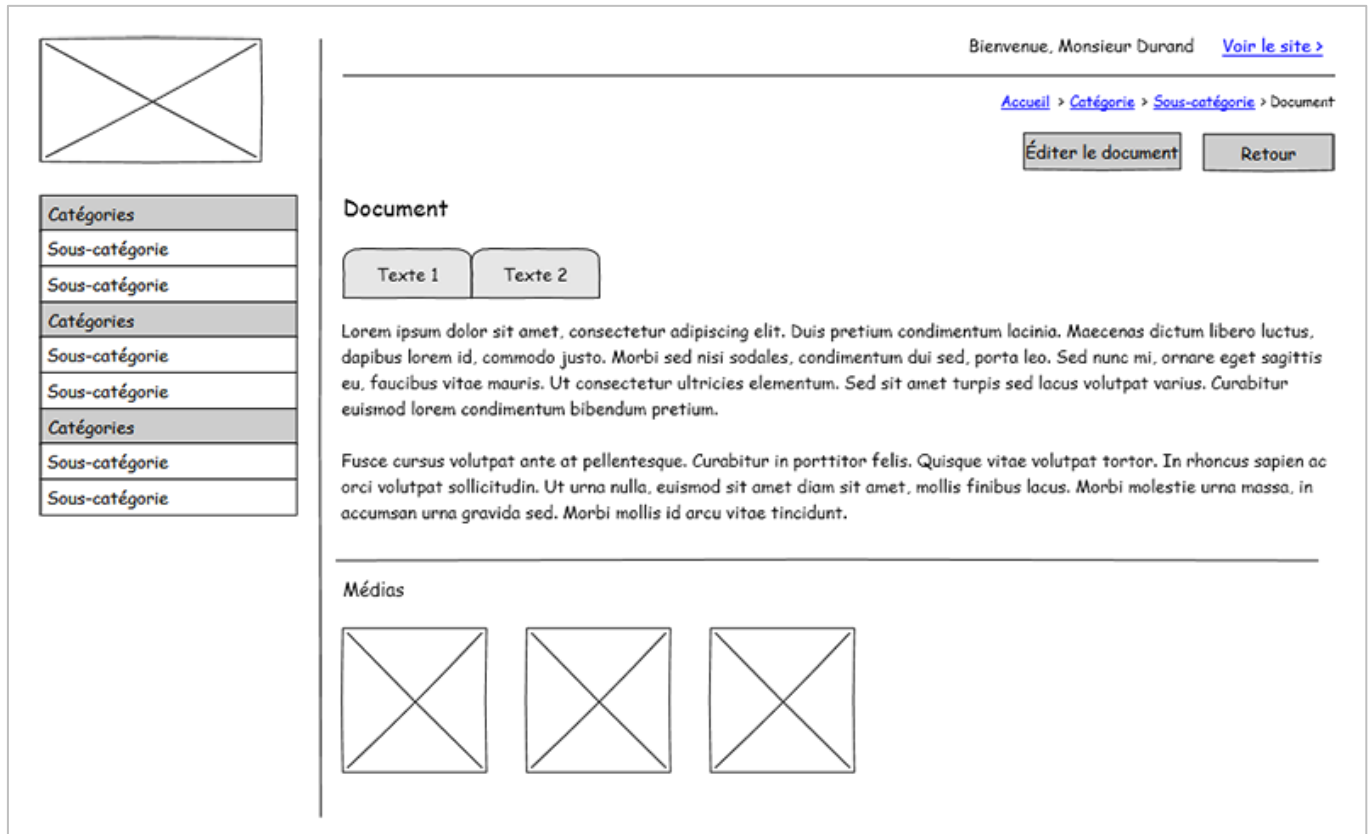
[Accueil](#) > [Catégorie](#) > Sous-catégorie

Créer un document

Sous-catégorie

id	Titre	Description	Créé le	Modifié le			
1	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail	Éditer	Supprimer
2	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail	Éditer	Supprimer
3	Lorem ipsum	Lorem ipsum	00/00/0000	00/00/0000	Détail	Éditer	Supprimer

Figure 6 - Affichage d'une sous-catégorie



3.4. Maquettes

La charte graphique actuelle est un thème de base Bootstrap.

Pour ModuloDocs, il a été décidé de changer les couleurs pour le moderniser et faire ressortir les informations importantes. Le menu et les contenus se confondent moins grâce au contraste des couleurs de fond.

Le logo de l'entreprise a été ajouté en haut à gauche : il pourra être personnalisé si le thème est repris sur les intranets d'autres clients.

Voici l'intranet existant :

Back-office Modulo+ [MC] Voir le site

INTRANET

Accueil

Commercial

Production

> Projets

> Jalons

> Consos

> Interventions

</> Technique

Achats

Autres recettes

Paramétrage

Projets (18)

Non Etat Client Collé Libellé Rechercher Annuler la recherche Ajouter

N°	Etat	Fin prévue	Fin réelle	Client + libellé projet	Dev	Prévu	Passé	Diff	Actions
	En cours			Refonte site web	RO MC				détail modifier
1	En cours			Création Intranet de gestion	SG MC				détail modifier
	En cours			Création site e-commerce	RO SG				détail modifier
2	En cours			Création site web vitrine	RO MC				détail modifier
3	En cours			Création templates newsletter [HTML / WEB / PDF] + envoi n°1	RO				détail modifier
	En cours			Création application [Phase 1]	SG				détail modifier
	En cours			Evolutions site web	RO SG MC				détail modifier
	En cours			1590 MODULO+ Base documentaire	MC				détail modifier
	En cours			Refonte site e-commerce	RO MC				détail modifier

www.moduloplus.com/bo/?lg=&r=intranet&m=production&a=projets

Figure 8 – Intranet de Modulo+

Voici les différentes maquettes proposées, réalisées sous Photoshop :

Modulo (+)

Bienvenue, Régis ! Se déconnecter Voir le site

Espace documentaire

Documentation technique

Snippets

HTML / CSS

Javascript

Bootstrap

Outils

Serveurs, hébergement, domaines

Ressources

Documents Internes

Snippets > HTML / CSS

Filtre 1 Filtre 2 Recherche Rechercher Effacer la recherche Ajouter

#	Position	Nom	Description	Projets liés	Créé le	Modifié le	Actions
1	1	Lorem ipsum dolor sit amet	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sed arcu...	456	00/00/0000	00/00/0000	Voir Éditer ✕
1	1	Lorem ipsum dolor sit amet	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sed arcu...	456	00/00/0000	00/00/0000	Voir Éditer ✕
1	1	Lorem ipsum dolor sit amet	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sed arcu...	456	00/00/0000	00/00/0000	Voir Éditer ✕
1	1	Lorem ipsum dolor sit amet	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sed arcu...	456	00/00/0000	00/00/0000	Voir Éditer ✕
1	1	Lorem ipsum dolor sit amet	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas sed arcu...	456	00/00/0000	00/00/0000	Voir Éditer ✕

Figure 9 - Maquette n°1

La maquette n°1 n'a pas été acceptée, le thème était trop sombre et les boutons trop clairs.

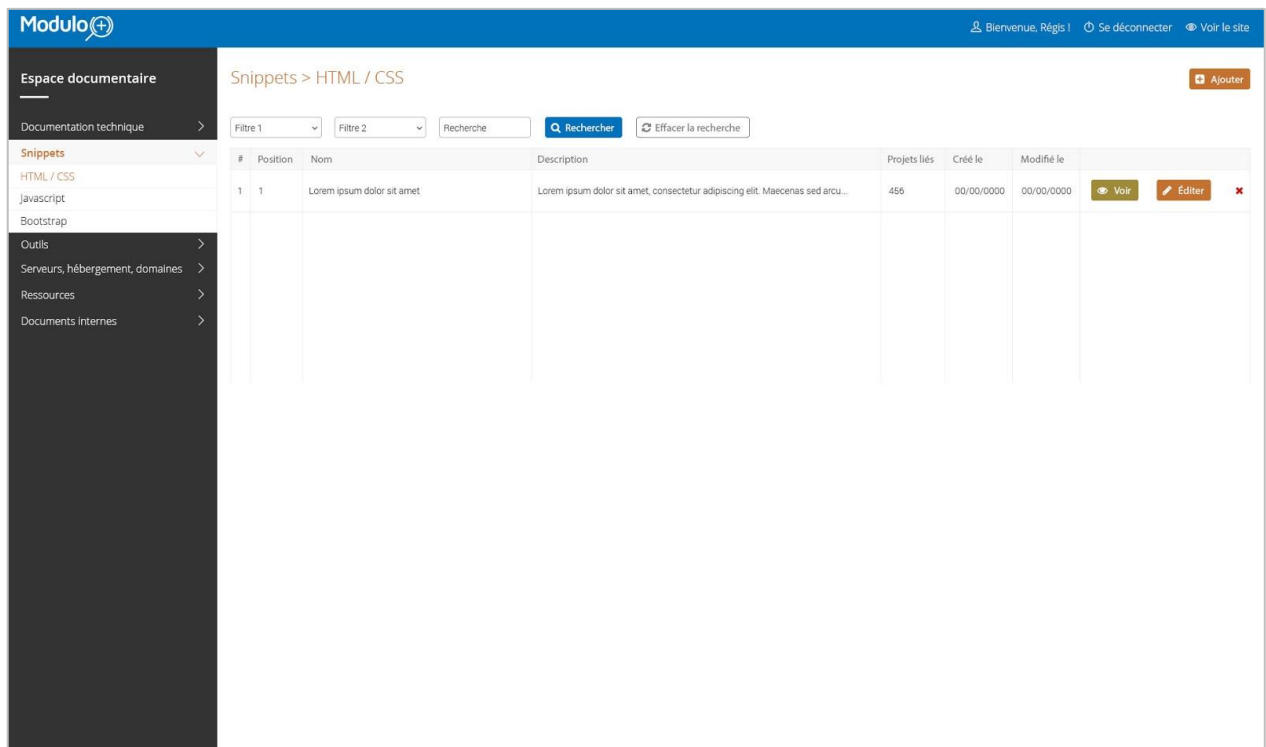


Figure 10 - Maquette n°2

La maquette n°2 n'a pas été acceptée, les couleurs étaient trop froides et les boutons manquaient de lisibilité.

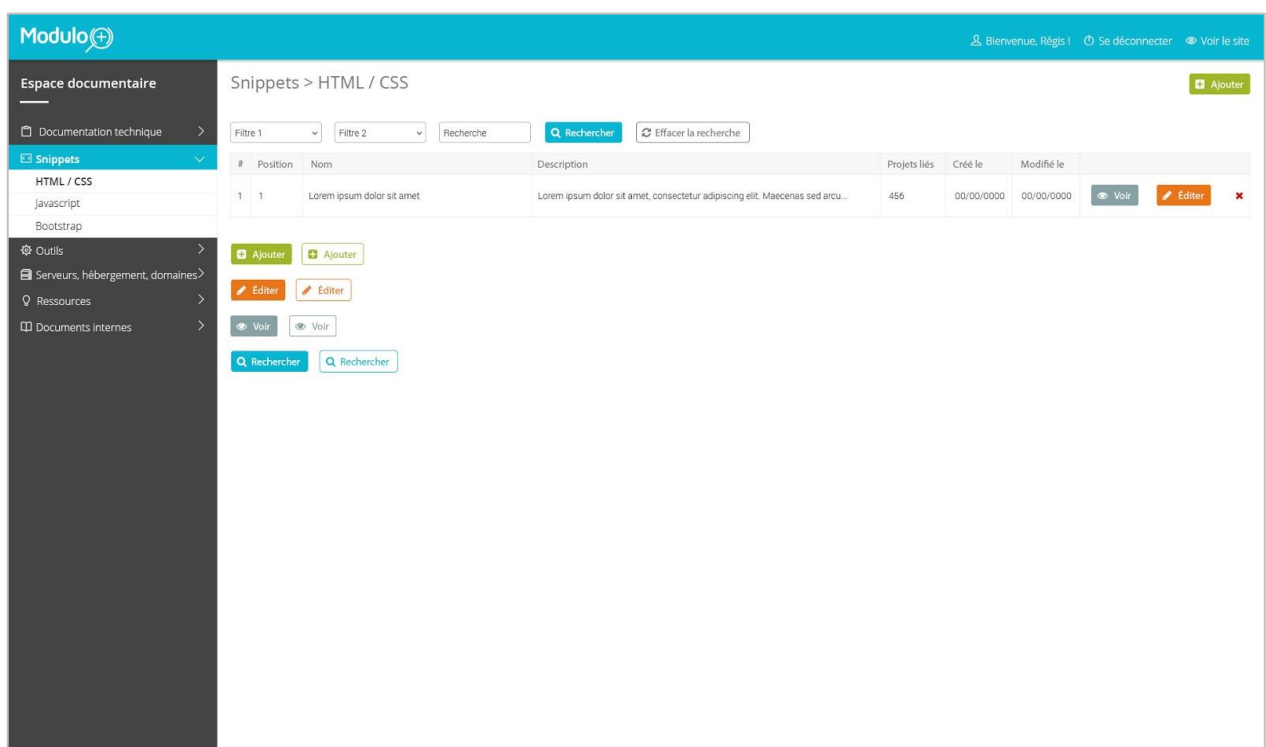


Figure 11 - Maquette n°3

La maquette n°3 a été choisie grâce au bandeau turquoise, qui rappelle le bleu de Modulo+ et donne un aspect moderne au tout. Il a été relevé qu'en plus du logo, la couleur de ce bandeau pourra être personnalisée si le thème est repris sur les intranets d'autres clients.

4. Réalisation

4.1. Environnements

Modulo+ travaille avec trois environnements :

- un local pour le développement (la machine Windows du développeur)
- un de pré-production pour les tests avant publication (une machine à distance identique à l'environnement final)
- un de production.

Les environnements de pré-production et de production sont sur un serveur loué à l'entreprise Alfa Safety. Le serveur est sur PHP 5.6, et possède une base de données sous MySQL 5.5 et Apache 2.4.6.

Le projet complet est versionné⁶ sur Bitbucket, via Git⁷.

Les environnements de pré-production et de production sont accessibles par FTP⁸.

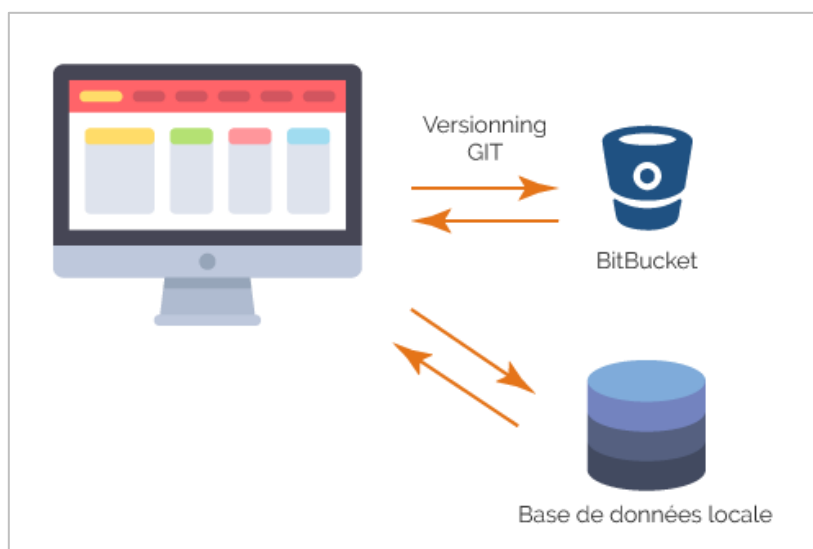


Figure 12 - Environnement local

⁶ La gestion de versions (*version control* ou *revision control*) consiste à maintenir l'ensemble des versions d'un ou plusieurs fichiers.

⁷ GIT : logiciel de gestion de versions décentralisé

⁸ FTP (File Transfer Protocol) : protocole de communication destiné à l'échange informatique de fichiers sur un réseau

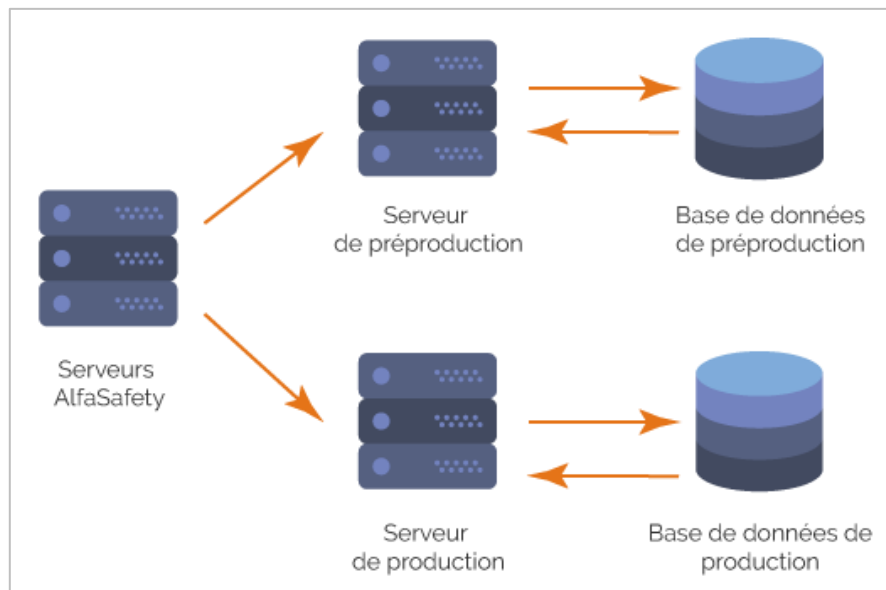


Figure 13 - Environnements de préproduction et de production

4.2. Technologies utilisées

Base de données



La base de données est sous MySQL 5.5 et Apache 2.4.6, et administrée avec PHPMyAdmin.

Développement



Les langages de programmation utilisés seront PHP 5.6, HTML 5, CSS 3, SASS et Javascript.

Frameworks



Les frameworks utilisés seront Slim (pour le routeur uniquement), Bootstrap pour le front-office, et PHPUnit pour les tests unitaires.

Versionning



Sur l'environnement en local, les fichiers seront versionnés sur BitBucket avec Git.

Logiciels



L'IDE utilisé sera PHPStorm. Le logiciel de transfert de fichiers par FTP sera Filezilla. Les logiciels de retouches et dessins seront Adobe Photoshop et Adobe Illustrator.

4.3. Arborescences

L'arborescence du site sera la suivante :

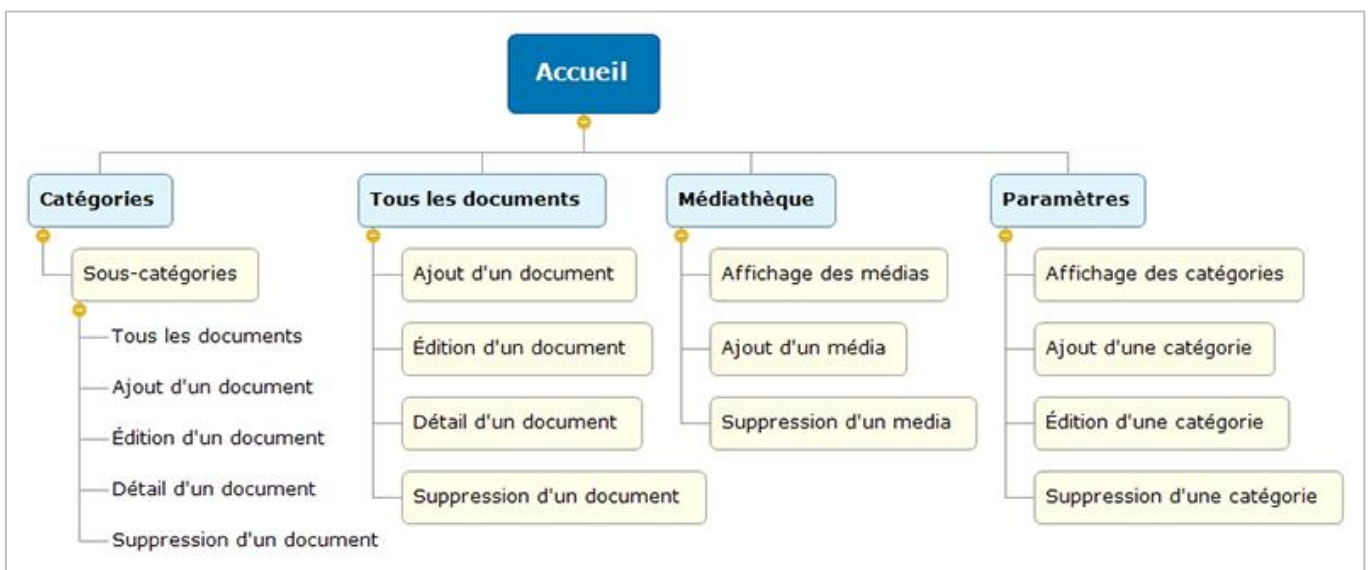


Figure 14 – Arborescence du site

Les documents seront triés et affichés par catégories, sous forme de liste. Il sera possible d'accéder au détail de chacun, de l'éditer ou de le modifier. Il y aura un accès à une page listant tous les documents, toutes catégories confondues. La page d'accueil affichera les dernières publications créées, les dernières publications modifiées et les publications à la une.

L'arborescence des fichiers de conception du site sera la suivante :

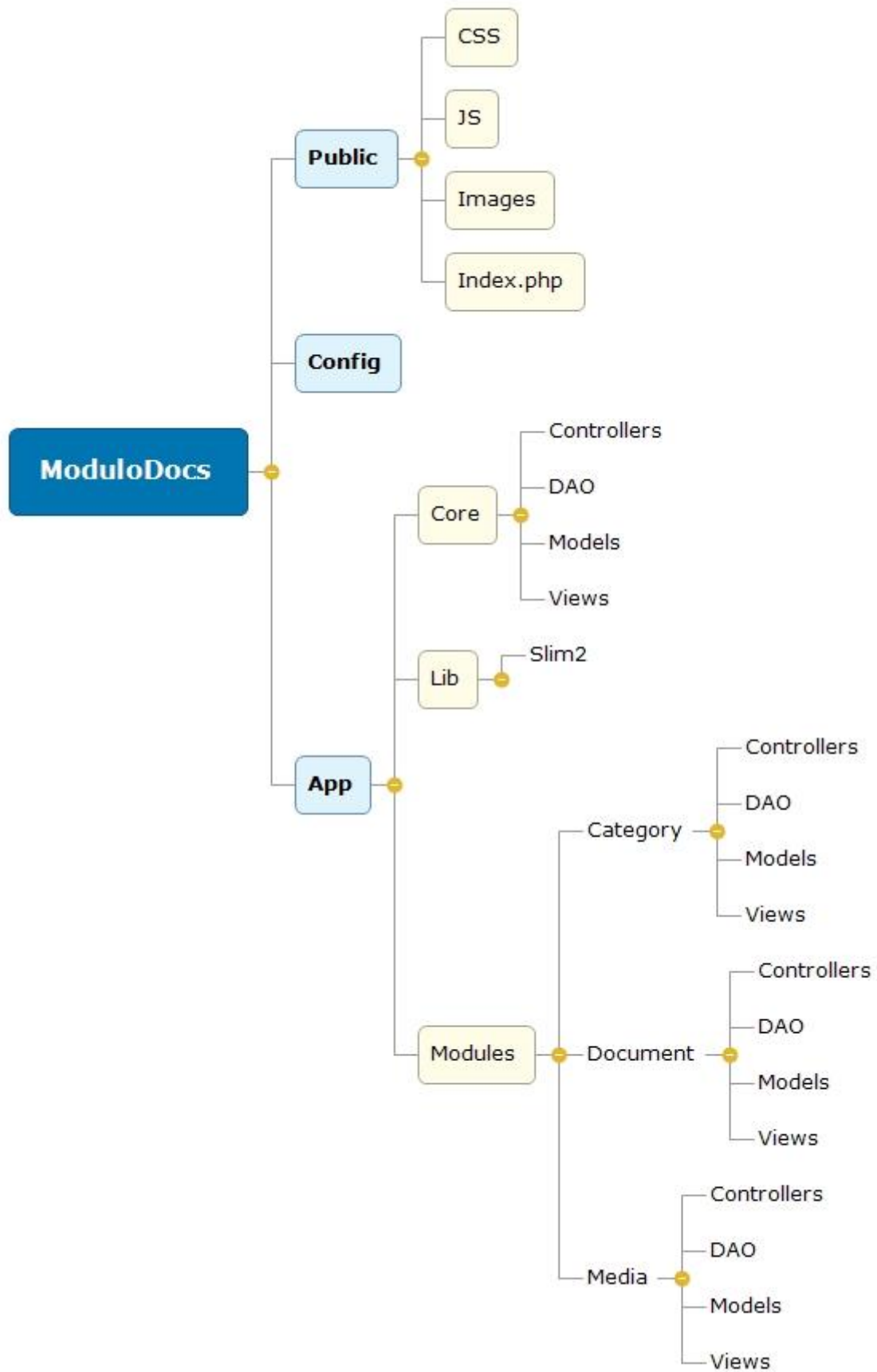


Figure 15 – Arborescence des fichiers de conception

L'arborescence des fichiers de conception du site comportera une partie publique, dans le dossier *public*, pour tous les fichiers auxquels les visiteurs pourront avoir accès (images, feuilles de style etc.), une partie *config* qui contiendra tous les fichiers de configuration (accès à la base de données, constantes de base etc.), et une partie *app* qui contiendra tous les fichiers constituant l'application proprement dite.

Cette partie *app* va se composer de modules correspondants aux différents packages du site (gestion des documents, catégories, médias), ainsi qu'un dossier *lib* contenant les librairies externes, et un dossier *test* contenant les tests unitaires.

Les fichiers correspondants aux modules reprendront une arborescence Modèle-Vue-Contrôleur, pour avoir une homogénéité entre les classes et les dossiers.

4.4. Base de données

La base de données est sous MySQL 5.5 et Apache 2.4.6, et administrée avec PHPMyAdmin.

Elle sera composée de trois tables, correspondant à trois packages du projet : documents, catégories et médias.

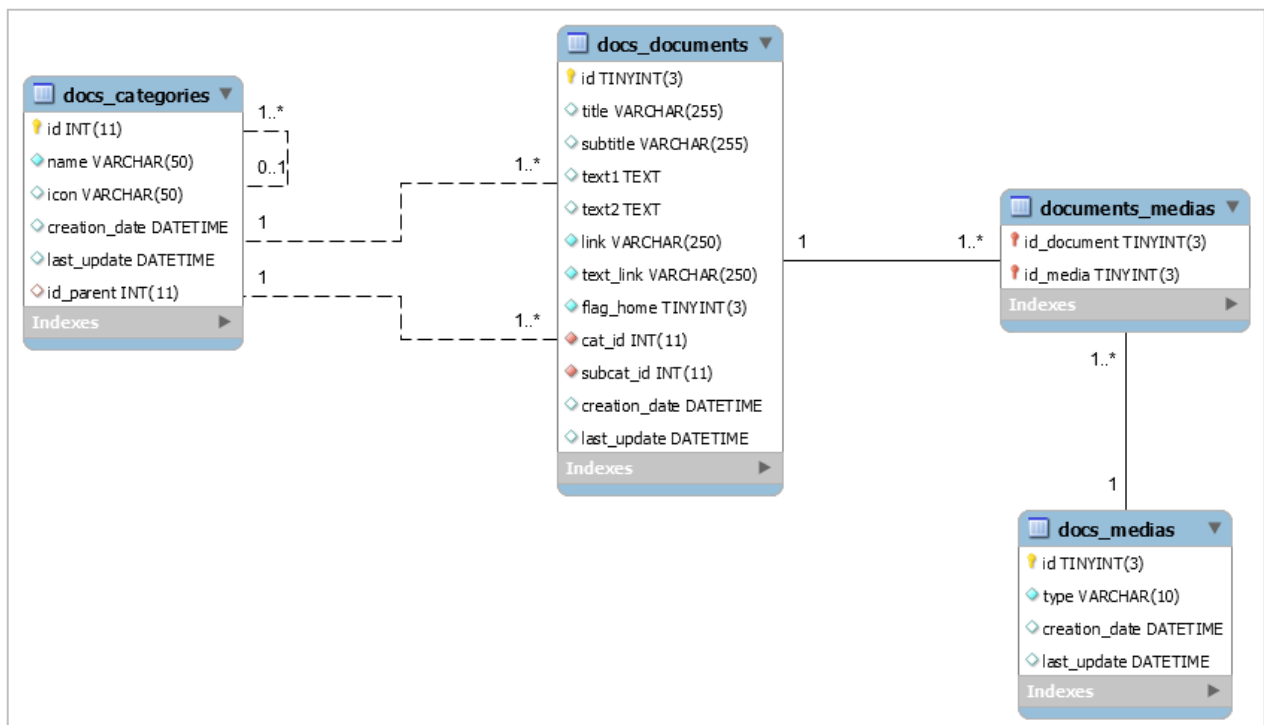


Figure 16 - Schéma de la base de données

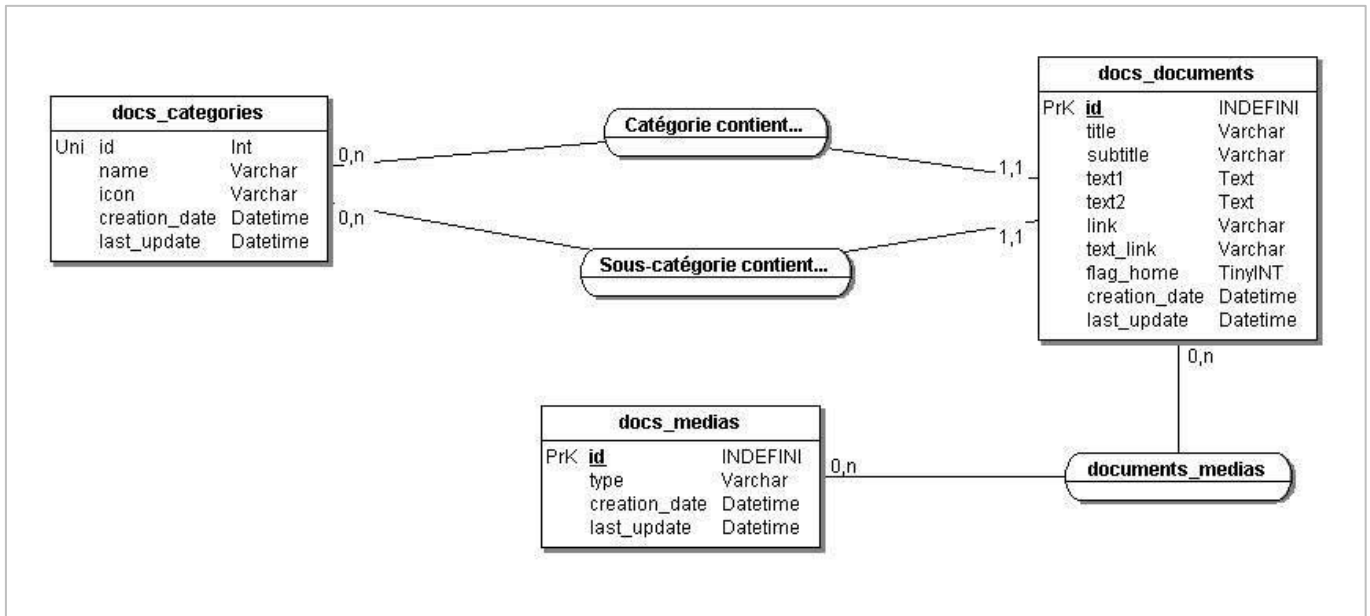


Figure 17 – Modèle conceptuel des données

Pour faciliter l'utilisation des catégories, qui apparaissent dans les documents et le menu, on utilisera une seule et même table qui aura une clé étrangère faisant référence à elle-même.

Pour associer les sous-catégories aux catégories, on utilisera un LEFT JOIN :

```

SELECT
    cat.id AS cat_id,
    cat.name AS cat_name,
    cat.icon AS cat_icon,
    GROUP_CONCAT(subcat.id) AS subcat_id,
    GROUP_CONCAT(subcat.name) AS subcat_name
FROM docs_categories AS cat
    LEFT JOIN docs_categories AS subcat
        ON subcat.id_parent = cat.id
WHERE cat.id_parent IS NULL
GROUP BY cat.id
ORDER BY cat_id
  
```

Cas particulier : les médias

Les médias peuvent être associés à un ou plusieurs documents, et un document peut avoir un ou plusieurs médias : c'est une relation *many-to-many*, on fait donc une table d'association entre les

documents et les médias. En cas de suppression d'un document ou d'un média, seule la table d'association sera impactée.

5. Architecture

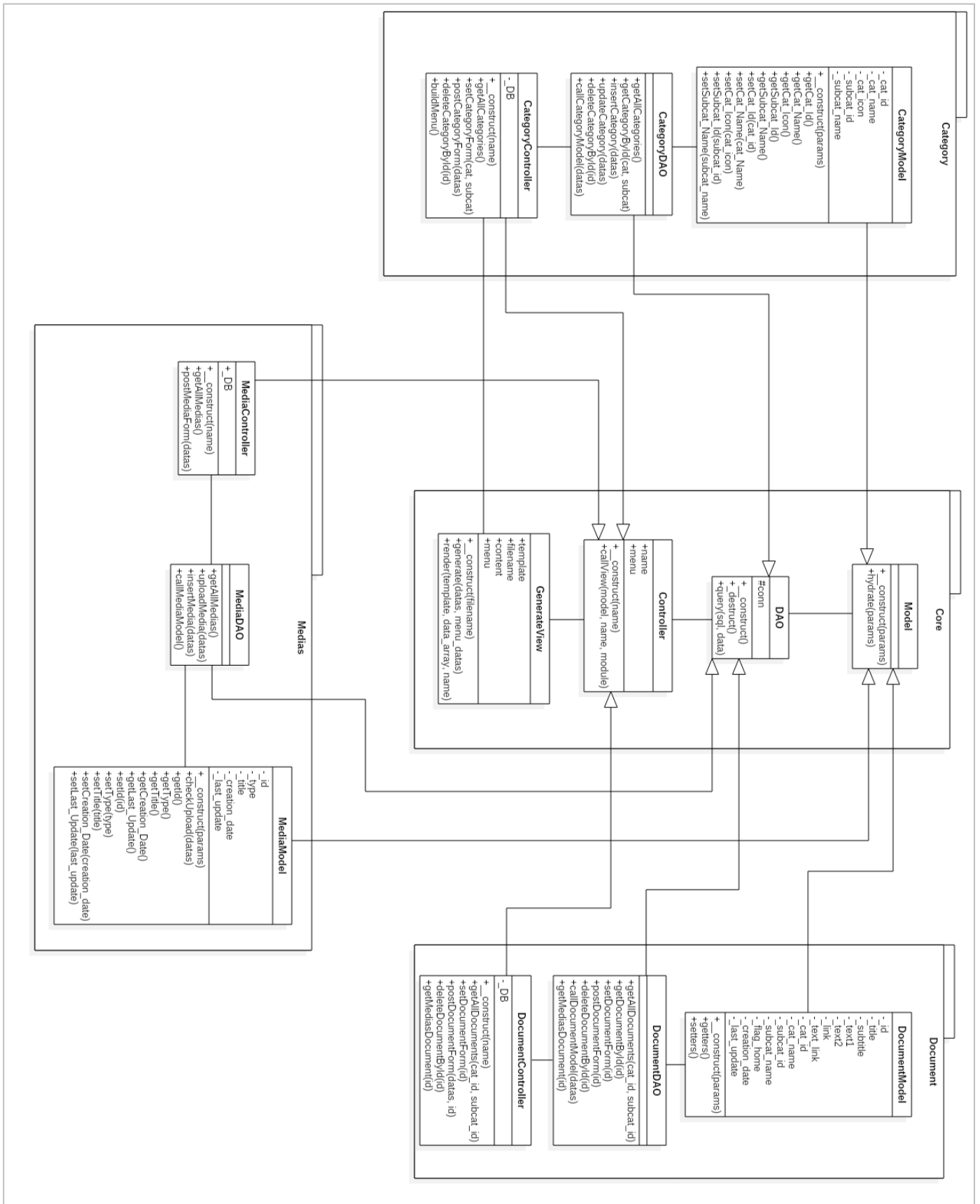


Figure 18 - Diagramme de classes

Les classes sont divisées en quatre packages : un package principal, Core, qui contiendra toutes les classes globales, et trois packages enfants, Categories, Documents, Medias. Il y a également le package PDO, une extension de PHP qui sert d'interface d'accès aux données.

Le patron de conception ou *design pattern*⁹ utilisé sera le MVC web (Modèle Vue Contrôleur). Cette architecture permet de séparer les responsabilités : le modèle s'occupera de la vérification des données, la vue s'occupera de l'affichage, et le contrôleur sera l'élément central qui prendra les décisions et fera l'intermédiaire entre le modèle et la vue. Il y aura également un DAO, qui s'occupera des requêtes SQL et des connexions à la base de données.

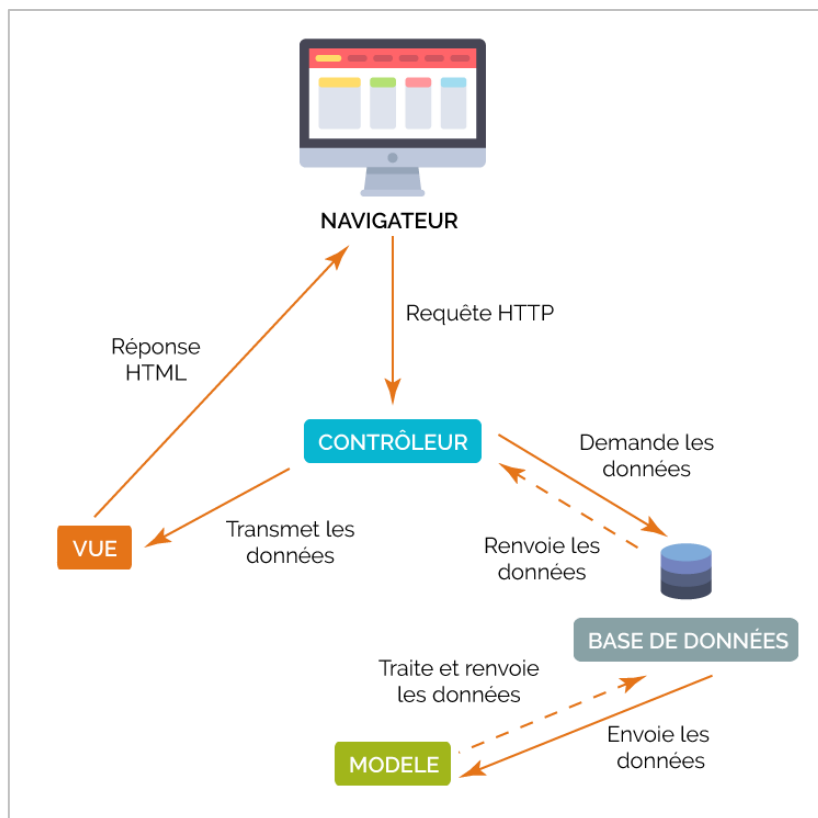


Figure 19 – Schéma MVC

5.1. Le routeur

Le document root du serveur apache2 est le dossier *public*. Seul le fichier `index.php` est exposé à l'utilisateur. Cette façon de procéder n'expose pas les fichiers composant l'application.

Pour faciliter l'affichage des vues sans surcharger le fichier `.htaccess`, un routeur sera mis en place. Un routeur est un contrôleur frontal qui permet différentes actions en fonction des URL et des requêtes HTTP (GET, POST, PUT etc). Pour ModuloDocs, le routeur utilisé sera Slim 2, pour sa facilité d'utilisation et sa documentation très détaillée.

⁹ Design pattern (ou patrons de conception) : arrangement récurrent de rôles et d'actions joués par des modules d'un logiciel

Pour que le routeur puisse fonctionner, toutes les pages sont redirigées vers le fichier index.php. La page index contiendra toutes les routes et se chargera de faire les actions correspondantes (par exemple : faire appel à une méthode avec un id en paramètre).

La redirection se fait dans le .htaccess :

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ index.php?url=$1 [QSA,L]
```

Dans index.php, le routeur est instancié.

```
/*create a new Slim instance, with debug on (for dev ONLY)*/
$app = new \Slim\Slim([
    "debug" => true
]);
```

Un objet Slim est créé avec la condition “debug” à “true” : c’est uniquement pour la phase de développement, afin d’avoir des messages d’erreurs précis et développés. Par défaut, cette condition est à “false”.

Il ne reste plus qu’à indiquer les routes. Ci-dessous, un exemple avec la page d’accueil :

```
$app->get('/', function(){
    $controller = new
    \Modules\Document\Controllers\DocumentController();
    $controller->getDocumentsHome();
});
```

Le routeur récupère l’URL passée en paramètre, instancie un contrôleur puis appelle la méthode getAllHome(), qui permet d’afficher la vue de la page.

En fonction de la requête HTTP, il va faire appel à la méthode get() pour GET, post() pour POST etc.

5.2. Le DAO

Le DAO permet de séparer la couche “métier” (la façon dont les données sont stockées) de la couche “données” (les données auxquelles on accède). Le DAO sera le seul à avoir un accès direct aux données et à la base de données, ce qui permet d’avoir un système plus sécurisé. Il est en lien avec le contrôleur et le modèle.

Dans ModuloDocs, le DAO “parent” se trouve dans le package Core. Il instancie la connexion à la base de données avec PDO. Les DAO enfants contiendront les requêtes que le DAO parent exécutera.

L'extension PDO (PHP Data Objects) est une interface qui permet d’accéder à une base de données depuis PHP.

```
namespace Core\DAO;
use \PDO;

/**
 * Class DAO
 * @package Core\DAO
 * Instantiates database connexion
 */

class DAO
{
    /**
     * @var PDO
     */
    protected static $conn;

    /**
     * DAO constructor
     */
    public function __construct(){
        if (!isset(self::$conn)) {
            try {
                self::$conn = new PDO('mysql:host=' . DB_HOST . ';dbname=' . DB_NAME ,
                DB_USER , DB_PASS, array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8'));
                self::$conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            } catch (PDOException $e) {
                $msg = 'ERREUR PDO dans ' . $e->getFile() . ' L.' . $e->getLine() . ' :
                ' . $e->getMessage();
                die($msg);
            }
        } //endif
    }
}
```

Ici, le constructeur vérifie qu’il n’y a aucune autre connexion en cours, puis essaie de se connecter à la base de données. En cas de problème dans la connexion, un message d’erreur s’affichera et tous les processus en cours seront stoppés.

Les informations de connexions sont enregistrées dans des constantes, à partir d'un fichier de configuration.

Une deuxième méthode, `query()`, va permettre de lancer les requêtes envoyées par les classes enfants et de retourner des données.

```
/**
 * @param $sql
 * @param array $data
 * @return array|int|string
 * execute a prepared query and return a datas array
 */
public function query($sql, $data = []){
    $sql = trim($sql);
    $stmt = self::$conn->prepare($sql);
    foreach ($data as $name => $value) {
        $stmt->bindValue(':'. $name, $value, PDO::PARAM_STR);
    }
    $stmt->execute();
    $type = strtoupper(substr(trim($stmt->queryString), 0, 6));
    if($type=='SELECT') return $stmt->fetchAll(PDO::FETCH_ASSOC);
    if($type=='INSERT') return $stmt->rowCount();
    return self::$conn->lastInsertId();
}
```

La méthode reçoit une requête SQL, et éventuellement un tableau de données en paramètres. PDO prépare la requête, c'est-à-dire qu'elle la stocke temporairement en mémoire : c'est le principe des **requêtes préparées**, très importantes pour la sécurité puisqu'elles permettent d'éviter les injections SQL¹⁰.

S'il y a un tableau de données en paramètres, on lie les valeurs du tableau aux paramètres de la requête préparée.

Par exemple, la requête suivante contient un paramètre "id" :

```
SELECT id, name FROM docs_categories WHERE id = :id
```

Si dans le tableau envoyé à la méthode `query()` on indique "id = 1", PDO ira remplacer le paramètre "id" par la valeur correspondant à "id" dans le tableau et exécutera la requête sous cette forme :

```
SELECT id, name FROM docs_categories WHERE id = 1
```

¹⁰ Cf 6.2. Sécurité

Voici d'autres types de requêtes préparées :

```
INSERT INTO docs_categories (id, name, icon, creation_date,
last_update, id_parent) VALUES (:id, :name, :icon, NOW(), NULL,
:id_parent)
```

INSERT INTO, qui permet d'insérer des valeurs dans les tables.

```
UPDATE docs_categories
SET name = :name, icon = :icon, last_update = NOW(), id_parent =
:id_parent
WHERE id = :id
```

UPDATE, qui permet de modifier des données dans les tables.

```
DELETE FROM docs_categories WHERE id = :id
```

DELETE qui permet d'effacer des données dans les tables.

Si la requête est correcte, elle est exécutée. En fonction du type de requête (SELECT, INSERT, UPDATE, DELETE), la méthode retournera un tableau de données ou un id :

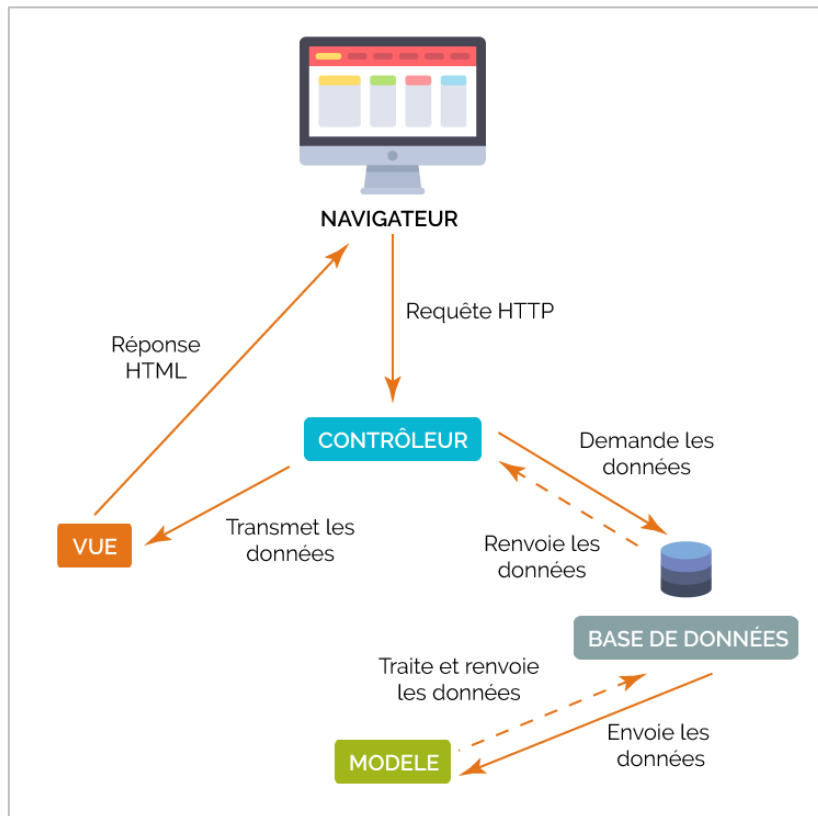
- SELECT : Tableau de données
- INSERT : Id du dernier élément inséré
- UPDATE : Id du dernier élément mis à jour
- DELETE : Id du dernier élément supprimé

5.3. Le MVC

Dans ModuloDocs, l'architecture MVC s'appliquera de la façon suivante :

- Le routeur instancie le contrôleur.
- Si on insère des données dans la base (à partir d'un formulaire, par exemple) :
 - Le contrôleur instancie le DAO avec les données en paramètre.
 - Le DAO renvoie les données au modèle, qui les vérifie.
 - Si les données sont correctes, le modèle renvoie un tableau d'objet au DAO
 - Le DAO insère les données dans la base de données et transmet le tableau d'objet au contrôleur.

- Si on sélectionne des données sans insertion dans la base de données :
 - Le contrôleur instancie le DAO.
 - Le DAO va lancer la requête en base de données.
 - Il envoie les données récupérées depuis la base au modèle, qui va vérifier les données.
 - Si les données sont correctes, le modèle renvoie un tableau d'objet au DAO
- Le DAO transmet le tableau d'objet au contrôleur.
- Le contrôleur envoie les données à la vue, qui va générer le HTML et afficher les données dans le navigateur.



Pour expliquer comment fonctionne cette architecture et comment elle s'applique dans ModuloDocs, nous allons nous baser sur deux cas d'utilisation : l'affichage d'un document et la création d'un document.

Cas d'utilisation n°1 : l'affichage d'un document

Le schéma du cas d'utilisation ci-dessous montre le parcours de l'utilisateur pour accéder à l'affichage d'un document.

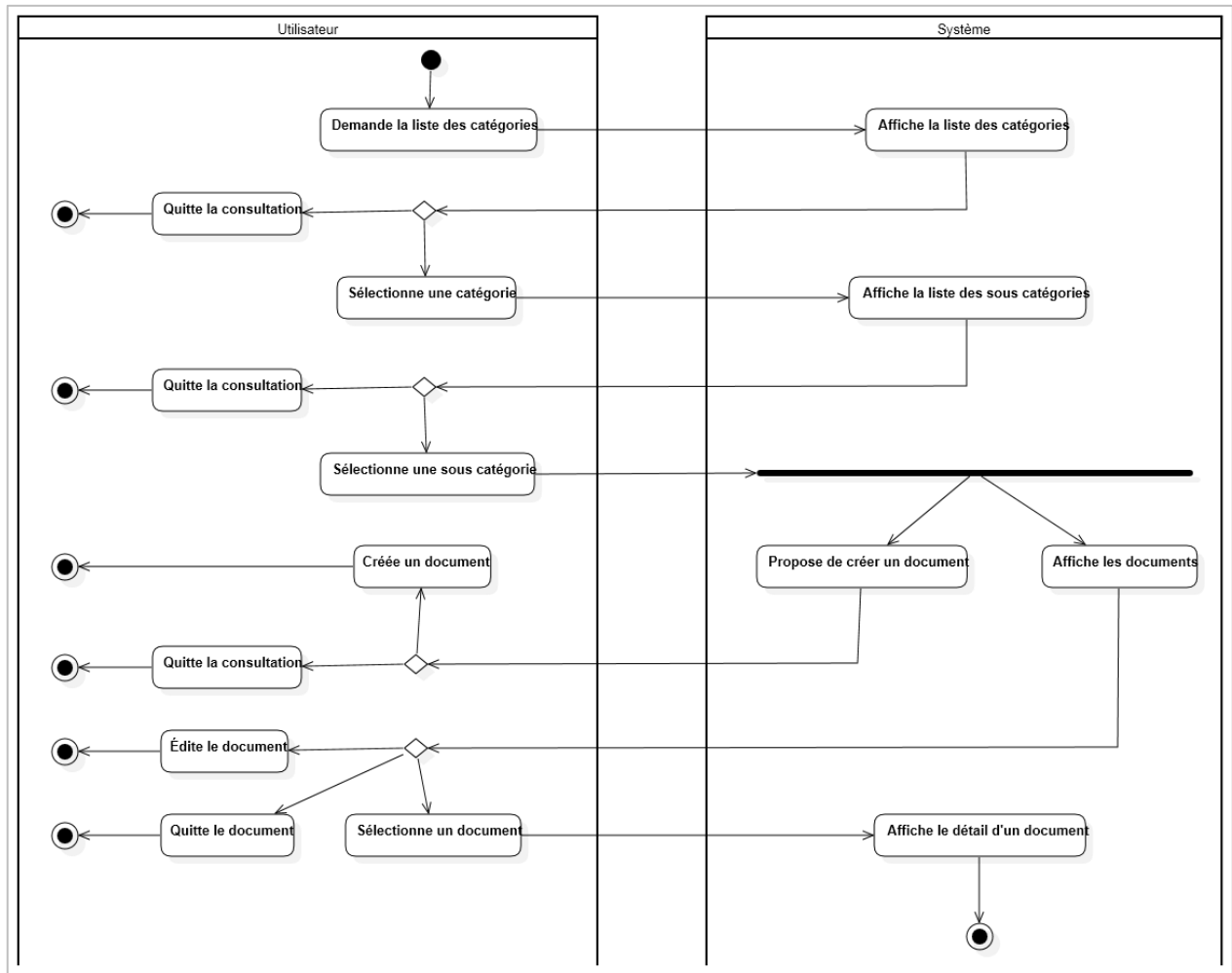


Figure 20 – Cas d'utilisation : Affichage du détail d'un document

Scénario nominal

1. Le système affiche une page contenant la liste des catégories.
2. L'utilisateur sélectionne une des catégories.
3. Le système affiche une page contenant la liste des sous-catégories.
4. L'utilisateur sélectionne une des sous-catégories.
5. Le système recherche les documents appartenant à cette sous-catégorie.
6. Le système affiche une liste des documents trouvés, avec leur titre et sous-titre.
7. L'utilisateur peut sélectionner un des documents affichés.
8. Le système affiche les détails du document sélectionné.
9. L'utilisateur peut ensuite quitter le détail du document.
10. Le système retourne à l'affichage de la liste des documents.

Scénarios alternatifs

1. 2.a. L'utilisateur quitte la consultation de la catégorie choisie.
2. 4.a. L'utilisateur quitte la consultation de la sous-catégorie choisie.
3. 6.a. Le système ne trouve aucun document, il propose d'en créer un, l'utilisateur décide d'en créer un. Cf Cas n°2

4. 6.b. L'utilisateur quitte la consultation.
5. 7.a. L'utilisateur quitte la consultation de la catégorie choisie.
6. 7.b. L'utilisateur créé, édite ou supprime un des documents affichés. Cf Cas n°2
7. 9.a. L'utilisateur édite le document qu'il consulte. Cf Cas n°2

A partir de ce cas d'utilisation, nous avons pu en déduire le diagramme de séquence ci-dessous, qui reprend l'architecture Modèle Vue Contrôleur.

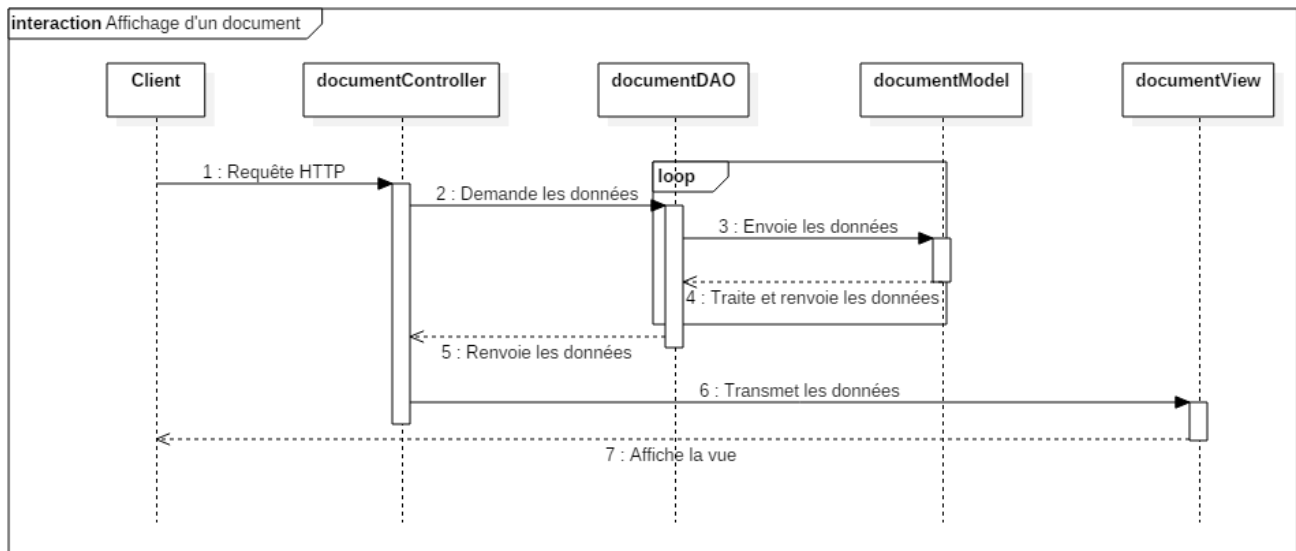


Figure 21 – Diagramme de séquence pour l'affichage d'un document

Le routeur

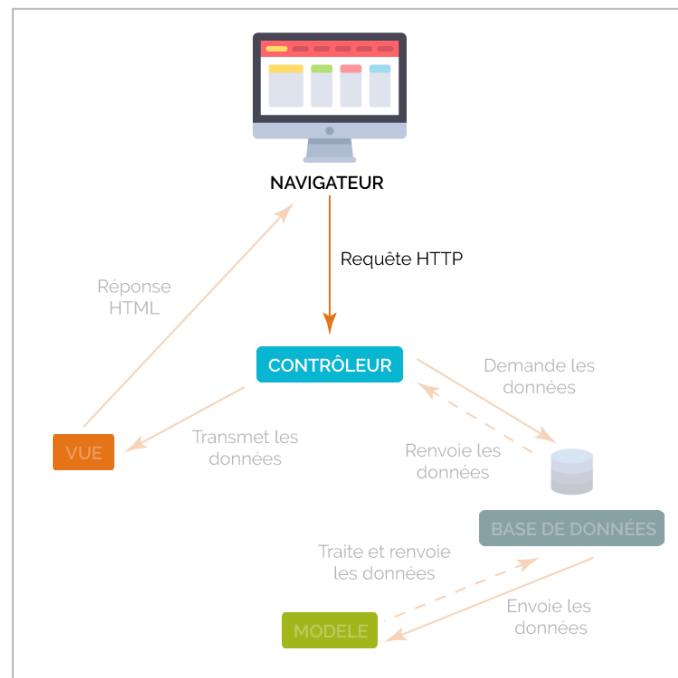


Figure 22 – MVC : la requête HTTP

L'accès au détail d'un document se fait via cette url : www.moduloplus.fr/document/detail/14, où « 14 » correspond à l'id du document.

La route correspondant à cette url est créée : ici, elle concerne les url commençant par '/document' suivi de l'id d'un document. Le paramètre id est traduit de cette façon : `'/document/:id'`.

La route récupère l'id, puis instancie le contrôleur et fait appel à la méthode `getDocumentById()` avec l'id en paramètre

```

$app->get('/document/detail/cat-:cat_id/subcat-:subcat_id/doc-:doc_id', function ($cat_id, $subcat_id, $doc_id){
    $controller = new
    \Modules\Document\Controllers\DocumentController($cat_id, $subcat_id);
    $controller->getDocumentById($doc_id);
});
  
```

Pour toutes les routes respectant ce schéma, le routeur va instancier le contrôleur `documentController()`, puis appeler la méthode `getDocumentById()` en lui passant en paramètre l'id du document.

Le contrôleur

```

/**
 * Class Controller
 * @package Core\Controllers
 */
class Controller
{
    private $_cat_id;
    private $_subcat_id;
    private $_page;

    /**
     * Controller constructor.
     * @param $name
     */
    public function __construct($name, $cat_id = null, $subcat_id =
null){
        $this->_cat_id = (int)$cat_id ? $cat_id : null;
        $this->_subcat_id = (int)$subcat_id ? $subcat_id : null;
        if(class_exists($name)){
            $this->_page = new \ReflectionClass($name);
            $this->_page = explode('Controller', $this->_page-
>getShortName());
        } else {
            $this->_page = $name;
        }
        return $this->_page;
    }
}

```

Le constructeur du contrôleur parent va définir un attribut `$_page`, qui correspondra au nom de la vue que l'on affichera. Ce nom correspond soit au nom de la classe sans le terme « Controller », soit à un nom transmis lors de l'instanciation. Il va également récupérer les id de la catégorie et sous-catégorie courante, pour le menu.

La récupération des données

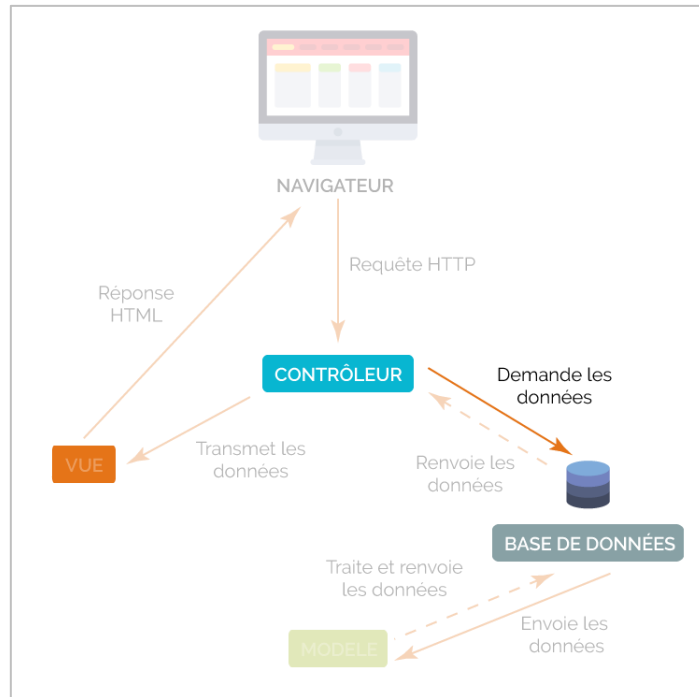


Figure 23 – MVC : l'appel à la base de données

La classe `documentController()` a un attribut privé `$_DB`, qui crée un objet d'accès aux données (`documentDAO()`), et un autre, `$_DB_cat`, qui va créer un objet d'accès aux données des catégories (pour l'affichage des formulaires).

```

/**
 * Class DocumentController
 * @package Modules\Document\Controllers
 */
class DocumentController extends Controller
{

    /**
     * @var DocumentDAO
     */
    private $_DB;
    /**
     * @var CategoryDAO
     */
    private $_DB_cat;

    /**
     * DocumentController constructor.
     * @param null $name
     */
    public function __construct($cat_id = null, $subcat_id = null,
$name = null)
    {
        $page = ($name == null) ? __CLASS__ : $name;
        parent::__construct($page, substr($cat_id, 0, 4),
substr($subcat_id, 0, 7));
        $this->_DB = new DocumentDAO();
        $this->_DB_cat = new CategoryDAO();
    }
}

```

La méthode `getDocumentById()` du contrôleur va faire appel au DAO pour récupérer et vérifier les données.

Le contrôleur :

```

/**
 * @param $id
 */
public function getDocumentById($id){
    $datas = $this->_DB->getDocumentById($id);
    if(is_array($datas)) {
        $this->callView($datas, 'document', 'document');
    } else {
        echo $datas;
    }
}

```

Le DAO :

```
/**
 * @param null $id
 * @return array|string
 */
public function getDocumentById($id = null){
    if($id == null){
        $model = [];
        $model[] = new documentModel();
        return $model;
    } else {
        $query = 'SELECT
doc.id, doc.title, doc.subtitle, doc.link, doc.text_link,
doc.creation_date, doc.last_update,
cat.id AS cat_id, cat.name AS cat_name,
subcat.id AS subcat_id, subcat.name AS subcat_name
FROM docs_documents AS doc
LEFT JOIN docs_categories AS cat
ON cat.id = doc.cat_id
LEFT JOIN docs_categories AS subcat
ON subcat.id = doc.subcat_id
WHERE doc.id = :id';
        $datas = array('id' => $id);
        $sql = $this->query($query, $datas);
        try {
            $model = $this->callDocumentModel($sql);
            return $model;
        } catch(\InvalidArgumentException $e) {
            return $e->getMessage();
        }
    }
}
```

On y retrouve la méthode query() décrite plus haut, dans laquelle on passe l'id du document en paramètre. La méthode renvoie un tableau de tableau de données.

```

array (size=1)
0 =>
  array (size=8)
    'id' => string '5' (length=1)
    'title' => string 'Lorem ipsum' (length=11)
    'subtitle' => string 'Lorem ipsum dolor' (length=18)
    'text1' => string 'Lorem ipsum dolor sit amet' (length=26)
    'text2' => string 'Ut sed tincidunt ipsum.' (length=23)
    'cat_id' => string '2' (length=1)
    'subcat_id' => string '11' (length=2)
    'flag_home' => string '0' (length=1)

```

Le traitement des données

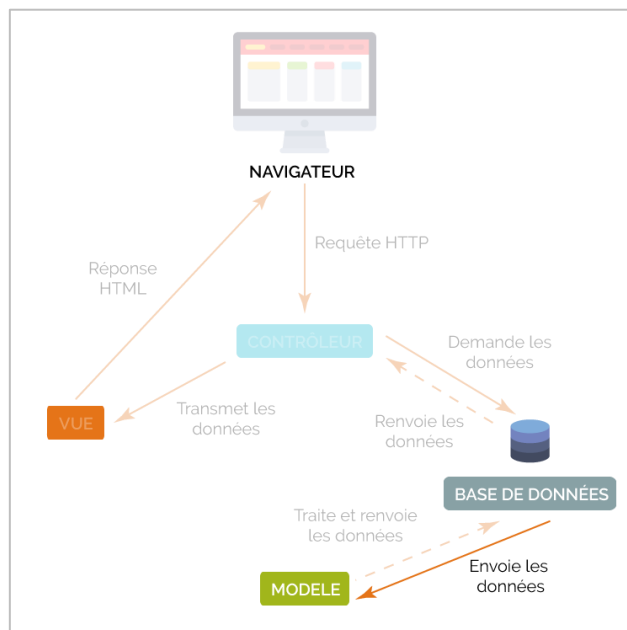


Figure 24 – MVC : Envoi des données pour le traitement

Le DAO va appeler la méthode `callDocumentModel()`, avec les données en paramètres.

Pour l'appeler, il va utiliser un « try... catch », qui permet de gérer les erreurs avec la classe PHP Exception¹¹.

En cas d'erreur, PHP retourne une erreur fatale avec un message : le processus en cours s'arrête net (la vue ne s'affiche pas, les données ne sont pas transmises à la base etc).

¹¹ Cf 6.2. Sécurité

```

public function getDocumentById($id){
    (...)
    try {
        $model = $this->callDocumentModel($sql);
        return $model;
    } catch(\InvalidArgumentException $e) {
        return $e->getMessage();
    }
}

/**
 * @param $datas
 * @return array
 */
public function callDocumentModel($datas)
{
    $dataArray = [];
    foreach ($datas as $data){
        $dataArray[] = new documentModel($data);
    }
    return $dataArray;
}

```

La méthode callDocumentModel() va instancier un documentModel() pour chaque tableau de données.

```

/**
 * Class Model
 * @package Core\Models
 */
class Model
{
    /**
     * Model constructor.
     * @param array $params
     */
    public function __construct($params = []){
        if (!empty($params)) {
            $this->hydrate($params);
        }
    }

    /**
     * @param array $params
     */
    public function hydrate($params = []){
        foreach ($params as $method => $value) {
            $setter = "set" . ucfirst($method);

            if (method_exists($this, $setter)) {
                $this->$setter($value);
            }
        }
    }
}

```



```

/**
 * Class DocumentModel
 * @package Modules\Document\Models
 */
class DocumentModel extends Model
{
    private $_id;
    private $_title;
    private $_subtitle;
    private $_text1;
    private $_text2;
    private $_link;
    private $_text_link;
    private $_cat_id;
    private $_cat_name;
    private $_subcat_id;
    private $_subcat_name;
    private $_flag_home = 0;
    private $_creation_date;
    private $_last_update;

    /**
     * DocumentModel constructor.
     * @param array $params
     */
    public function __construct($params = []){
        parent::__construct($params);
    }
}

```

La méthode hydrate() du constructeur permet de faire passer les données dans les setters de la classe enfant.

Accesseur et mutateur (ou Getter et setter en anglais)

Dans un modèle PHP, un setter est une méthode qui permet de vérifier les données avant que le DAO ne les stocke. Le getter va renvoyer les données, après les avoir éventuellement ajustées (par exemple, changer une date au format local).

Le setter va faire les vérifications, par exemple sur le type de données et la longueur des chaînes.

```

/**
 * @param $title
 */
public function setTitle($title){
    if(is_string($title) && strlen($title) > 0) {
        $this->_title = $title;
    } else {
        throw new \InvalidArgumentException("Le titre doit être du
texte");
    }
}

```

En cas d'erreur, PHP retourne une erreur fatale avec un message : le processus en cours s'arrête (la vue ne s'affiche pas, les données ne sont pas transmises à la base etc).

Si tout va bien, le getter va renvoyer la valeur au DAO :

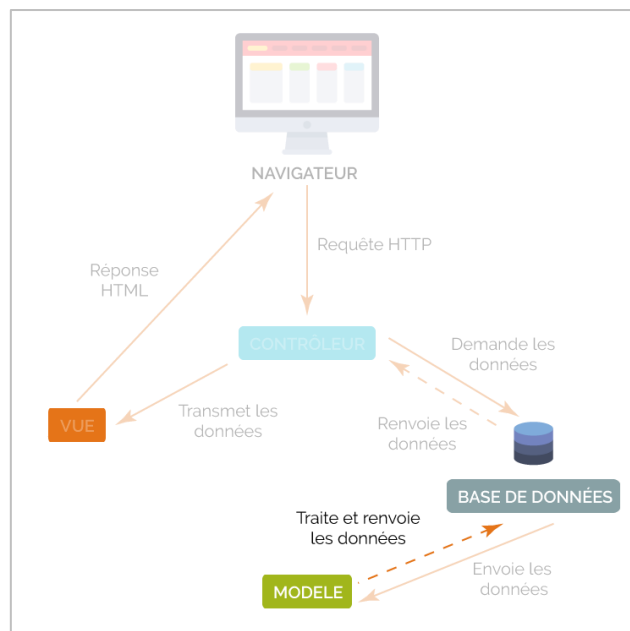


Figure 25 – MVC : traitement des données

```

/**
 * @return mixed
 */
public function getTitle(){
    return $this->_title;
}

```

Sinon, il renvoie un message d'erreur.

Le DAO va pouvoir renvoyer les données au contrôleur (ou le message d'erreur, le cas échéant), qui va le transmettre à la vue.

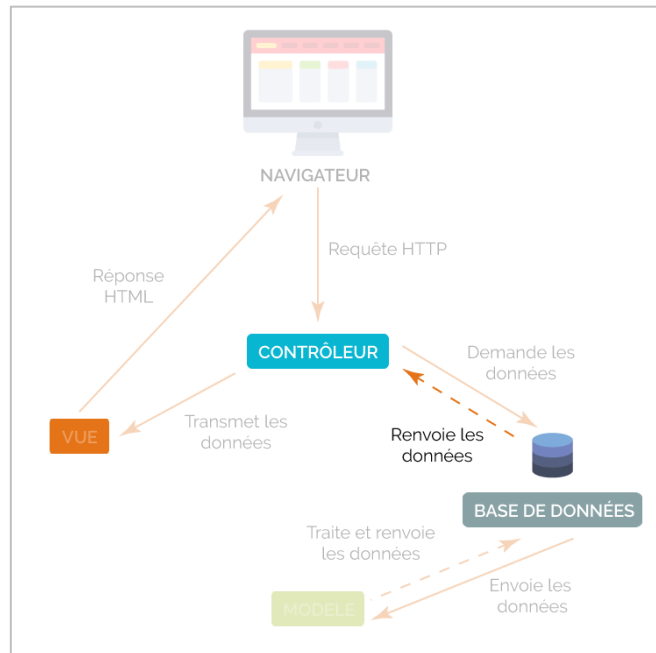


Figure 26 – MVC : renvoi des données traitées

L'affichage de la vue

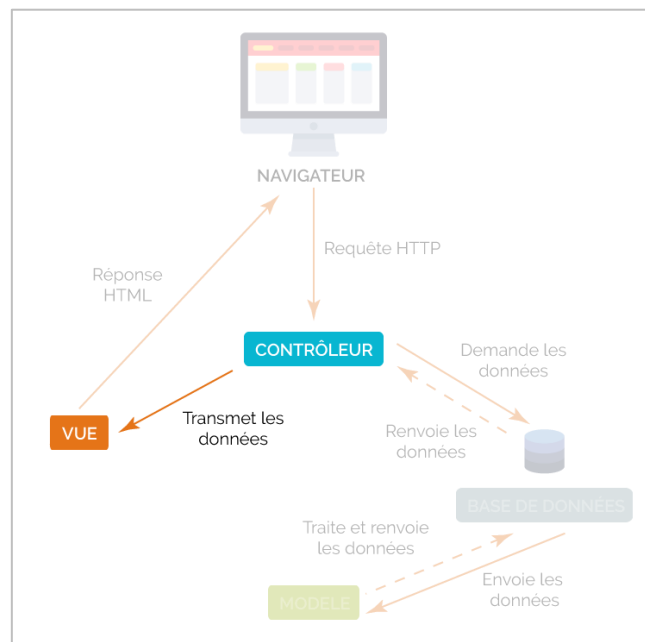


Figure 27 – MVC : la préparation de la vue

Le contrôleur va transférer les données à la vue avec la méthode `callView()` du contrôleur parent.

```

/**
 * @param $model
 * @param null $name
 * @param null $module
 */
public function callView($model, $name = null, $module = null){
    if($name) $this->_page = $name;
    $menu = new CategoryController();
    $menu_model = $menu->buildMenu();
    $view = new generateView($this->_page, $module);
    $view->generate($model, $menu_model, $this->_cat_id, $this->_subcat_id);
}

```

Cette méthode crée d'abord le menu et récupère les données le concernant (le menu étant sur toutes les pages, il est créé dans toutes les vues).

Dans l'ordre, la méthode :

- Récupère le nom de la vue
- Instancie le contrôleur du menu
- Fait appel à la méthode pour construire le menu
- Instancie le générateur de la vue
- Lance la méthode pour générer la vue

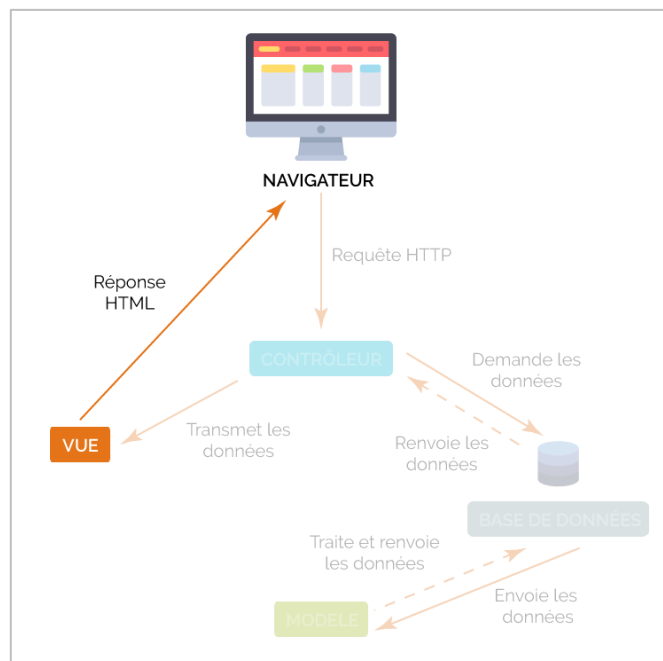


Figure 28 – MVC : le rendu HTML de la vue

Elle instancie ensuite la classe `generateView()`. La classe va d'abord générer le HTML du contenu, à partir du template correspondant. Ensuite, elle va générer le HTML du menu, puis intégrer les deux dans le layout principal.

On peut alors afficher la vue.

Étape 1 :
Génération du HTML
pour le contenu principal

Tous les documents

[Ajouter un document](#)

#	Titre	Sous-titre	
5	testtest		Detail Modifier Supprimer
7	sans sous titre		Detail Modifier Supprimer
8	sans sous titre		Detail Modifier Supprimer
9	sans sous titre		Detail Modifier Supprimer
11	Alquam blandit sed sapien eu fringilla		Detail Modifier Supprimer

Étape 2 :
Génération du HTML
pour le menu

- Accueil
- Tous les documents
- Documentation technique
- Snippets
- Outils
- Serveurs, hébergements...
- Ressources
- Documents internes
- Paramètres

Étape 3 :
Intégration du HTML dans le layout

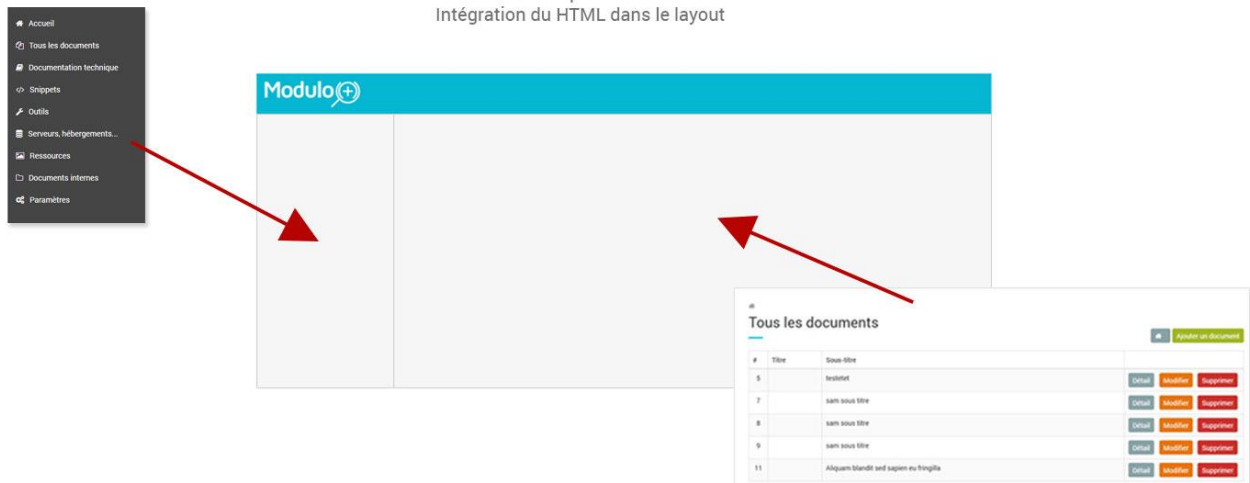


Figure 29 – Intégration de la vue

```

/**
 * generate method
 * @param $datas
 * @param $menu_datas
 */
public function generate($datas, $menu_datas, $cat_id = null,
$subcat_id = null){
    $content = $this->render($this->_content, $datas, $this->_datas);
    $menu = $this->render($this->_menu, array('menu' => $menu_datas,
'cat_id' => $cat_id, 'subcat_id' => $subcat_id, 'page' => $this-
>_page), 'menuData');
    $view = $this->render($this->_layout, array('menu' => $menu,
'content' => $content, 'page' => $this->_page), 'viewData');
    echo $view;
}

```

La méthode `render()` permet de générer le HTML pour chaque contenu : elle récupère le tableau de données, vérifie qu'un template correspondant au contenu existe, et extrait les données dans un tableau associatif. Ensuite, elle met les données en mémoire (buffer) sans rien envoyer au système. Elle intègre le template qui se construit en intégrant les données en mémoire, puis retourne la page HTML ainsi créée.

```

/**
 * render methode
 * @param $template
 * @param array $data_array
 * @param $viewPage
 * @return string
 */
public function render($template, $data_array = [], $viewPage){
    if(file_exists($template)){
        $datas = [$viewPage => $data_array];
        if(!empty($data_array)) extract($datas);
        ob_start();
        require_once $template;
        return ob_get_clean();
    }
}

```

Cas d'utilisation n°2 : la création d'un document

Ce cas d'utilisation a comme point de départ le moment où l'utilisateur clique sur « Créer un document »

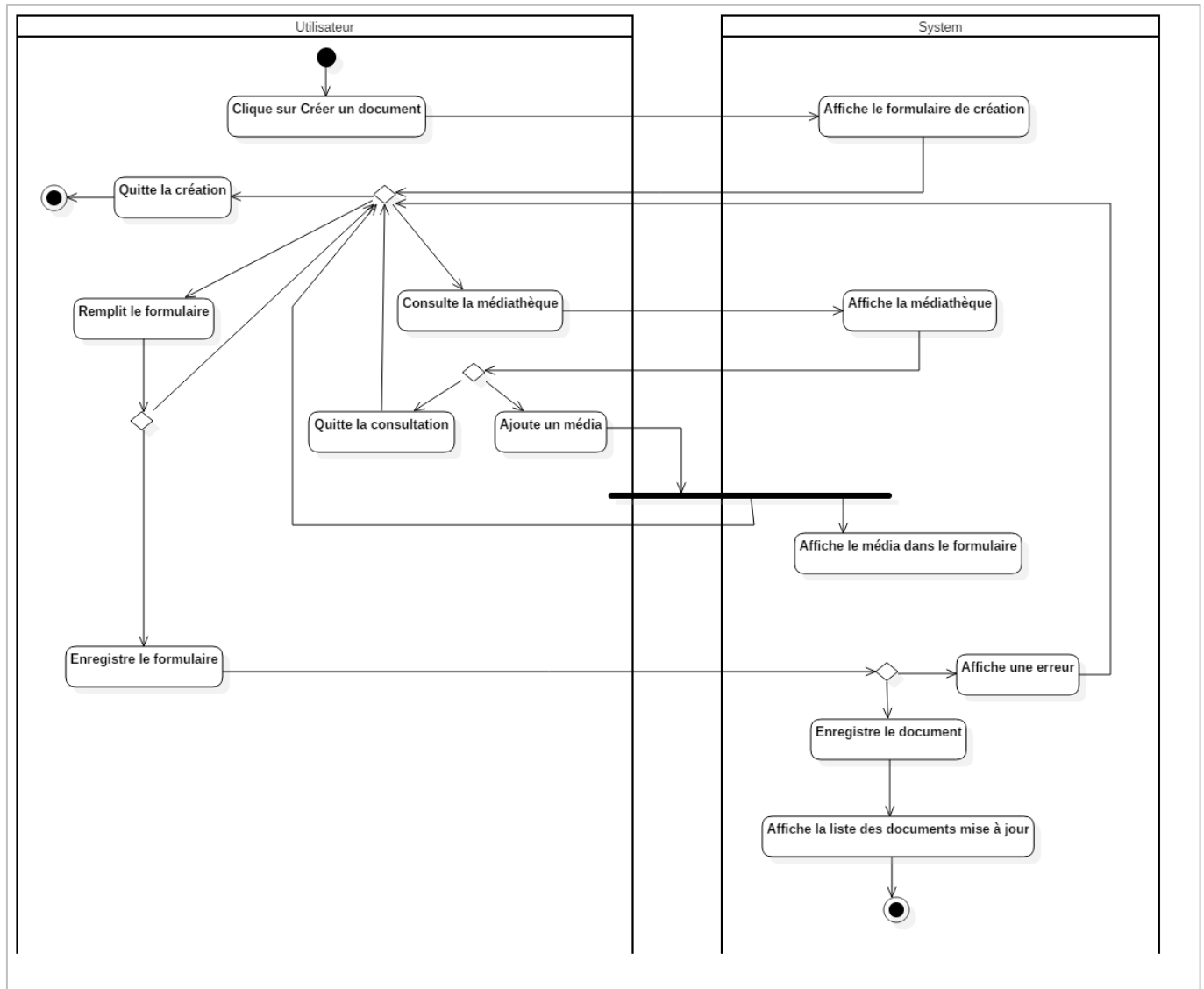


Figure 30 Scénario d'utilisation : création d'un document

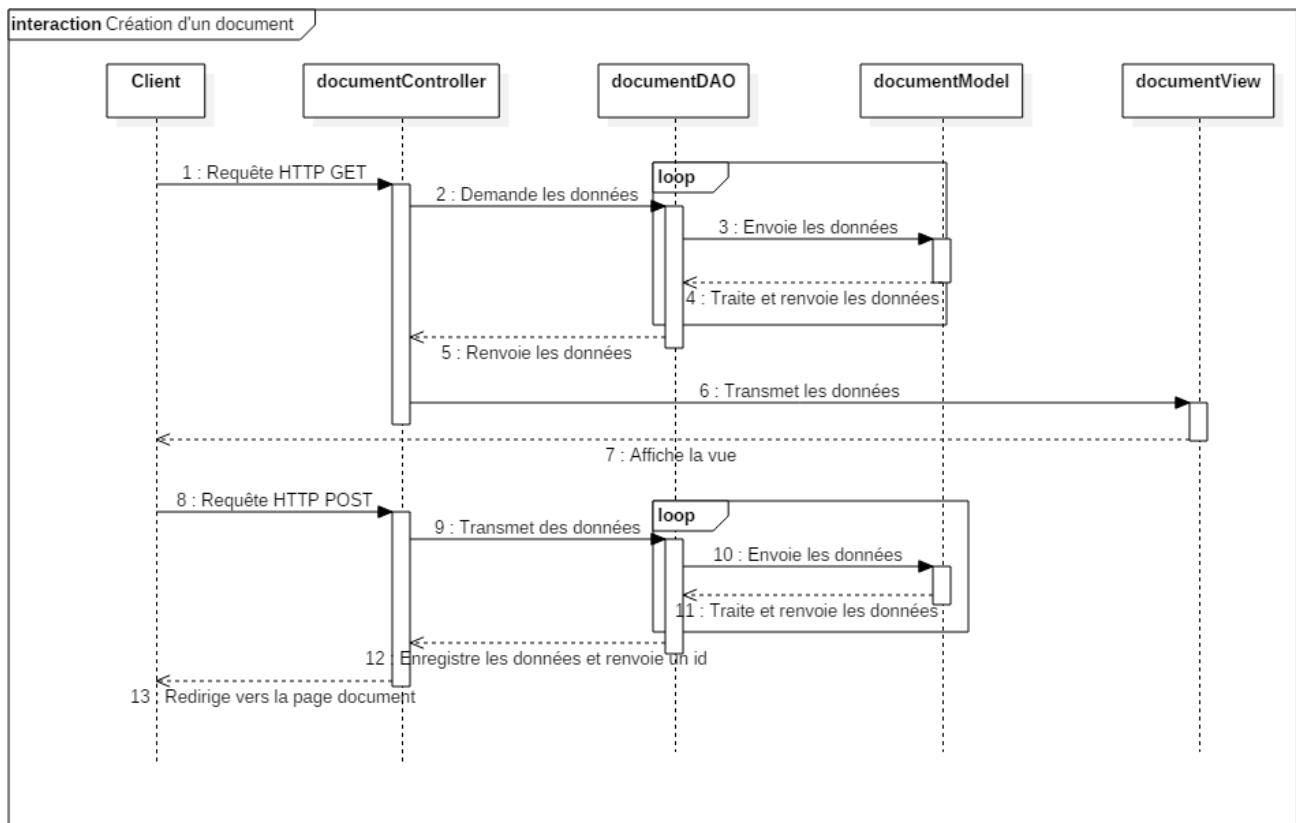


Figure 31 – Diagramme de séquence : création d'un document

On affiche d'abord la vue correspondant au formulaire de création du document, puis on va récupérer les données de ce formulaire en POST et les enregistrer en base de données.

L'url pour la création d'un document correspond à celle-ci : www.moduloplus.fr/document/new

La route correspondant à cette url est créée, route qui instancie le contrôleur et appelle la méthode dont on a besoin.

```

$app->get('/document/new(/cat-:cat_id)(/subcat-:subcat_id)(/doc-:doc_id)(/p-:param)',
    function ($cat_id = null, $subcat_id = null, $doc_id = null, $param = null){
        $controller = new
        \Modules\Document\Controllers\DocumentController($cat_id, $subcat_id);
        $controller->setDocumentForm($doc_id, $param);
    });
  
```

Ici, la route concerne les URL commençant par '/document/new'. Le paramètre entre parenthèses est optionnel : s'il a une valeur, alors ce sera de l'édition de document, pas de la création.

Le routeur va instancier le contrôleur `documentController()`, puis appeler la méthode `setDocumentForm()`. Cette méthode va simplement récupérer les données concernant les catégories et sous-catégories, afin de les afficher dans le formulaire.

Une fois le formulaire rempli, l'utilisateur va envoyer les données pour qu'elles soient enregistrées en base de données. La route correspondant à cette action est celle-ci :

```
$app->post('/document/new/post(/cat-:cat_id)(/subcat-:subcat_id)(/doc-:doc_id)(/p-:param)',
    function ($cat_id = null, $subcat_id = null, $doc_id = null,
    $param = null) use ($app){
        $req = $app->request(); //get the POST datas
        $controller = new
        \Modules\Document\Controllers\DocumentController($cat_id, $subcat_id);
        $result = $controller->postDocumentForm($req->post(), $doc_id,
        $param);
        if($result > 0) $app->redirect(FO_URL.'/documents/cat-
        '.$cat_id.'/subcat-.'.$subcat_id);
    });
```

Ici, la méthode devient `post()`, puisque la requête HTTP est un POST. Le routeur va récupérer les données envoyées avec la méthode `request()`, puis instancier le contrôleur `documentFormController()`.

Il va appeler la méthode `postDocumentForm()`, en lui passant les données en paramètres.

```
/**
 * @param $datas
 * @param null $id
 * @param null $duplicate
 * @return array|int|string
 */
public function postDocumentForm($datas, $id = null, $param = null){
    $datas['flag_home'] = isset($datas['flag_home']) ? 1 : 0;
    foreach ($datas as $k => $data){
        $datas[$k] = htmlspecialchars(trim($data));
    }
    if($id == null || $param === 'duplicate') $id_doc = $this->_DB-
    >insertDocument($datas);
    else $id_doc = $this->_DB->updateDocument($datas);
    return $id_doc;
}
```

Le contrôleur va faire appel au DAO, avec la méthode `insertDocument()`.

Contrairement à l’affichage des documents, on ne va pas faire l’appel à la base de données avant l’envoi dans le modèle, mais le contraire. Le modèle va vérifier que les données sont correctes avant de les envoyer en base de données.

```
/**
 * @param $datas
 * @return array|int|string
 */
public function insertDocument($datas){
    try {
        $this->callDocumentModel($datas);
        $datas = $this->toArray($datas);
        $datas['id'] = '';
        $query = 'INSERT INTO docs_documents (id, title, subtitle,
text1, text2, cat_id, subcat_id, link, text_link, flag_home,
creation_date, last_update)
VALUES (:id, :title, :subtitle, :text1, :text2, :cat_id,
:subcat_id, :link, :text_link, :flag_home, NOW(), NULL)';
        return $this->query($query, $datas);
    } catch(\InvalidArgumentException $e) {
        echo $e->getMessage();
    }
}
```

Le traitement des données

Le DAO va appeler la méthode `callDocumentModel()`, avec les données en paramètres. Pour l’appeler, il va utiliser un « try... catch », qui permet de gérer les erreurs avec la classe PHP Exception.

En cas d’erreur, PHP retourne une erreur fatale avec un message : le processus en cours s’arrête (la vue ne s’affiche pas, les données ne sont pas transmises à la base etc).

Le setter va faire les vérifications, comme par exemple sur le type de données et la longueur des chaînes. En cas d’erreur, il va renvoyer une exception avec un message que le DAO va capturer et afficher.

```
/**
 * @param $title
 */
public function setTitle($title){
    if(is_string($title) && strlen($title) > 0) {
        $this->_title = $title;
    } else {
        throw new \InvalidArgumentException("Le titre doit être du
texte");
    }
}
```

Si tout va bien, le DAO va exécuter la requête d'insertion des données dans la base de données, et retourner l'id du dernier élément inséré au contrôleur.

Le contrôleur va transmettre l'id au routeur, qui va rediriger le navigateur vers la vue du détail du dernier document créé.

6. Sécurité

6.1. Tests

Les tests unitaires

Le test unitaire est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme.

Les tests unitaires sont réalisés avec PHPUnit, un framework spécialisé pour PHP permettant d'automatiser les tests. Les objectifs sont multiples : tester le comportement des contrôleurs et des modèles, vérifier l'intégrité des formulaires, des données envoyées etc.

Nous allons prendre l'exemple de la création d'une catégorie, et du scénario qui permet de vérifier que le modèle n'accepte pas n'importe quelle donnée du formulaire.

Première étape : créer une classe `TestCategoryModel`, qui est descend de `PHPUnit`. Dans cette classe, on crée une méthode `setUp()`, qui va instancier la classe `CategoryModel`.

```
class TESTCategoryModel extends PHPUnit_Framework_TestCase
{
    /**
     * @var \Modules\Category\Models\categoryModel
     */
    public $objects;

    public function setUp(){
        $this->objects = new \Modules\Category\Models\categoryModel();
    }
}
```

On va créer un tableau de données, qui seront envoyées au modèle et serviront de données test. Dans ces données, il y aura des chiffres (int), et des suites de caractères (string), et une valeur « nulle ».

```

/**
 * @return array
 */
public function datasModel(){
    return [
        'is int' => [2],
        'is string' => ['Lorem Ipsum'],
        'is null' => [null]
    ];
}

```

On crée ensuite la première méthode, qui va vérifier les valeurs pour le nom de la catégorie. On va d'abord créer un objet `setCatName()`, qui prendra les données en paramètres. On va ensuite récupérer la valeur de `getCatName()`, et vérifier que la valeur correspond aux données envoyées.

Le commentaire « `@dataProvider datasModel` » au début de la méthode indique que les données proviennent du tableau de données créé plus haut. Chaque donnée va être envoyée dans la méthode pour être testée.

```

/**
 * @dataProvider datasModel
 */
public function testCategoryCatNameVal($datas){
    $this->objects->setCat_Name($datas);
    $model_return = $this->objects->getCat_Name();
    $this->assertEquals($datas, $model_return);
}

```

Dans un terminal, on lance la ligne de commande suivante pour lancer le test :

```
phpunit --bootstrap bootstrapPhpUnit.php modules\category\models\TESTcategoryModel.php
```

“--bootstrap” est une option de la commande PHPUnit. Elle permet de spécifier un fichier de configuration à PHPUnit. Ce fichier peut contenir un autoloader de classe par exemple.

Le terminal nous renvoie ceci :

```

Time: 0 seconds, Memory: 2.25Mb

There were 2 errors:

1) TESTCategoryModel::testCategoryCatNameVal with data set "is int"
(2)
InvalidArgumentException: Le nom de la catégorie doit être du texte

C:\xampp\htdocs\projets\zeproject\app\modules\category\models\category
Model.php:60
C:\xampp\htdocs\projets\zeproject\app\tests\modules\category\models\TE
STcategoryModel.php:27

2) TESTCategoryModel::testCategoryCatNameVal with data set "is null"
(NULL)
InvalidArgumentException: Le nom de la catégorie doit être du texte

C:\xampp\htdocs\projets\zeproject\app\modules\category\models\category
Model.php:60
C:\xampp\htdocs\projets\zeproject\app\tests\modules\category\models\TE
STcategoryModel.php:27

←[37;41m←[2KFAILURES!
←[0m←[37;41m←[2KTests: 3, Assertions: 12, Errors: 2.

```

Le test a retourné deux erreurs, avec les messages transmis par le modèle. Les erreurs étaient sur les données int et nulle : le nom d'une catégorie ne doit pas être un chiffre et ne doit pas être nul. La dernière assertion, avec la suite de caractères, est correcte.

6.2. Sécurité

Les risques de failles de sécurité sont particulièrement importants lors des connexions à la base de données, et dans les informations envoyées dans les formulaires.

Les injections SQL

L'injection SQL est une méthode d'attaque qui consiste à modifier une requête SQL en lui injectant des morceaux de code non filtrés, généralement par le biais d'un formulaire. Le risque est de voir sa base de données corrompue (voire totalement supprimée).

L'utilisation des requêtes préparées permettent de supprimer les injections SQL.

La faille XSS

La faille XSS (pour cross-site scripting) est une technique qui consiste à injecter du code HTML contenant un script dans les pages. Cette faille peut être dangereuse pour les cookies contenant les mots de passe, par exemple.

Pour l'éviter, il suffit d'employer la fonction *htmlspecialchars* sur tous les textes envoyés depuis un formulaire. On peut également utiliser la fonction *trim*, qui supprime les espaces (ou d'autres caractères) en début et fin de chaîne.

```
foreach ($datas as $k => $data){
    $datas[$k] = htmlspecialchars(trim($data));
}
```

Avec cette fonction, le tableau suivant :

```
array (size=4)
'name' => string ' Documentation technique & " ' (length=31)
'id_parent' => string '2' (length=1)
'icon' => string 'book " ' (length=7)
'id' => string '2' (length=1)
```

sera retourné dans la base comme ceci :

```
array (size=4)
'name' => string 'Documentation technique & &quot;&quot; ' (length=43)
'id_parent' => string '2' (length=1)
'icon' => string 'book &quot; ' (length=11)
'id' => string '2' (length=1)
```

La gestion des erreurs en PHP

Pour gérer les erreurs, nous utiliserons la classe PHP Exception qui permet de gérer les erreurs des scripts PHP.

La gestion des exceptions se fait dans le DAO et le modèle. Le modèle vérifie les données et les renvoie au DAO sous la forme d'un tableau d'objet.

Si le type de données ne correspond pas à celui demandé, le modèle renvoie (throw) un objet Exception qui contient un message d'erreur.

```

/**
 * @return mixed
 */
public function getId(){
    return $this->id;
}

/**
 * @param mixed $id
 */
public function setId($id){
    $id = (int) $id;
    if(is_int($id) && $id > 0) {
        $this->id = $id;
    } else {
        throw new \InvalidArgumentException("L'id de la catégorie doit
etre un nombre");
    }
}

```

Le DAO récupère (“catch”) le message d’erreur et retourne une erreur fatale permettant de stopper la procédure en cours.

```

try {
    $model = $this->callCategoryModel($sql);
    return $model;
} catch(\InvalidArgumentException $e) {
    return $e->getMessage();
}

```

La gestion des erreurs dans les formulaires

Enfin, pour sécuriser au mieux les formulaires dès que le client tape son texte, on utilise Javascript pour afficher des messages d’erreurs dans les champs vides qui ne devraient pas l’être (avec l’attribut HTML *required*), ou pour imposer un nombre de caractères minimum et maximum.

Par exemple, dans le formulaire de création d’un document, le choix d’une catégorie est obligatoire. On va donc ajouter l’attribut “required” au formulaire, ce qui empêchera la validation du formulaire tant que le champ sera vide :

```

<label for="cat_id">Catégorie</label>
  <select name="cat_id" id="cat_id" class="form-control" required>
    [...]
  </select>
</label>

```


La gestion de l’affichage des messages d’erreur pour les champs “required” est à personnaliser. Ici, nous utilisons Bootstrap, ce qui permettra d’avoir un affichage comme ci-dessous :



The image shows a Bootstrap form layout. At the top is a dropdown menu with a downward arrow. Below it, a light blue error message box contains the text "Veuillez sélectionner un élément de la liste." (Please select an item from the list). Underneath the error message are two text input fields. The first field is labeled "Titre" (Title) and the second field is labeled "Sous-titre" (Subtitle). Both fields are empty.

Figure 32 – Exemple d’erreur sur un formulaire

7. Gestion du projet

7.1. Livrables

Le premier livrable sera le package de Gestion des catégories. Les utilisateurs devront pouvoir créer, éditer et supprimer des catégories.

Le deuxième livrable sera le package de Gestion des documents. Les utilisateurs devront pouvoir créer, éditer et supprimer des documents, et affilier des documents à des catégories.

Le troisième livrable sera le package de Gestion des médias. Les utilisateurs devront pouvoir créer, éditer et supprimer des médias, et affilier des médias à des documents. Ils pourront avoir accès à une médiathèque.

Le quatrième et dernier livrable sera la mise en place des filtres sur les pages de liste de documents, avec notamment un filtre de recherche, et la possibilité de modifier l'ordre d'apparition des catégories.

La date de mise en production a été fixée au 1^{er} juillet.

7.2. Planning

WBS (Work Breakdown Structure)

Une structure WBS (Work Breakdown Structure) est une structure qui permet de découper le projet en sections à gérer. Sur ce projet, les sections correspondent aux différents livrables, avec les tâches qui leurs sont associées.

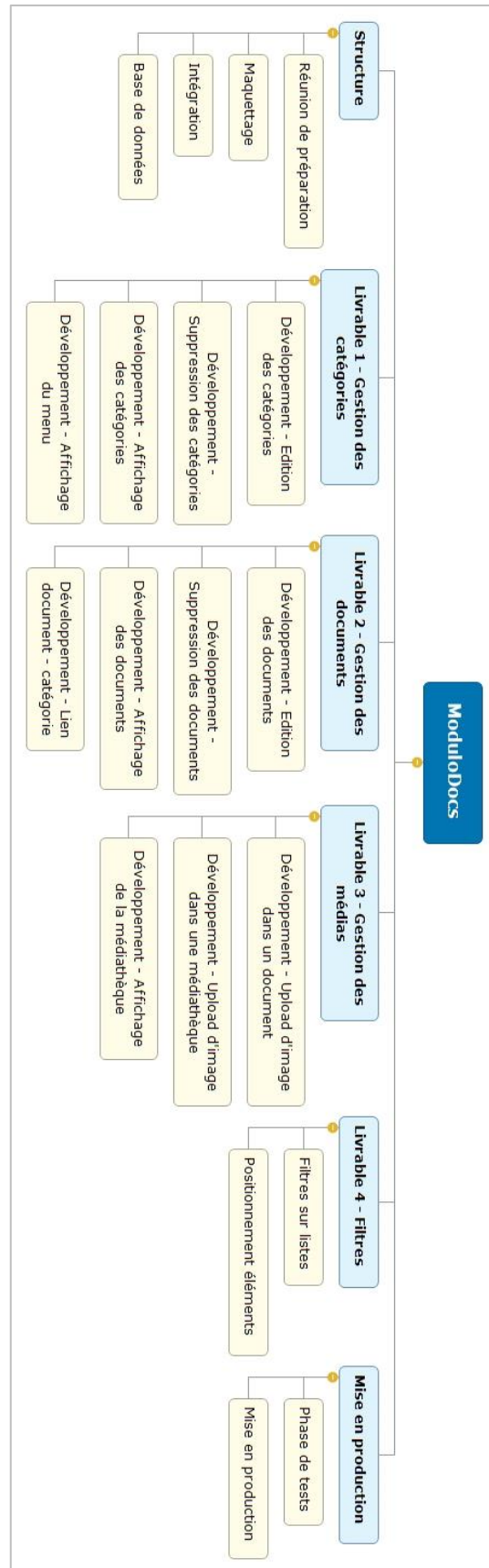


Figure 33 - WBS

Pert

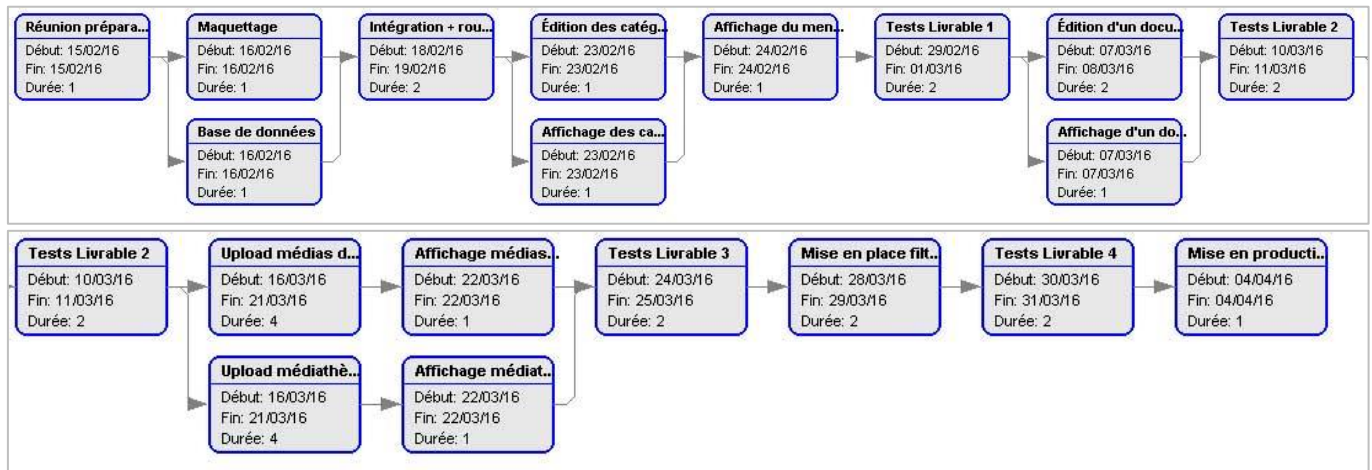


Figure 34 – Diagramme de PERT

Certaines tâches peuvent être faites en parallèle, comme le montre le diagramme de PERT. Un temps de latence est prévu à la fin de chaque livrable, pour les retards éventuels.

Dans les faits, le projet étant développé par une seule personne, les tâches se sont faites en continu.

Gantt

Le diagramme de Gantt est basé sur un planning en continu.

Dans les faits, le développement du projet se fera en fonction des autres projets de l'entreprise, et du temps disponible en-dehors de ces projets. Il ne faut donc pas tenir compte des dates. Le temps prévu, en revanche, a été calculé de la façon la plus réaliste possible.

La durée totale du projet est estimée à 30 jours de développement.

Les diagrammes ont été réalisés sous GanttProject.

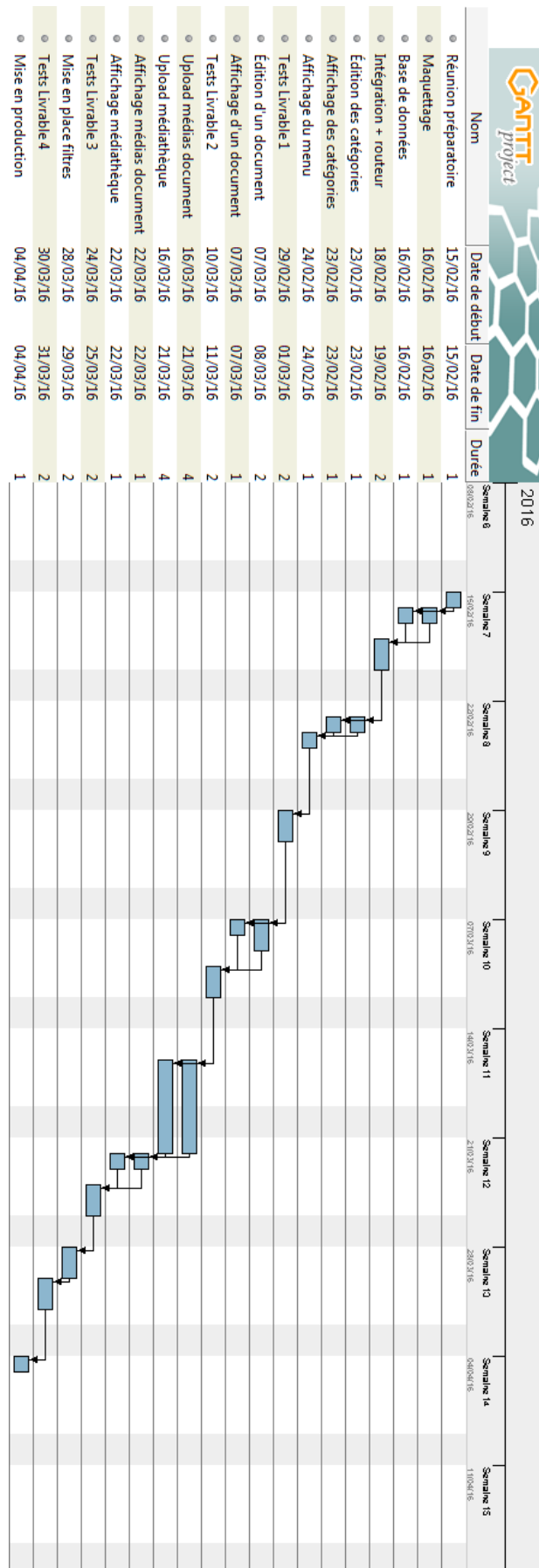


Figure 35 – Diagramme de Gantt

7.3. Déploiement

Les livrables sont développés et testés sur les machines en local, et envoyés sur BitBucket via GIT. Avant le passage en pré-production, on crée la base de données à partir du script de la base de données locale.

Chaque livrable, après vérifications et tests en local, est envoyé en préproduction par FTP.

Une fois que tous les livrables sont prêts et que les tests fonctionnels et unitaires ont été effectués, on crée une base de données en production à partir du script de la base de données en préproduction. Enfin, on envoie les fichiers en production par FTP.

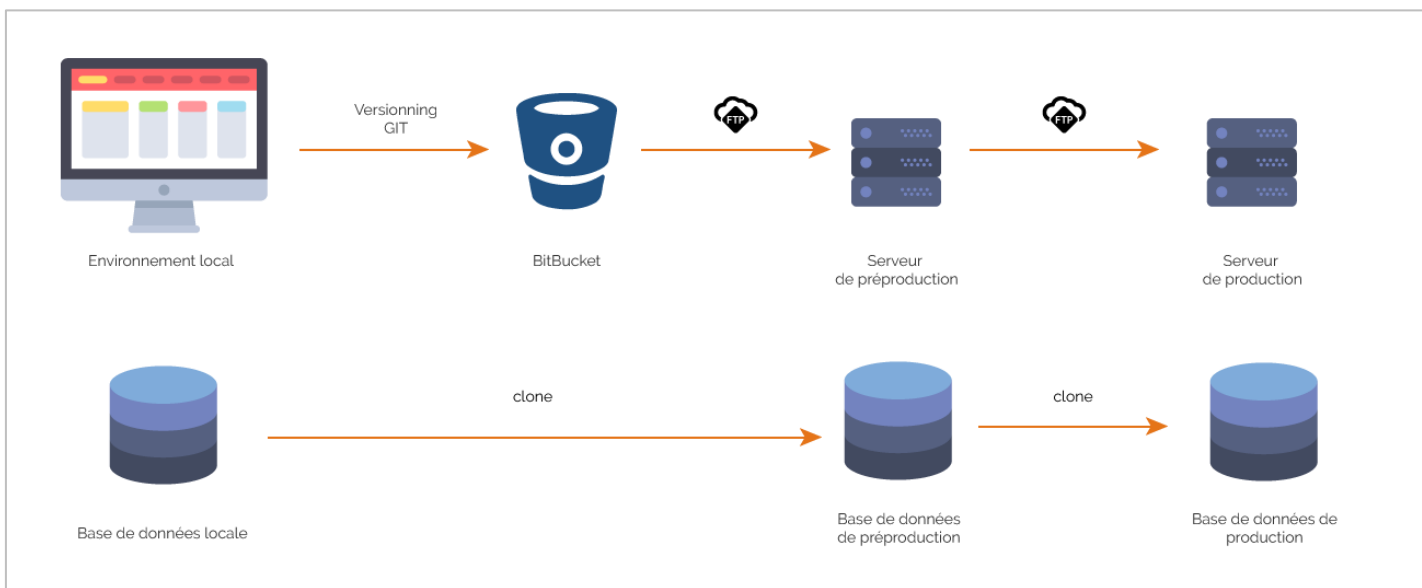


Figure 36 - Diagramme de déploiement

8. Conclusion

8.1. Bilan

La mise en place du projet et l'analyse des besoins était intéressante : elle a permis à l'entreprise de faire un point sur les ressources et documentations dont ils disposaient, et sur ce qu'ils souhaitaient en faire.

En pratique, l'espace documentaire commence à se remplir, surtout avec de la documentation technique. Il sert également à noter les compte-rendus des réunions-clients.

Côté technique, il a permis de mettre en place et de tester des technologies que l'entreprise n'utilisait pas : SASS, un routeur, un autoloader...

Comparaison planning prévu / planning réel

Nom	Date de début	Date de fin	Durée		Nom	Date de début	Date de fin	Durée
• Réunion préparatoire	15/02/16	15/02/16	1		• Réunion préparatoire	15/02/16	15/02/16	1
• Maquettage	16/02/16	16/02/16	1		• Base de données	16/02/16	16/02/16	1
• Base de données	16/02/16	16/02/16	1		• Maquettage	17/02/16	17/02/16	1
• Intégration + routeur	18/02/16	19/02/16	2		• Intégration + routeur	18/02/16	19/02/16	2
• Édition des catégories	23/02/16	23/02/16	1		• Édition des catégories	29/03/16	30/03/16	2
• Affichage des catégories	23/02/16	23/02/16	1		• Affichage des catégories	31/03/16	31/03/16	1
• Affichage du menu	24/02/16	24/02/16	1		• Affichage du menu	18/04/16	18/04/16	1
• Tests Livrable 1	29/02/16	01/03/16	2		• Affichage d'un document	09/06/16	09/06/16	1
• Édition d'un document	07/03/16	08/03/16	2		• Édition d'un document	01/07/16	05/07/16	3
• Affichage d'un document	07/03/16	07/03/16	1		• Upload médiathèque	16/08/16	18/08/16	3
• Tests Livrable 2	10/03/16	11/03/16	2		• Affichage médiathèque	19/08/16	19/08/16	1
• Upload médias document	16/03/16	21/03/16	4		• Upload médias document	22/08/16	22/08/16	1
• Upload médiathèque	16/03/16	21/03/16	4		• Affichage médias document	23/08/16	23/08/16	1
• Affichage médias document	22/03/16	22/03/16	1		• Tests unitaires	24/08/16	24/08/16	1
• Affichage médiathèque	22/03/16	22/03/16	1		• Mise en production	25/08/16	25/08/16	1
• Tests Livrable 3	24/03/16	25/03/16	2					
• Mise en place filtres	28/03/16	29/03/16	2					
• Tests Livrable 4	30/03/16	31/03/16	2					
• Mise en production	04/04/16	04/04/16	1					

Figure 37 – Planning prévu, planning réel

Afin de répondre aux nouvelles demandes des clients avant la période estivale, le projet a été mis en attente pendant les mois de mai et juin pour finaliser les autres productions.

Pour pouvoir faire la mise en production avant la rentrée, il a été décidé de ne pas faire les tests unitaires à chaque livrable et de ne pas mettre les filtres de tri en place.

L'estimation de la durée de développement était plutôt juste : l'édition des documents et catégories a été un peu plus longue que prévu, mais l'upload des médias dans les documents a, lui, été beaucoup plus rapide.

Au total, en réduisant les tests et en enlevant le dernier livrable, le développement a pris 21 jours.

8.2. Améliorations

- Côté sécurité, l'espace documentaire pourra être amélioré avec un https.
- Il sera intéressant d'utiliser Composer pour gérer les dépendances (Slim, autoloader, Bootstrap).
- Les options de tri et de filtres seront à développer dans le futur.
- L'espace médiathèque peut être amélioré et développé : édition des médias, ajout d'un média dans la médiathèque via les documents...

8.3. Comparatif maquette / site final

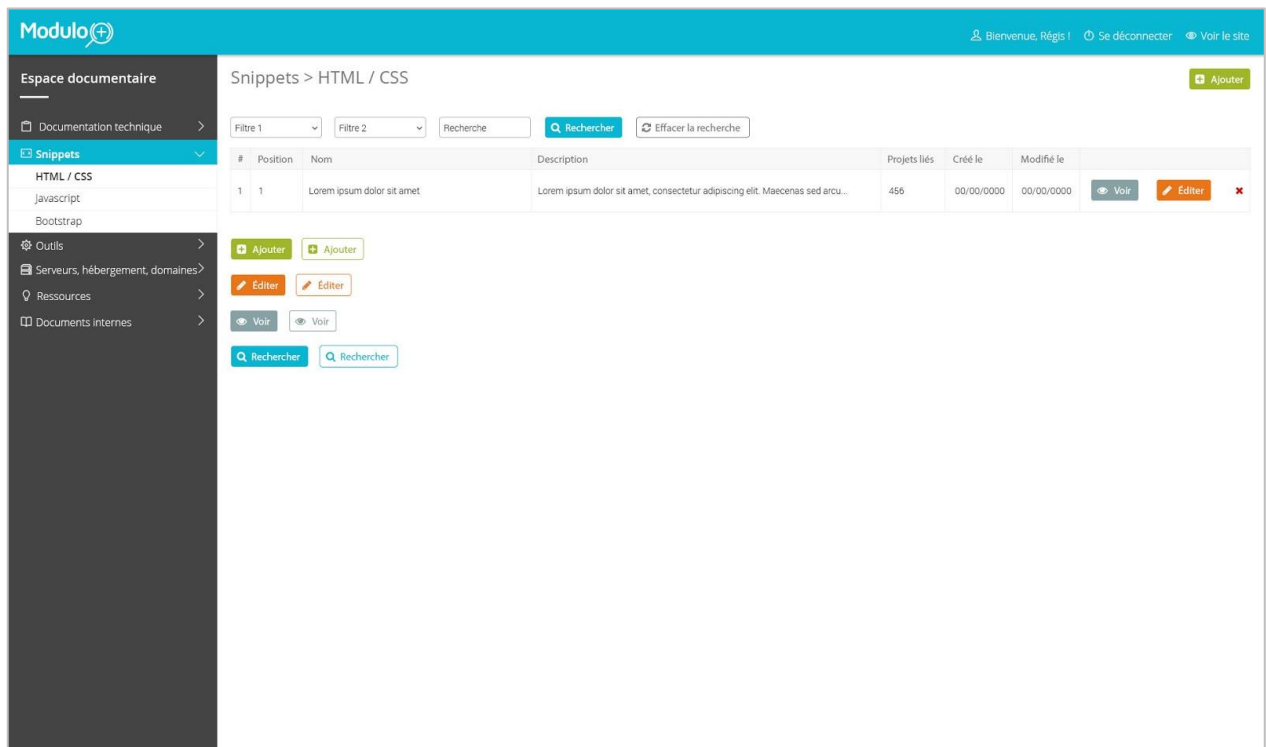


Figure 38 - Maquette choisie par Modulo+

Espace documentaire ModuloPlus Ajouter un document

Les documents à la une

#	Titre	Sous-titre	Univers	Rubrique	Créé le	Modifié le	
5	Test1	super test test	Documentation technique	Projets	18/04/2016	22/07/2016	Détail Modifier

Les dernières publications

#	Titre	Sous-titre	Univers	Rubrique	Créé le	Modifié le	A la une	
21	TEST submit	Re test	Documentation technique	Projets	22/07/2016	22/07/2016		Détail Modifier
20	TEST submit	Re test	Documentation technique	Projets	22/07/2016	22/07/2016		Détail Modifier
19	TEST submit	Re test	Documentation technique	Projets	01/07/2016	01/07/2016		Détail Modifier
11	Hello, this is the test n°23' #@!	Aliquam blandit sed sapien eu fringilla	Documentation technique	Projets	25/04/2016	09/06/2016		Détail Modifier
9	Titre du fichier	Re test	Documentation technique	Projets	18/04/2016	30/06/2016		Détail Modifier
8	Test3	sam sous titre	Snippets	HTML / CSS	18/04/2016	09/06/2016		Détail Modifier
7	Test	sam sous titre	Documentation technique	Interlib	18/04/2016	09/06/2016		Détail Modifier
5	Test1	super test test	Documentation technique	Projets	18/04/2016	22/07/2016	✓	Détail Modifier

Figure 39 - Capture d'écran du site final

La maquette a été respectée pour le rendu final, à quelques petites différences près : le bouton « Modifier » est passé en bleu, de peur que la couleur orange soit associée à la couleur rouge du bouton supprimer.

9. Annexes

9.1. Glossaire

Intranet : réseau informatique utilisé à l'intérieur d'une entreprise

Front-office : partie d'un système informatique accessible aux utilisateurs finaux ou aux clients

Back-office : la partie d'un système informatique qui n'est pas accessible aux utilisateurs finaux ou aux clients

Framework : ensemble de classes d'objet, utilisables pour créer des applications informatiques.

Responsive design : un design qui s'adapte à tous les appareils (ordinateurs, portables, tablettes...)

Design pattern (ou patrons de conception) : arrangement récurrent de rôles et d'actions joués par des modules d'un logiciel.

FTP (File Transfer Protocol) : protocole de communication destiné à l'échange informatique de fichiers sur un réseau

La gestion de versions (*version control* ou *revision control*) consiste à maintenir l'ensemble des versions d'un ou plusieurs fichiers.

GIT : logiciel de gestion de versions décentralisé

9.2. Table des illustrations

FIGURE 1 – STRUCTURE DU SITE WWW.MODULO+.COM	6
FIGURE 2 - SCHEMA DES CAS D'UTILISATION	8
FIGURE 3 - STYLETILE	10
FIGURE 4 - AFFICHAGE DE LA PAGE D'ACCUEIL	11
FIGURE 5 - AFFICHAGE D'UNE CATEGORIE.....	12
FIGURE 6 - AFFICHAGE D'UNE SOUS-CATEGORIE.....	12
FIGURE 7 - AFFICHAGE DU DETAIL D'UN DOCUMENT	13
FIGURE 8 – INTRANET DE MODULO+	14
FIGURE 9 - MAQUETTE N°1	14
FIGURE 10 - MAQUETTE N°2	15
FIGURE 11 - MAQUETTE N°3	15
FIGURE 12 - ENVIRONNEMENT LOCAL.....	16
FIGURE 13 - ENVIRONNEMENTS DE PREPRODUCTION ET DE PRODUCTION.....	17
FIGURE 14 – ARBORESCENCE DU SITE	18
FIGURE 15 – ARBORESCENCE DES FICHIERS DE CONCEPTION	20
FIGURE 16 - SCHEMA DE LA BASE DE DONNEES	21
FIGURE 17 – MODELE CONCEPTUEL DES DONNEES.....	22
FIGURE 18 - DIAGRAMME DE CLASSES	25
FIGURE 19 – SCHEMA MVC.....	25
FIGURE 20 – CAS D'UTILISATION : AFFICHAGE DU DETAIL D'UN DOCUMENT	31
FIGURE 21 – DIAGRAMME DE SEQUENCE POUR L’AFFICHAGE D’UN DOCUMENT	32
FIGURE 22 – MVC : LA REQUETE HTTP	33
FIGURE 23 – MVC : L’APPEL A LA BASE DE DONNEES.....	35
FIGURE 24 – MVC : ENVOI DES DONNEES POUR LE TRAITEMENT	38
FIGURE 25 – MVC : TRAITEMENT DES DONNEES.....	42

FIGURE 26 – MVC : RENVOI DES DONNEES TRAITEES.....	43
FIGURE 27 – MVC : LA PREPARATION DE LA VUE	43
FIGURE 28 – MVC : LE RENDU HTML DE LA VUE	44
FIGURE 29 – INTEGRATION DE LA VUE.....	45
FIGURE 30 – CAS D’UTILISATION : CREATION D’UN DOCUMENT	47
FIGURE 31 – DIAGRAMME DE SEQUENCE : CREATION D’UN DOCUMENT	48
FIGURE 32 – EXEMPLE D’ERREUR SUR UN FORMULAIRE	57
FIGURE 33 - WBS	59
FIGURE 34 – DIAGRAMME DE PERT	60
FIGURE 35 – DIAGRAMME DE GANTT	61
FIGURE 36 - DIAGRAMME DE DEPLOIEMENT	62
FIGURE 37 – PLANNING PREVU, PLANNING REEL.....	63
FIGURE 38 - MAQUETTE CHOISIE PAR MODULO+	64
FIGURE 39 - CAPTURE D’ECRAN DU SITE FINAL	65