ENI Service - WordPress - PHP

Paul Schuhmacher

Durée : 28h

Novembre 2025

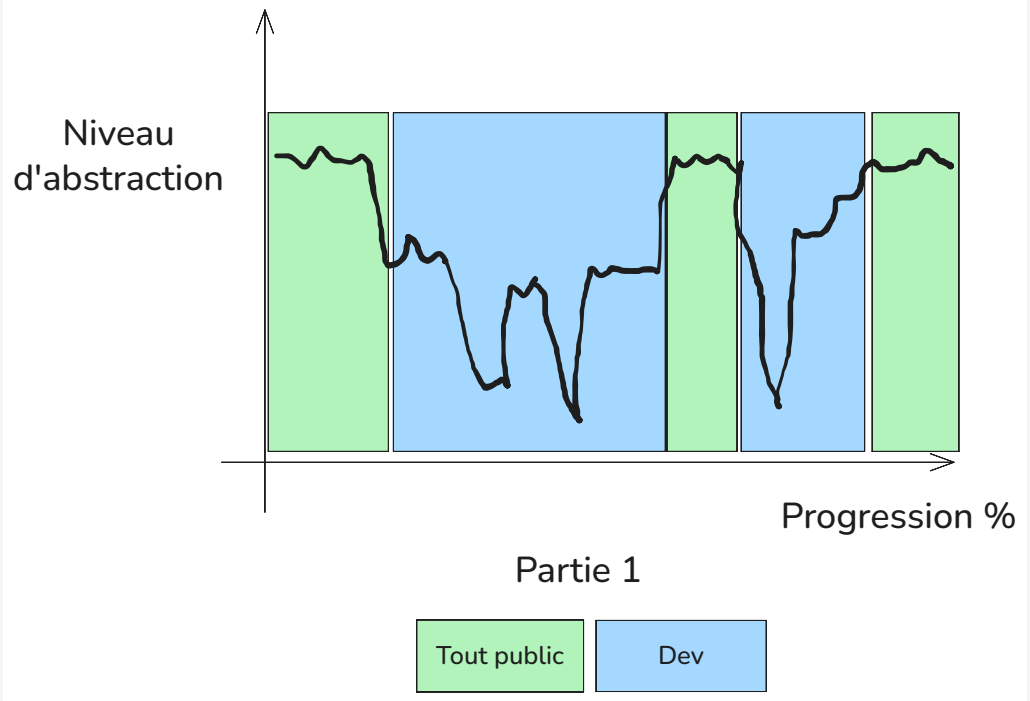
Partie 1 : Présentation, setup et fonctionnement général (*core*)

Partie 1 : Présentation, setup et fonctionnement général (*core*)

{.marp-bg-img}

Pour bien s'*installer* et commencer à se familiariser avec le framework.

- *Challenge* : s'adresser à deux audiences
- Cahier des charges : orienté *dev*



Questions préliminaires à l'audience

Pour adapter au mieux le reste de la formation :

- Sites sur lesquels vous travaillez ? (nature, contraintes, état actuel, évolutions futures ?)
- Les sites utilisent des thèmes classiques, FSE (entièrement basés sur des blocs et l'éditeur de site) ou **hybrides** (blocks pour le contenu uniquement) ?
- Quels problèmes rencontrés en particulier ? *Uses Cases* ? Contraintes ? Difficultés ? Organisation/processus éditorial ?
- Plugins actuellement utilisés ? Hébergements ? Abonnements ?
- Quelle méthode actuelle pour déployer ? Ouvert à d'autres méthodes (propositions) ?

WordPress, en bref

Historique

- A pour origine le logiciel b2/cafelog développé en 2001, par Michel Valdrighi. Logiciel de blog open-source avec une base de données;
- Reprise de b2/cafelog par Matthew Mullenweg (startup Automattic) et Mike Little, rejoints par Valdrighi, en 2003;
- **Lancement de WordPress Version 1.0 en janvier 2004** (21 ans d'existence !)

WordPress, en bref

Quelques actualités récentes

- **11 novembre 2025** : WordPress 6.9 Release Candidate 1
- **Septembre 2025** : Release WordPress 6.8.3
- **2024** : Matthew Mullenweg accusé par WP Engine d'extorsion et abus de marque après que Matthew Mullenweg ait dénigré WP Engine ("cancer" de Wordpress), leur ait bloqué l'accès à wordpress.org, forké et distribué leur plugin ACF (SCF), et accusé cette dernière de bénéfices économiques sans retour au logiciel open-source.
- **2023** : Migration des sites de la NASA sous WordPress;

Caractéristiques générales et évolution

- *Content-Management-System (CMS)* **open-source** : fournir un certain niveau d'automatisation pour gérer (gestion, livraison) efficacement du **contenu**, adapté à un **processus éditorial** ;
- Maintenu et développé par différents types de contributeur·ices;
- Plutôt orienté *final user* que *user (developer)*;
- Écrit en PHP, basé sur une **base de données relationnelle** ;
- Framework *MVC*, développement sur mesure basé sur des thèmes (*View*) et des plugins (*Model*);
- De nombreuses APIs : Settings API, REST API, Plugin API, Blocks API, Database, etc.;
- Depuis la V5 (2018) MAJ concentrées sur le Full Site Editing (FSE) avec l'éditeur de blocs Gutenberg (concurrence Wix, Webflow, etc.), partie CMS et ses APIs un peu délaissées...

Usages de WordPress

- **CMS** / Site web / Application web "classique";
- Service web (programmable) / web **API** / CMS *Headless* : WordPress peut exposer ses ressources *via* son API RESTful **native** personnalisable :

```
{domain}/wp-json/wp
{domain}/wp-json/wp/v2/posts
```

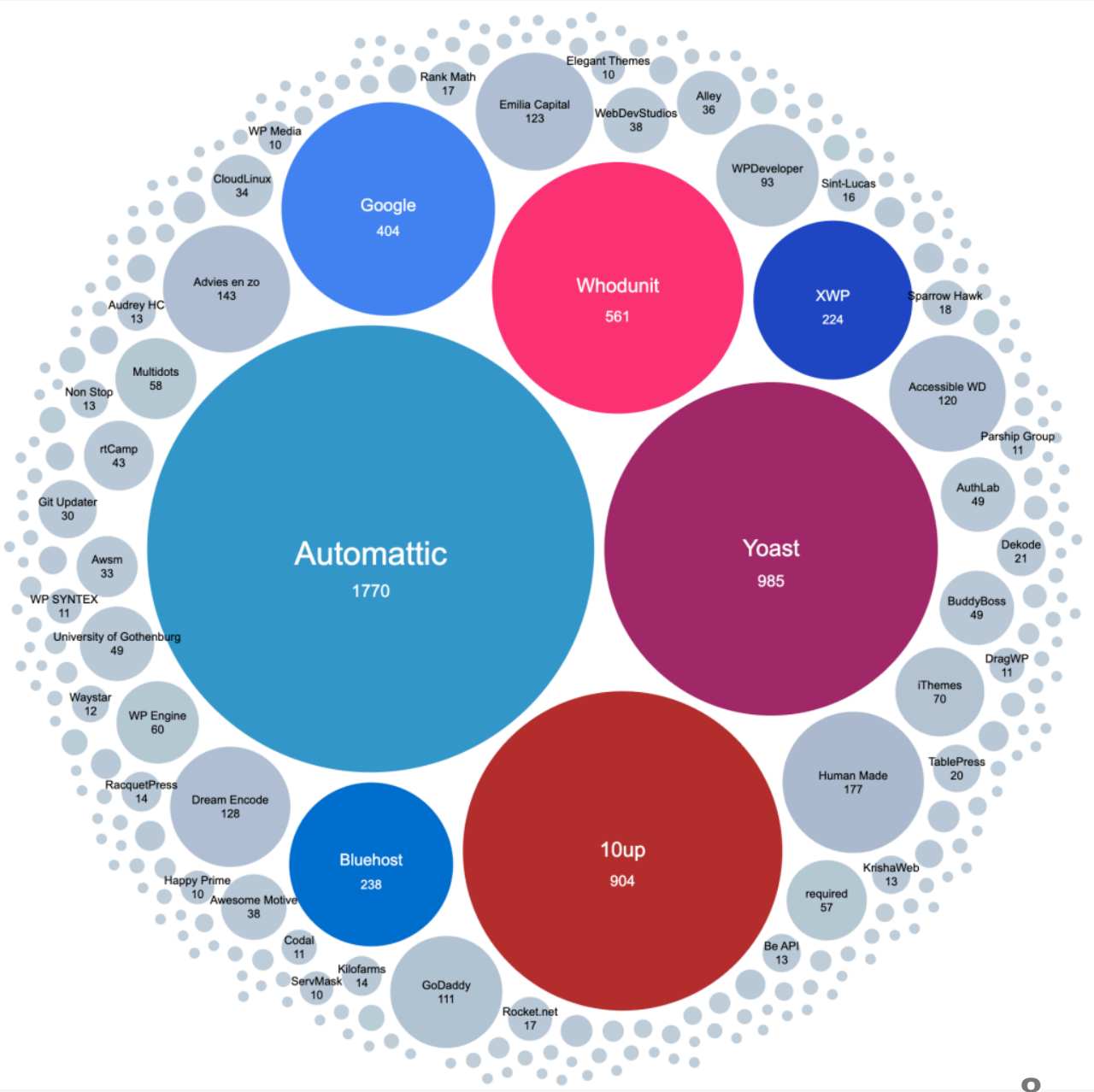
- **Multisites network** : administrer et gérer un *réseau* de sites, partage de contenu;
- **Engine**/Framework applicatif : peut être utilisé comme un framework PHP léger en se servant uniquement de fonctionnalités *core*.

Contributeur·ices au core (2023)

Nombre de commits par entreprise

{.marp-bg-img}

> Source

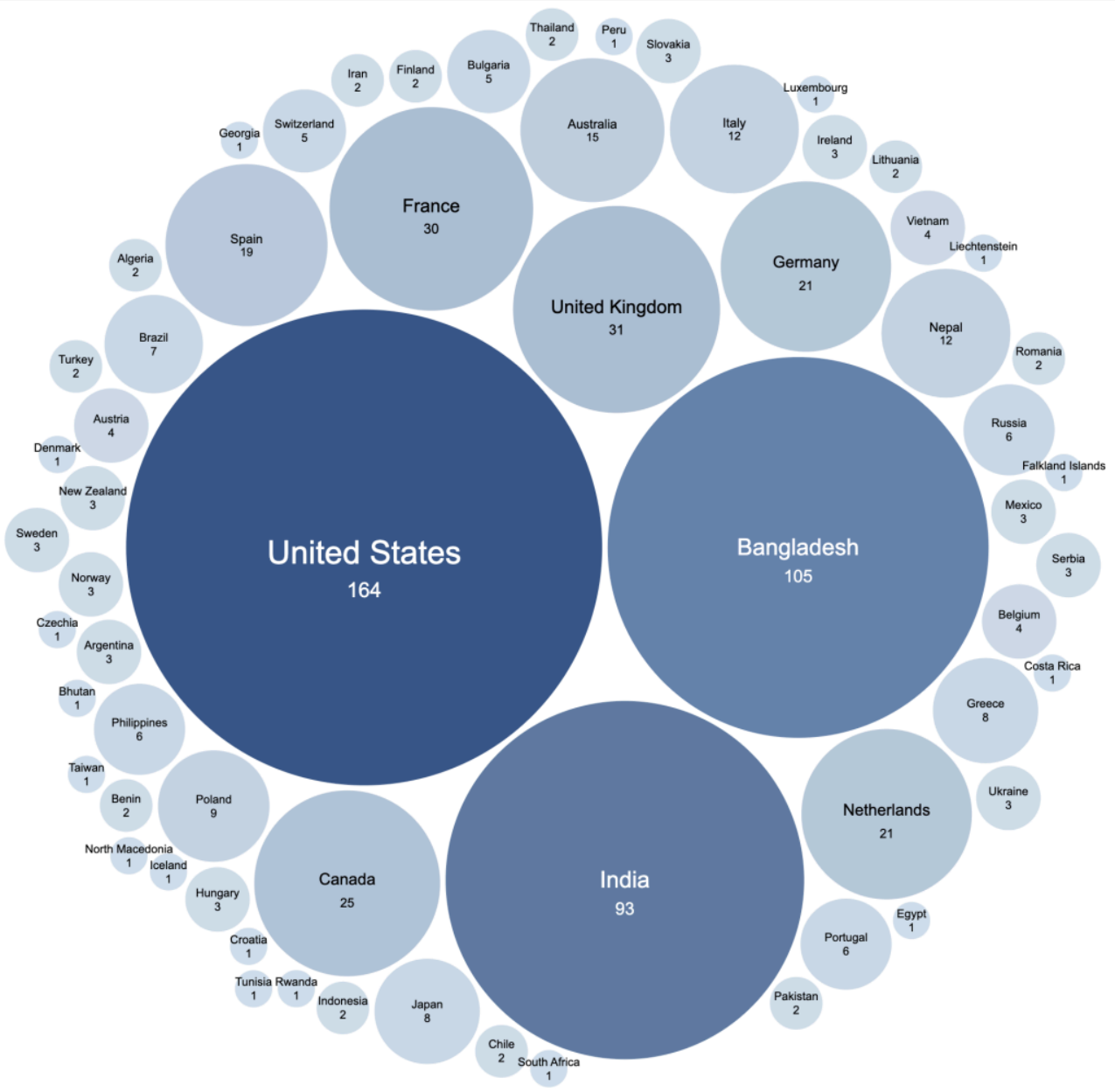


Contributeur·ices au core (2023)

Nombre de contributeur·ices par pays

{.marp-bg-img}

> Source



Releases, *roadmap* de WordPress

- **Version actuelle 6.8**
- Prochaines releases : 6.9 (prévue le 02/12/25), 7.0 (2026)
- Roadmap :
  - **Gutenberg, Gutenberg, Gutenberg !** :
    - Améliorer l'éditeur de contenu/de site, experience utilisateur (final user);
    - Customisation : **Site Editing, block patterns, block directory, block themes** (user)
  - **Collaboration** : intégrer l'écriture collaborative sur le contenu (impossible pour l'instant)
  - Intégrer le **support multilingue** au core (actuellement réalisé via plugins, multi-sites)

> Pas grand chose de nouveau sur les autres API... (Settings, REST, etc..)

> Accéder à la roadmap

Écosystème

Projet open-source, protégé par la WordPress Foundation (protège la marque WordPress).

wordpress.org

- Site de la communauté open-source WordPress, accès au code source;
- Documentation officielle/Codex;
- Héberge des themes/plugins **gratuits**. *Marketplace* officiel.

Vendors majeurs (services)

- Wordpress.com : société d'Automattic (Matthew Mullenweg), *Wordpress as a Service (WAAS?)*, WooCommerce, JetPack
- WP Engine (**Plugins ACF**, WP Migrate. Genesis Framework. Hébergement, etc.)
- Kinsta (Hébergement infogéré spécialisé WordPress, DevKinsta, expertise/blog);
- WPForms: form builder/handler;
- Yoast: plugin + admin SEO;

> WordPress (wordpress.org) != Wordpress.com (Automattic) != WP Engine !

Préparation de l'environnement de développement

Versions de PHP supportées

> À l'heure de la formation

- **WordPress core** vise toujours le support des nouvelles versions de PHP ;
- Aujourd'hui, PHP >= 7.3 ;
- Recommandé : **PHP8.3+**, **PHP7 minimum**.

> Attention : il est de la **responsabilité de chaque *vendor* (thème, plugin)** de maintenir la compatibilité de ses modules avec WordPress !

> Voir la compatibilité entre WordPress et PHP selon leurs versions.

SBGBDR (base de données) supportés

> À l'heure de la formation

- WordPress *core* utilise MySQL ou son *fork* MariaDB pour des raisons historiques. **Aucun autre SGBD n'est officiellement supporté** ;
- L'écosystème WordPress (plugins, migrations, sauvegardes) repose sur MySQL. **La majorité des extensions présupposent MySQL** (syntaxe requêtes SQL, moteur de table transactionnel InnoDB, charset `utf8mb4`, attribut `AUTO_INCREMENT`, etc.) ;
- Versions recommandées :
  - **MySQL : 8.0+** (*Legacy* : MySQL 5 n'est plus maintenue !)
  - MariaDB : 10.6+

> Possible d'utiliser d'autres systèmes (PostgreSQL, SQLite) mais demande une couche logicielle supplémentaire (plugins), peu d'intérêt si le choix ;

Installation de l'environnement de développement

Télécharger le *kit de développement* (dossier `kit-dev`) préparé pour la formation.

1. Cloner le dépôt;
2. **Suivre les instructions**.;
3. Annexe : Se préparer un environnement de développement complet sous Windows avec la WSL.

> Nous utiliserons Docker pour servir le projet en local.

Présentation de l'environnement de développement

- **WordPress + base de données MySQL conteneurisés** (*platform independant*) : Permet de travailler sereinement sur plusieurs projets en même temps avec des versions WordPress, PHP, MySQL potentiellement différentes sur différentes machines (à condition d'y installer Docker);
- **Services de développement** :
  - **Administration de base de données** Adminer (remplacer par phpMyAdmin ou autre au besoin),
  - **MailCatcher** (Mailhog) : pour debug l'envoi d'email,
  - **WP-CLI**, pour interagir avec WordPress en ligne de commande,
  - **Pipeline Compilation d'assets** avec Gulp, pour compiler les assets CSS, JS des thèmes,
- **Intégration Continue** :
  - Linter(PHP_CodeSniffer),
  - Analyse statique de code (phpStan),
  - Générateur de documentation de référence (phpDocumentor).

WordPress *Core* : Architecture

- WordPress contient des sources PHP, JS, CSS ;
- Chaque fichier a une *fonction précise* ;
- **Inspectons** le contenu de WordPress :

```
#A la racine du kit de développement, après installation
cd web
ls -C1 --group-directories-first
```

WordPress Core

```
wp-admin/
wp-content/
wp-includes/
index.php
license.txt
readme.html
wp-activate.php
wp-blog-header.php
wp-comments-post.php
wp-config-docker.php
wp-config.php
wp-config-sample.php
wp-cron.php
wp-links-opml.php
wp-load.php
wp-login.php
wp-mail.php
wp-settings.php
wp-signup.php
wp-trackback.php
xmlrpc.php
```

WordPress Core

Trois répertoires :

- `wp-admin` + `wp-includes` + plupart des fichiers à la racine : fichiers *core* (moteur WordPress). **Ne jamais les modifier**, "Don't hack the core!" (ces fichiers sont réécrits à chaque mise à jour de WordPress !) ;
- `wp-content` : **nos fichiers** (themes, plugins, media), **user space** ;

Les **contenus** HTML (*pages*, *posts*, *users*, etc.) et leurs métadonnées sont **stockés** dans la **base de données**.

Configuration de WordPress

```
wp-admin/
wp-content/
wp-includes/
index.php
license.txt
readme.html
wp-activate.php
wp-blog-header.php
wp-comments-post.php
wp-config-docker.php
.......................... => wp-config.php
wp-config-sample.php
wp-cron.php
wp-links-opml.php
wp-load.php
wp-login.php
wp-mail.php
wp-settings.php
wp-signup.php
wp-trackback.php
xmlrpc.php
```

Configuration de WordPress

- Le fichier `wp-config.php` est le **fichier de configuration le plus important** !
- Propre **à chaque installation** (chaque environnement) ;
- **À conserver secret !** Il contient :
  - les credentials d'**accès à la base de données**,
  - les **clefs et sels de sécurité pour signer** les cookies et tokens de session,
  - le **préfixe** des noms des tables WordPress *core*,
  - l'**activation/désactivation du mode debug**,
  - vos **configurations customs**,
  - etc;

> Le fichier `wp-config-sample.php` sert de base à la création du fichier `wp-config.php` lors de l'installation.

> Accéder à la documentation de `wp-config.php`

Authentification sous WordPress

- L'authentification native de WordPress se base sur des **cookies d'authentification signés** :
  - `wordpress_[hash]` : authentifie l'utilisateur·ice pour l’administration (wp-admin);
  - `wordpress_logged_in_[hash]` : authentifie l’utilisateur·ice sur la partie publique (frontend).
- Ces cookies ont la forme :
  - **clé** : hash
  - **valeur** : ID utilisateur | pass_frag | timestamp | token unique

Le hash est calculé avec les clés et sels définis dans `wp-config.php` (cf slide suivante).

> Il est également possible de s'authentifier auprès de la REST API via plusieurs méthodes (JWT, Basic Auth, etc.)

Clefs et sels

Les clefs et sels sont générés à l'installation et sont utilisés pour signer les cookies d'authentification et les nonces (tokens anti-CSRF pour les formulaires) :

```
//Clefs secrètes pour signer :
//- les cookies d'authentification admin
define( 'AUTH_KEY',         'put your unique phrase here' );
//- les cookies d'authentification admin en HTTPS
define( 'SECURE_AUTH_KEY',  'put your unique phrase here' );
//- les cookies d'authentification d'un utilisateur connecté à l'interface publique, sans accès à l’administration.
define( 'LOGGED_IN_KEY',    'put your unique phrase here' );
//- les nonces WordPress (tokens CSRF) utilisés pour valider les formulaires d'un user authentifié
define( 'NONCE_KEY',        'put your unique phrase here' );
//Sels (salts) associés à leurs clés respectives
define( 'AUTH_SALT',        'put your unique phrase here' );
define( 'SECURE_AUTH_SALT', 'put your unique phrase here' );
define( 'LOGGED_IN_SALT',   'put your unique phrase here' );
define( 'NONCE_SALT',       'put your unique phrase here' );
```

WordPress utilise HMAC pour **signer** les cookies et les nonces (`wp-includes/pluggable.php`) :

```
//Génération d'un cookie dans la fonction wp_generate_auth_cookie() basé sur un $token (valeur aléatoire) conservé en base
//clé
$key = wp_hash( $user->user_login . '|' . $pass_frag . '|' . $expiration . '|' . $token, $scheme );
//valeur
$hash = hash_hmac( 'sha256', $user->user_login . '|' . $expiration . '|' . $token, $key );
```

Configurer WordPress en *Debug mode*

Dans le fichier `wp-config.php`, configurer le mode debug pour avoir accès à toutes les informations d'erreur et diagnostics:

```
define( 'WP_ENVIRONMENT_TYPE', 'development' ); // Permet de définir le mode d'execution dans le code.
//Disposer de toutes les informations de debug
define( 'WP_DISABLE_FATAL_ERROR_HANDLER', true ); // Désactiver le gestionnaire d'erreur global > 5.2
define( 'WP_DEBUG', true );                       // Active le mode debug global
define( 'WP_DEBUG_DISPLAY', true );               // Affiche les erreurs à l'écran (activé si WP_DEBUG activé)
define( 'WP_DEBUG_LOG', true );                   // Log dans wp-content/debug.log
define( 'SAVEQUERIES', true );                    // Log SQL dans $wpdb->queries (debug DB)
define( 'SCRIPT_DEBUG', true );                   // Charge assets JS/CSS non minifiés du core

//Afficher la valeur des constantes
//print_r( get_defined_constants() );
```

> On reviendra sur la configuration en mode *production* plus tard dans la formation.

Configurer WordPress en *Debug mode*

Quelques configurations :

- **Tout afficher** :

```
define('WP_DEBUG', true); //erreurs sur la sortie (à l'écran)
define('WP_DEBUG_LOG', true); //erreurs dans le fichier de log wp-content/debug.log
```

- Afficher seulement dans le fichier de *log* (pas polluer la page en cours de développement) :

```
define('WP_DEBUG', true);
define('WP_DEBUG_LOG', true);
define('WP_DEBUG_DISPLAY', false); // pour éviter l'affichage sur la sortie (à l'écran)
```

Configurer WordPress en *Debug mode*

`WP_ENVIRONMENT_TYPE` est une constante introduite depuis WordPress 5.5 pour identifier le contexte d’exécution du site web.
Permet aux thèmes et aux plugins d'adapter leur comportement :

```
if ( wp_get_environment_type() === 'development' ) {
  //Faire quelque chose de spécifique en environnement de dev
}
```

> Doc wp_get_environment_type()

Configurer WordPress en *Debug mode*

Pour débuger les requêtes SQL :

```php
// Log toutes les requêtes vers la base dans un tableau PHP
define( 'SAVEQUERIES', true );
// Pour afficher le tableau de la requête
global $wpdb;
print_r( $wpdb->queries );
```

Configuration des logs dans WordPress

- **Toujours log** (erreurs PHP, *logs manuels* via la fonction `error_log`) **en environnement de développement et de production**.
- **Mesure de sécurité (*auditability*)** : conserver une trace de toutes les erreurs et événements critiques pour garantir l'auditabilité, définir les causes et responsabilités.
- On peut (et *devrait*) écrire un log lors d'évènements importants ou le déclenchement de procédures nécessitant des privilèges: authentification d'un admin, installation d'un plugin, etc. Pour cela, on peut utiliser la fonction PHP native error_log :

```php
error_log("Utilisateur X a installé plugin Y");
//Envoi d'un email
error_log('Message critique pour notifier l\'administrateur', 1, "admin@email.com");
```

> En savoir plus sur la fonction native PHP `error_log`

Configuration des logs dans WordPress

Par défaut, si `WP_DEBUG` est définie à `true`, WordPress crée un fichier `wp-content/debug.log` et y écrit les logs.

Ce fichier est placé dans `wp-content` qui est par définition **public** !

Essayer : http://localhost:8080/wp-content/debug.log

Embrassant !

Il ne doit évidemment **pas être accessible aux clients**.

Sur Apache, on peut restreindre son accès via un fichier `.htaccess` placé dans le dossier `wp-content` :

```
<Files "debug.log">
    Require all denied
</Files>
```

Essayer : http://localhost:8080/wp-content/debug.log

Mieux !

Où sont log les erreurs PHP ?

Les **erreurs PHP** peuvent être logés **à différents endroits en fonction de l'emplacement de l'erreur** :

- **Avant** chargement de `wp-settings.php`, WordPress n'a pas mis en place sa redirection des logs vers `wp-content/debug.log.` Le fichier où est consigné l'erreur est défini **par la configuration de PHP**.
- **Après** chargement de `wp-settings.php`, WordPress a mis en place sa redirection des logs vers `wp-content/debug.log.`

```
//Dans wp-settings.php
// Check if WP_DEBUG mode is enabled.
wp_debug_mode(); // enregistre un gestionnaire d'erreur qui redirige les logs vers wp-content/debug.loh
```

> wp_debug_mode() est implémentée dans `wp-includes/load.php`. **Définit le fichier de redirection des logs** et le **niveau** des erreurs consignées (ini_set('error_log', $log_path); et error_reporting())

Couverture complète des erreurs

Pour avoir une **couverture complète** des erreurs, il faut donc :

- Configurer WordPress pour écrire des log (*vu*);
- **Configurer le système de logs de PHP**;

Configurer le système de logs de PHP

Il y a ~~trois~~ deux possibilités :

- *via* des instructions PHP directement dans `wp-config.php` ;
- *via* un fichier de configuration `php.ini` ;
- ~~(Apache) via des directives directement dans le `.htaccess` , propre à l'ancien module CGI d'Apache `mod_php`~~ , **ne fonctionnera pas avec PHP-FPM (que vous devriez utiliser) !**

> Quelle méthode choisir ? Cela dépend de votre environnement de production et de votre liberté d'action sur le serveur. Mon conseil : **Privilégier via `php.ini` si possible !**

Configuration PHP

Il faut utiliser **trois** directives PHP :

- `display_errors` : afficher les erreurs sur la sortie ou non. Valeurs possibles (string) : `On`, `Off`, `stdout`, `stderr`;
- `log_errors` : utiliser le système de logs du serveur. Valeurs possibles (bool) : `1` (oui) ou `0` (non)
- `error_log` : **path de destination des logs** (nom complet du fichier). Ce fichier doit être inscriptible par le serveur (Apache, Nginx, Caddy, etc.). Valeur (string) : chemin du fichier.

> Voir toutes les directives et options pour le système de logging de PHP

Solution *via* des instructions PHP

```
//wp-config.php
@ini_set('display_errors', 0);              // désactive l'affichage
@ini_set('log_errors', 1);                  // active le logging PHP
@ini_set('error_log', '/var/log/wordpress.log'); // log PHP global
```

Solution *via* un fichier `php.ini`

```
; Désactiver l'affichage des erreurs aux utilisateurs
display_errors = Off
; Activer le logging des erreurs PHP
log_errors = On
; Fichier de log PHP global
error_log = /var/log/wordpress.log
; Niveau d'erreur à loger. Important pour pas polluer ses logs
error_reporting = E_ALL & ~E_NOTICE & ~E_DEPRECATED & ~E_STRICT
```

> À propos de cette instruction : `error_reporting = E_ALL & ~E_NOTICE & ~E_DEPRECATED & ~E_STRICT`. Le & est un ET binaire sur les flags. Le ~ est un NOT binaire qui inverse le flag. On prend tous les types d'erreurs (`E_ALL`), puis on **retire** ceux qu'on ne veut pas (`E_NOTICE`, `E_DEPRECATED`, `E_STRICT`).

À privilégier !

Avec Apache PHP-FPM, placer le fichier ini dans le répertoire `/etc/php/8.x/fpm/pool.d/www.conf`. Voir comment configurer PHP-FPM

> Ne plus utiliser l'ancien module CGI PHP d'Apache (`mod_php`). Il est déprécié.

Configuration typique de WordPress en *production*

En production, on veut toujours *log* les erreurs, mais on ne souhaite pas les afficher sur la sortie (et finissent chez le client) :

```php
// Mode production
define('WP_ENVIRONMENT_TYPE', 'production');
// Désactive l'affichage
define('WP_DEBUG', true);           // Activé pour que les erreurs soient loguées
define('WP_DEBUG_LOG', true);       // Consigne les erreurs dans wp-content/debug.log
define('WP_DEBUG_DISPLAY', false); // RIEN AFFICHER SUR LA SORTIE (ECRAN)
define('SAVEQUERIES', false);       // Surtout pas !
define('SCRIPT_DEBUG', false);      // Désactive le chargement de assets du core non minifiées
```

> Un exemple complet de fichier de configuration sera fourni dans la section consacrée à la mise en production.

Configurer WordPress : éléments de configuration personnalisés

Dans le fichier `wp-config.php`, un emplacement vous est indiqué pour **ajouter vos propres configurations**.

```
/* Add any custom values between this line and the "stop editing" line. */

// => AJOUTER VOS CONFIGS ICI (ou inclure un fichier de config custom) <=
if ( ! defined( 'MACONFIG' ) ){
  define( 'MACONFIG', 'MACONFIG_VALUE' );
}

/* That's all, stop editing! Happy publishing. */
/** Absolute path to the WordPress directory. */
if ( ! defined( 'ABSPATH' ) ) {
 define( 'ABSPATH', __DIR__ . '/' );
}
...
```

Configurer WordPress : mises à jour automatiques

Depuis WordPress 3.7, on peut activer une politique de mise à jour du cœur en arrière-plan. Il y a quatre types de mises à jour automatiques que l'ont peut configurer :

- **Les mises à jour du cœur** ;
- Les mises à jour des fichiers de traduction du core (activées par défaut).
- Les mises à jour des extensions (désactivées par défaut) ;
- Les mises à jour des thèmes (désactivées par défaut);

Configurer WordPress : mises à jour automatiques du coeur

Trois types de mises à jour du **cœur** :

- Les mises à jour du **développement** (*pre-release*), si on est aventureux·se ou pressé·e ;
- Les mises à jour **mineures** (6.X.X), pour des raisons de maintenance et de sécurité. **Mises à jour automatiques activées par défaut** ;
- Les mises à jour des versions **majeures** (X.).

Configurer WordPress : mises à jour automatiques du coeur

```
//désactiver complètement TOUS les types de mises à jour automatiques (core, theme, plugins, traductions)
define( 'AUTOMATIC_UPDATER_DISABLED', true );
//mises à jour du core
//aucune mises à jour auto
define( 'WP_AUTO_UPDATE_CORE', false );
//toutes les mises à jour activées ;
define( 'WP_AUTO_UPDATE_CORE', true );
//Recommandé : mises à jour auto pour versions mineures seulement (sécurité, bug fix)
//Valeur par défaut depuis WordPress 5.6
define( 'WP_AUTO_UPDATE_CORE', 'minor' );
```

Tour du Core (suite)

```
wp-admin/           : "gestion de l'administration du site "[x]
wp-content/         : "notre application embarquée par wordpress" [x]
..................................................................... => wp-includes/
index.php           : "point d'entrée de l'application WordPress" [x]
license.txt
readme.html
wp-activate.php
wp-blog-header.php
wp-comments-post.php
wp-config-docker.php
wp-config.php [x]
wp-config-sample.php [x]
wp-cron.php         : "configure un gestionnaire de tâches automatisées (cron job) (déclenché à chaque visite)" [x]
wp-links-opml.php
wp-load.php
wp-login.php        : "gestion du login" [x]
wp-mail.php
wp-settings.php     : "définit variables globales, gère includes (core)" [x]
wp-signup.php       : "gestion de la création de compte" [x]
wp-trackback.php
xmlrpc.php
```

> Commencer à se sentir chez soi...

Tour du Core (suite)

(Quelques) fichiers du *core*, qui implémentent toutes les API de WordPress, dans `/wp-includes` :

- `class-*.php` : tous un tas de services sous forme de classes (hashing, json format, requêtes HTTP, etc.);
- `functions.php` : fonctions principales de l'API de WordPress. Utilisées par le core, les plugins, themes;
- `option.php` : fonctions de l'API Options (ex `add_option()`);
- `formatting.php` : fonctions de formatage, d'échappement, sanitization. Très utiles et importantes;
- `pluggable.php` : fonctions pour surcharger des fonctions du core (ex `wp_mail`);
- `plugin.php` : fonctions de l'API Plugins, pour développer ses propres plugins !
- `user.php` : fonctions de l'API User;
- `post.php` : fonctions de traitement des posts;
- `taxonomy.php` : fonctions de l'API des taxonomies.

Nous allons apprendre à nous servir de nombreuses API (fonctions, classes, variables globales) pour développer notre site sur mesure (theme, plugins).

Les API de WordPress

Pour interagir avec le *core*, WordPress expose de nombreuses API. Voici les principales :

- **Hooks** (*action*, *filter*)
- **Theme API** : **Template Hierarchy**, **Template tags**, **Blocks API (FSE)**
- **Database API** : **Options API**, Transients API, Metadata API
- **Shortcode API**
- **Plugins API**
- **Settings API**
- Widgets API
- REST API
- Localization

> Les API **mises en avant** sont celles que nous allons aborder au cours de cette formation !

La base de données WordPress

Il est utile d'avoir *une vision d'ensemble du schéma* de la base de données de WordPress pour :

- **Comprendre le fonctionnement global** du framework ;
- **Écrire ses propres requêtes SQL sur mesure au besoin** (au delà de l'API `WP_Query`) ;
- **Optimiser** son usage (quelles tables contiennent quoi, quand créer ses propres tables ? placer des indexs ?) ;

Schéma de la base de données WordPress

- WordPress possède **12 tables**. Schéma très (très) **stable**;
- Chaque **nom de table est préfixé** par défaut par `wp_` (Recommandé : **modifier ce préfixe** dans `wp-config.php` via la **variable globale** `$table_prefix` pour des raisons de sécurité);
- Le schéma est *designé* pour être **simple** et **très flexible** (s'adapter à de nouveaux *use cases*);

Points techniques :

- WordPress utilise *dorénavant* le moteur InnoDB pour toutes ses tables (**transactions**, journalisation, **index**, intégrité référentielle, *locking* au niveau des lignes)
- WordPress **n'utilise pas de contrainte de clé étrangère** (*Foreign Key*) ! **Pas d'intégrité référentielle** (uniquement *via core*).

> Des versions majeures de WordPress peuvent changer (très) légèrement la structure des tables (label, champ). Mais la rétrocompatibilité chez WordPress est un objectif principal (pas de craintes).

Schéma de la base de données WordPress : Trois *régions*

{.marp-bg-img}

Trois grandes *régions* :

- Taxonomies;
- Posts et métadonnées;
- Users et métadonnées;

> La table `wp_links` (gestion de liens vers d'autres sites) a été déprécié en WordPress 3.5 (présente pour rétrocompatibilité, toujours utilisable si besoin !) On l'ignorera ici.

> Voir une documentation du schéma publiée sur le Codex

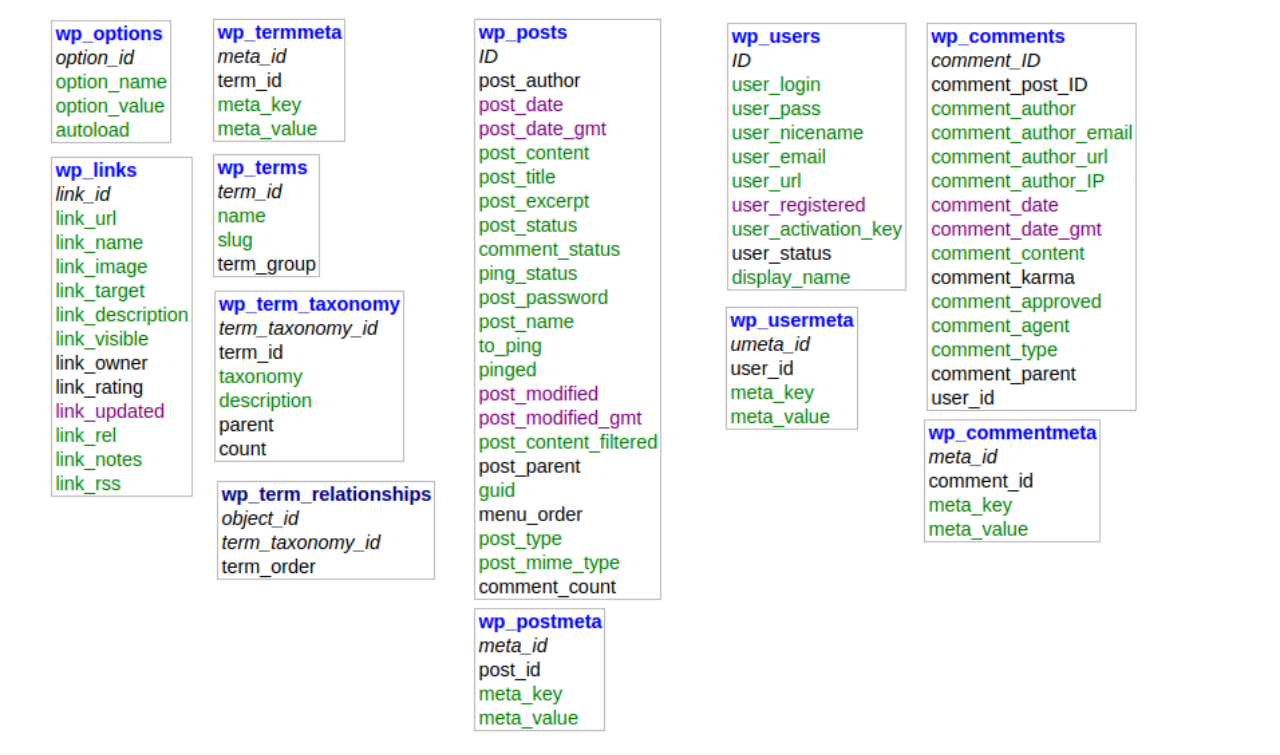


Schéma de la base de données WordPress : Détail (1/2)

{.marp-bg-img}

Posts

- `posts` : tous les posts (défaut et *custom*), **pages**, media records, revisions. Table la plus grosse d'un site WordPress ;
- `postmeta` : toutes les métadonnées des posts (**custom fields**).

Taxonomies (catégories et tags)

- `terms` : tous les *termes* (valeurs) des différentes taxonomies ;
- `termmeta` : métadonnées des termes ;
- `term_taxonomy` : association terme-taxonomie ;
- `term_relationships` : table de jointure term et posts.

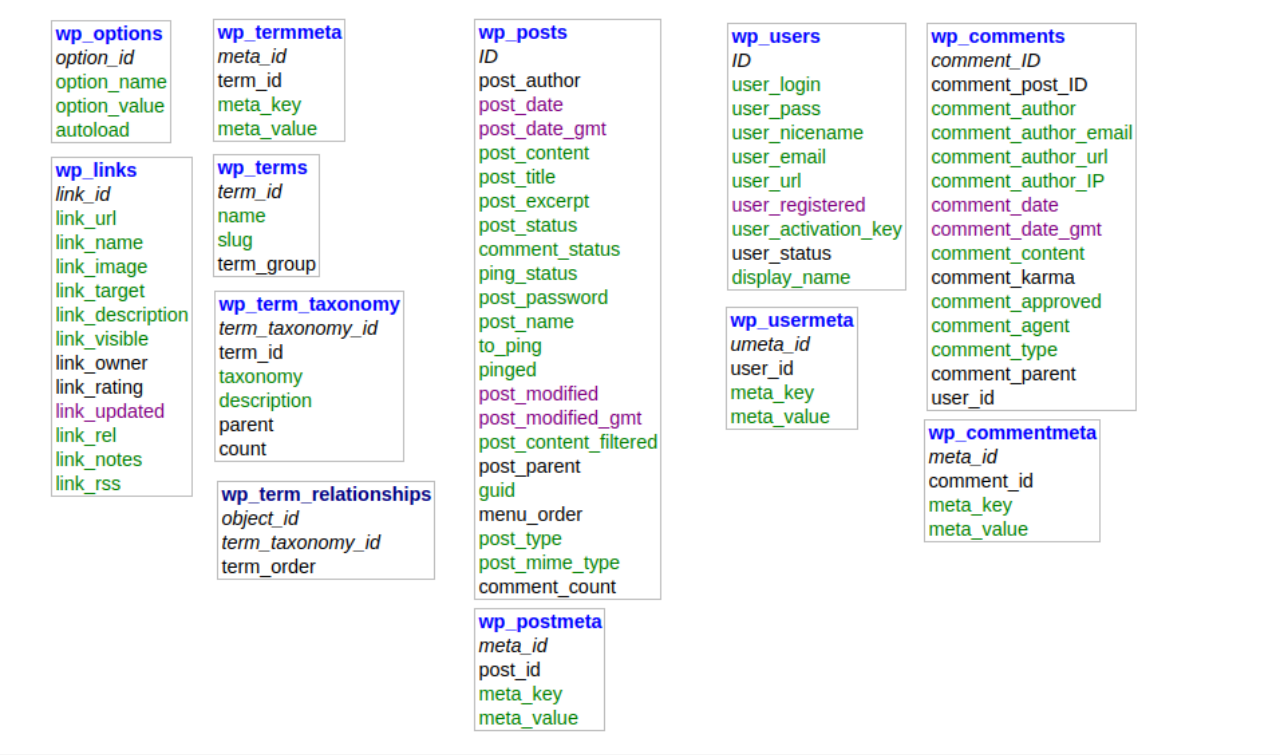


Schéma de la base de données WordPress Détail (2/2)

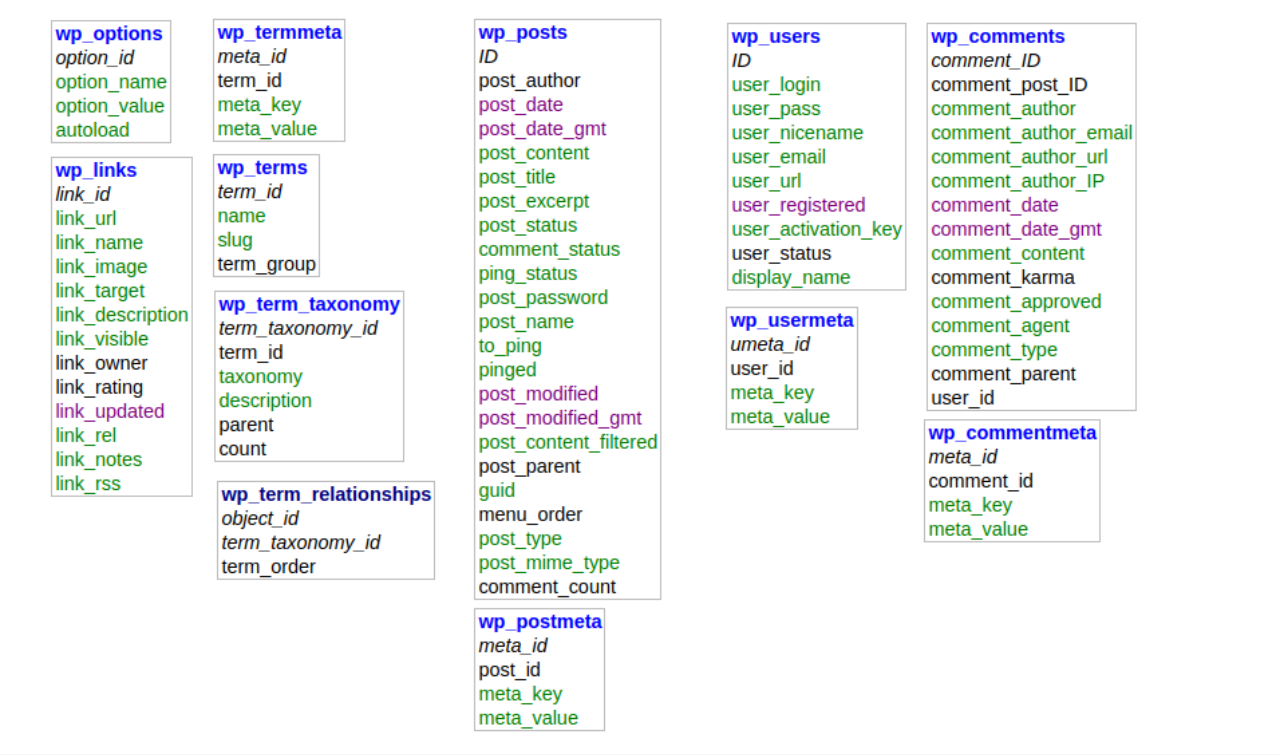
{.marp-bg-img}

Utilisateurs

- `users` : login, password, email, etc. ;
- `usersmeta` : **roles** (user level) ;
- `comments` : commentaires ;
- `commentmeta` : métadonnées des commentaires.

Options

- `options` : toutes les options de la page Settings et Settings API, roles et capabilities existant dans le système.



De la base de données aux "primitives" WordPress

Données primitives de WordPress : **pages**, **posts**, **taxonomies**, **utilisateurs**, **rôles**

Posts (et pages)

- **Posts** : "objet"/structure de données de base de WordPress, champs par défaut et personnalisés (*custom fields*)
- **Pages** : Un type "spécial" de post (`post_type` = 'page'), logique spécifique (hiérarchie parent/enfant, contenu *statique*)

> Presque tout est un *post* en WordPress !

Taxonomies

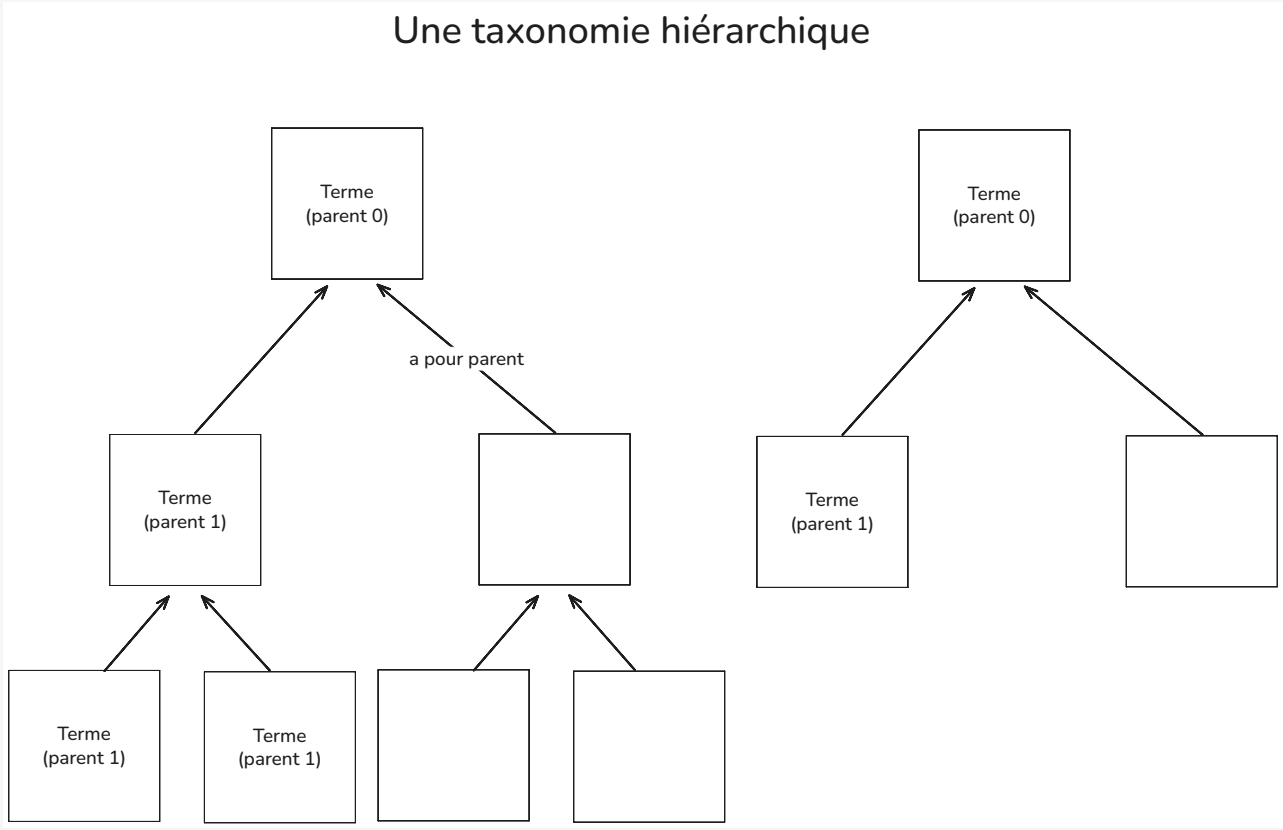
- **Taxonomie** : Terme CMS (Architecture de l'information).
- **Définition** : (syn *taxinomie*) Science des classifications. Classification, suite d'éléments formant des listes qui concernent un domaine, une science. Arbre hiérarchique de catégorisation thématique (*répertoire*). Permet de regrouper, organiser et filtrer les objets (dans WordPress des posts, pages, CPTs, Users). ;
- **Deux types** de taxonomies :
  - **Catégories** :
    - **hiérarchique** (un terme peut avoir un parent),
    - très structurant (design du site, sitemap, etc.) et plus rigide
    - Associées à des posts
  - **Etiquettes/Tags** :
    - **non hiérarchique**
    - plus *dynamique*, *transversal*
    - Associés à des posts et des pages.
- **Termes** (*terms*) : éléments d'une taxonomie .
  - Exemple : `Football` et `Tennis` sont des *termes* de la *taxonomie* `Sport`,
  - Dans une taxonomie hiérarchique, un terme peut donc avoir un terme parent. Par exemple, le terme `Football en salle`

Taxonomie hiérarchique

{.marp-bg-img}

Terminologie :

- Une *taxomonie* hiérarchique ;
- Chaque **terme** (élément) peut avoir un parent ou non. Chaque terme peut être un *répertoire*.

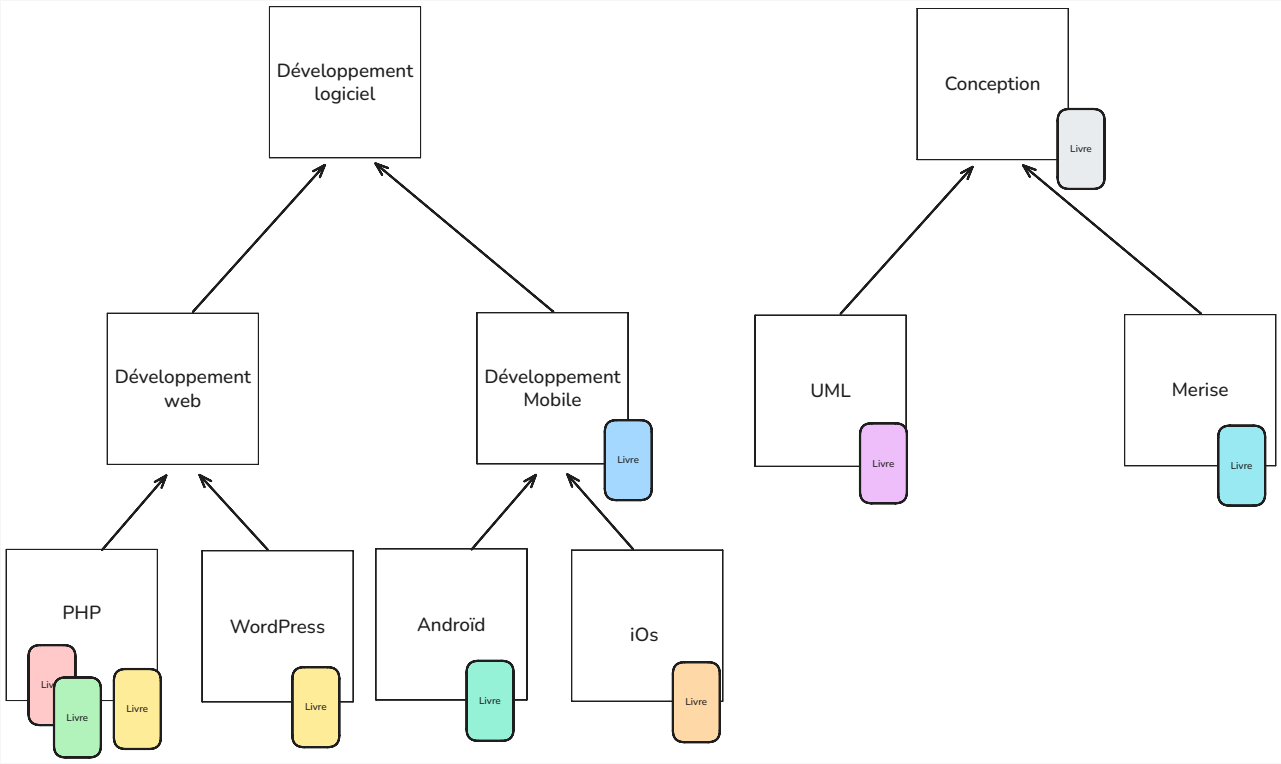


Exemple de taxonomie hiérarchique

{.marp-bg-img}

Un exemple :

- Chaque item (*post*) peut appartenir à un ou plusieurs termes (i.e être dans une ou plusieurs "catégories")

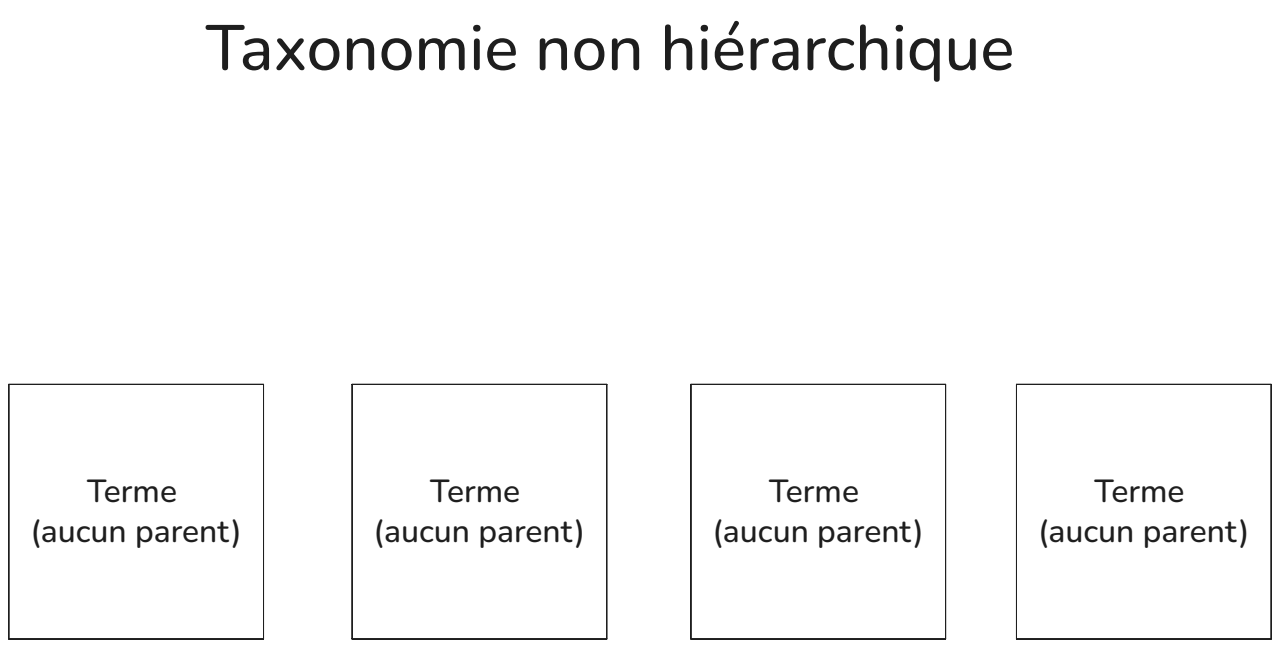


Taxonomie non hiérarchique

{.marp-bg-img}

Terminologie :

- Une *taxomonie* non hiérarchique ;
- Aucun terme ne peut avoir de parents.

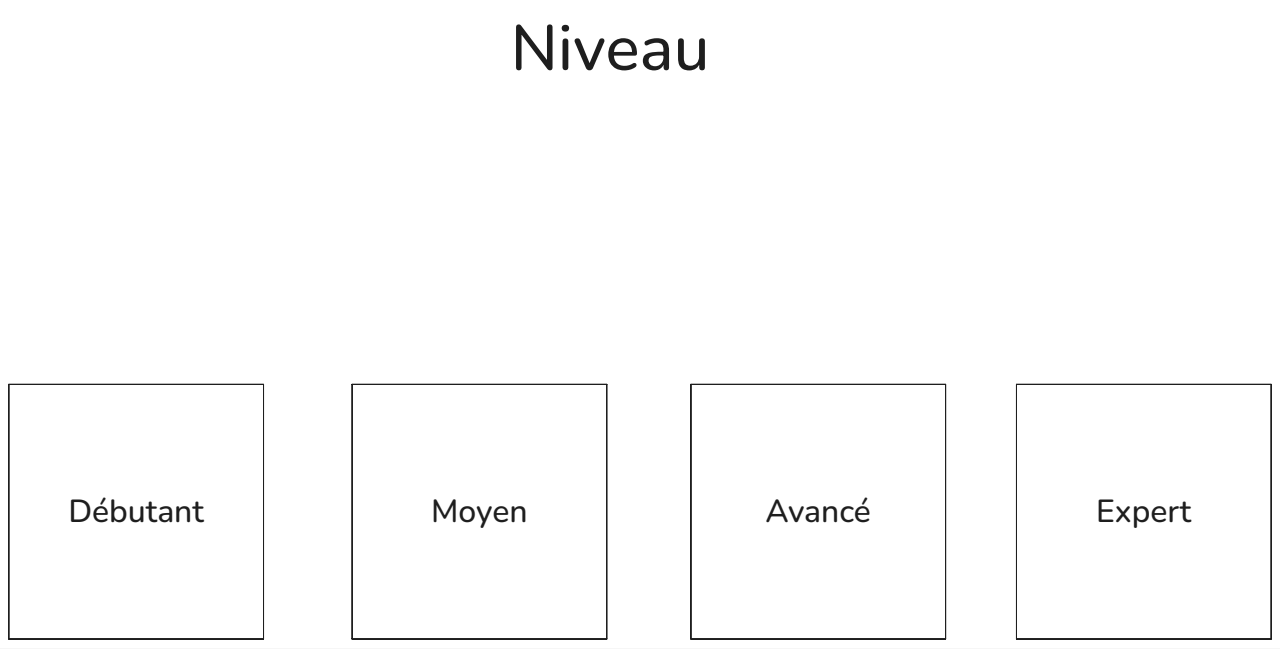


Exemple de taxonomie non hiérarchique

{.marp-bg-img}

Exemple :

- Une *taxomonie* non hiérarchique ;
- Aucun terme ne peut avoir de parents.
- Un post peut appartenir à un ou plusieurs termes. Par exemple, l'ouvrage "Développer votre premier site web avec PHP" pourrait appartenir aux termes "Débutant" et "Moyen".



Le cas des taxonomies par défaut "Categories" et "Tags"

- **Taxonomies par défaut** : à l'installation, WordPress crée deux **taxonomies** :
  - *Categories* (`category`), hiérarchique
  - *Tags* (`post_tag`), non hiérarchique
- On peut **créer ses propres taxonomies** (*Custom Taxonomies*)
- **Cas particulier** : si un post n’a pas de catégorie, il est automatiquement associé au terme `Uncategorized` de la taxonomie `category` .

Design des taxonomies

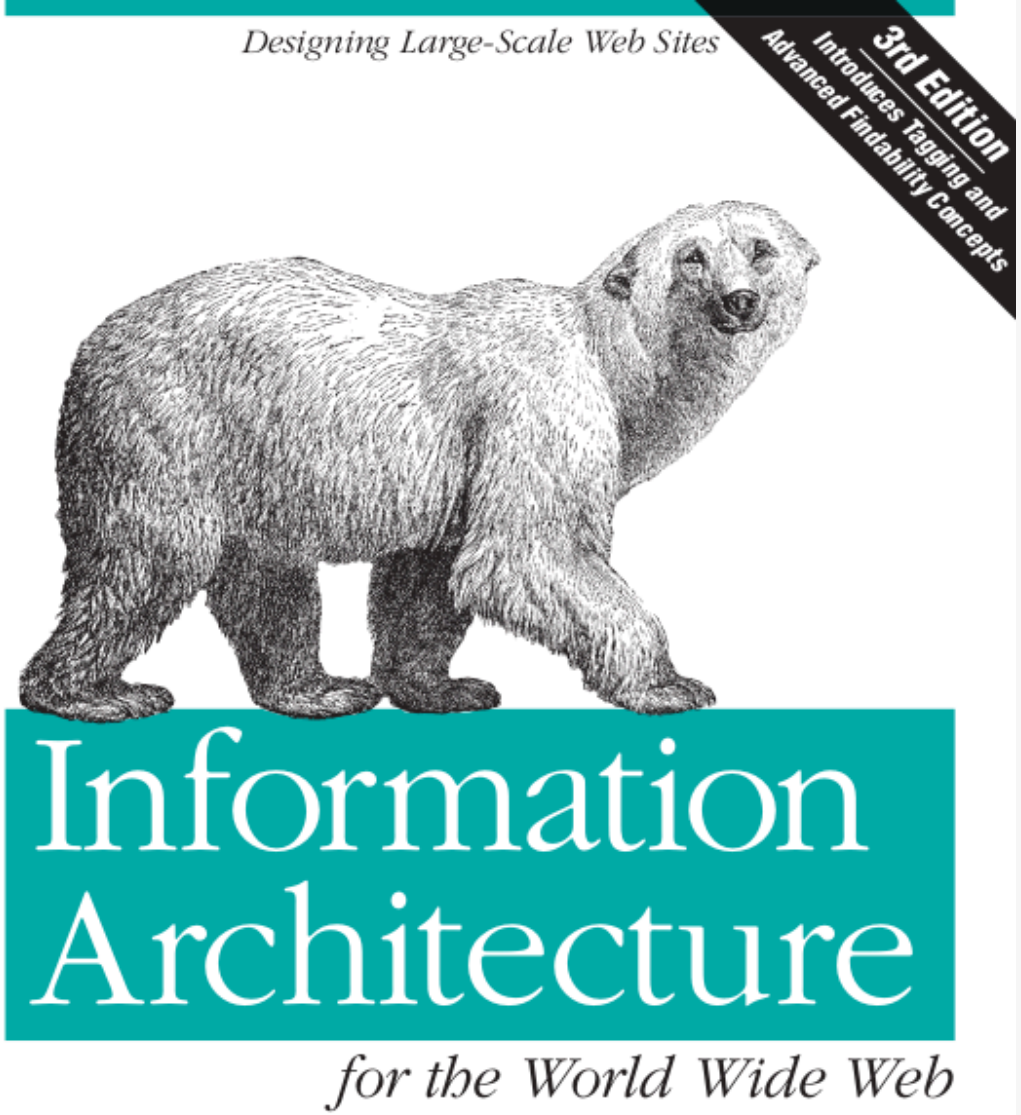
{.marp-bg-img}

Un vaste sujet (passionnant) !

- Catégories exclusives ou non ?
- Taxonomies hiérarchiques ou non ?
- Rechercher du contenu par quel moyen ?
- Quels labels utiliser ?

Pose des questions sur la *navigabilité* et les *possibilités* de recherche offertes par le système, et donc sa structure et son design.

> *Feuilletage recommandé* : Information Architecture for the World Wide Web 3rd Edition, de Peter Morville et Louis Rosenfeld, publié chez O'Reilly Media, 2006. Consulter notamment le chapitres 1 et 2.



Gestion des taxonomies en détail

{.marp-bg-img}

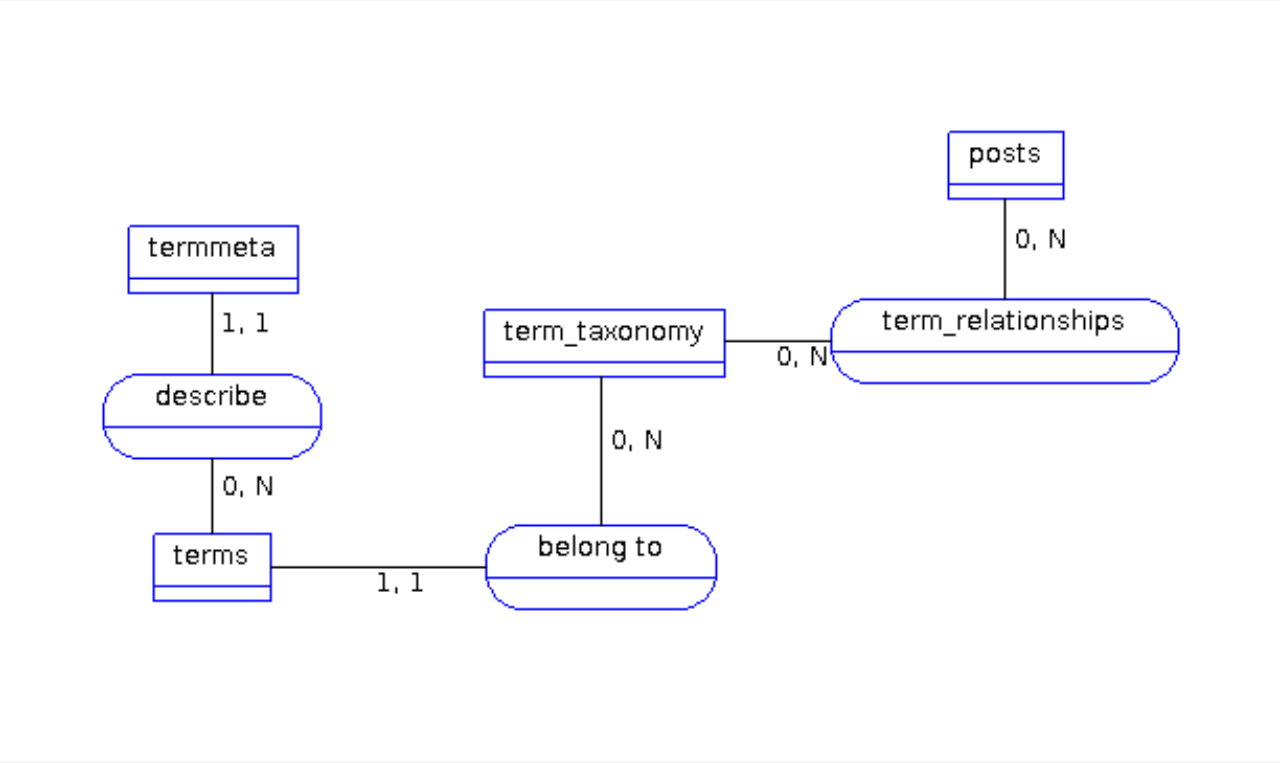
Les tables pour la gestion des taxonomies et leurs associations (formalisme MERISE).

Exemple :

- Deux termes `A` (id `4`) et `B` (id `5`) de la taxonomie `category`;
- Un post (id `6`) dans la catégorie `A`;

En base :

- `term_taxonomy` : contient association terme-taxonomie. (`1`, `4`, `category`), (`2`, `5`, `category`)
- `terms` : valeurs des termes. `(4, 'A')`, `(5, 'B')`
- `term_relationships` : association post-termes . `(6, 1)`



Users et métadonnées : *roles*, *capabilities*

- **Utilisateurs** (*Users*) : représentent les comptes qui se connectent au site.
  - Stockés dans la table `users` (login, email, password *hashé* (bcrypt + salt))
  - Métadonnées : stockées dans `usermeta` (préférences, informations supplémentaires).
- **Rôles** (*Roles*) : définissent ce qu'un utilisateur peut faire via ses `capabilities`.
  - Définis dans la table `options` dans l'attribut `wp_user_roles`.
  - Exemples de rôles par défaut : `administrator`, `editor`, `author`, `contributor`, `subscriber`.
  - Chaque rôle est **associé** à un ensemble de *capacités* ou `capabilities` (ex: `edit_posts`, `publish_posts`, `manage_options`).
- **Lien entre utilisateur, rôle et capabilities** :
  - L'utilisateur reçoit un rôle, **le rôle définit les capabilities** (table `usermeta`)
  - Les capabilities contrôlent l'accès aux fonctionnalités, vérifiables avec la fonction `user_can()`
- On peut **créer ses propres rôles et capabilities** en fonction des **besoins métiers** et du **processus éditorial**.

Le cœur du système de permissions WordPress

Les rôles et capacités sont stockés sous forme d'un tableau PHP sérialisé en base.

Extrait de la valeur par défaut (à l'installation) de `wp_options.wp_user_roles` :

```
a:5:
  {
    s:13:"administrator";
    a:2:{
      s:4:"name";s:13:"Administrator";s:12:"capabilities";
      a:61:{
        s:13:"switch_themes";b:1;
        s:11:"edit_themes";b:1;
        s:16:"activate_plugins";b:1;
        s:12:"edit_plugins";b:1;
  ...
```

Pourquoi se familiariser avec la base de données de WordPress

Se familiariser avec la base permet :

- De **mieux comprendre les API offertes par WordPress** (fonctions, classes, variables) que l'on utilisera dans le développement du site (en PHP);
- Prendre *confiance*;
- Savoir **écrire ses propres requêtes SQL sur mesure** au besoin;
- S'**inspirer** de certains choix de conception intéressants.

Pratique : interroger la base

Pour cela, utilisez *au choix* :

- adminer ;
- le client `mysql` présent sur le conteneur `db`. Pour ouvrir une session :

```
docker compose exec -it db mysql -uroot -proot
```

Exercice requêtes SQL

1. **Retrouver** dans la base l'**identifiant**, le **titre** et le **status** du post *Hello world!* crée par défaut.
2. Dans le *dashboard*, **modifier** le contenu du post et refaire la requête. Que remarque-t-on ? Afficher également le parent du post;
3. Dans le *dashboard* :
    i. **Créer** une nouvelle catégorie `formation`,
    ii. **Créer** deux nouveaux posts et les **placer** dans cette catégorie,
4. **Écrire une requête SQL** pour :
    i. Retrouver le **slug** de la catégorie `Formation` dans la base de données;
    ii. Retrouver uniquement les posts appartenant à la catégorie `formation`. Afficher le **titre** du poste, le nom de la **taxonomie** et le nom du **terme**;
    iii. **Modifier** le **mot de passe** de votre compte administrateur du site directement dans la base (peut s'avérer utile !)

Solution

```sql
-- Question 1
SELECT ID, post_title, post_status, post_parent FROM wp_posts WHERE post_title = "Hello World!";
-- Question 4.1
SELECT slug FROM wp_terms WHERE name="Formation";
-- Question 4.2
SELECT p.post_title, tt.taxonomy, t.name
FROM wp_term_relationships tr
JOIN wp_posts p ON tr.object_id = p.ID
JOIN wp_term_taxonomy tt ON tt.term_taxonomy_id = tr.term_taxonomy_id
JOIN wp_terms t ON t.term_id = tt.term_id
WHERE t.name = "Formation";
-- Question 4.3
UPDATE wp_users
SET user_pass = MD5('nouveau_mot_de_passe')
WHERE user_id = 1;
```

Attention, ici on *hash* le mot de passe avec l'algorithme MD5 (déprécié !). À la première connexion, WordPress va re *hasher* le mot de passe en utilisant bcrypt et son salt. Se reconnecter immédiatement !

En cas de besoin (plus d'accès à `/login`), utiliser plutôt WP-CLI :

```
docker compose run --rm wpcli user update 1 --user_pass="new_pass"
```

Remarque (importante) sur les révisions

Observations :

- On apprend que chaque post **peut posséder plusieurs posts enfants**, correspondant aux **révisions** (historique pour rollback) ;
- La taille de `wp_posts` **augmente rapidement avec le nombre de posts** ! ($N_{posts} \times N_{revisions}$)
- Par défaut, WordPress garde **un nombre illimité de révisions** d'un post à chaque sauvegarde après publication, si le contenu du post a changé (title, content, excerpt) !

> Essayer de créer plusieurs révisions en modifiant le contenu du post "Hello World" et en enregistrant.

Conséquences et problèmes :

- Certaines requêtes `WP_Query` deviennent moins performantes.
- Backups plus lents et volumineux ;

Intervenir sur les révisions

Pour inspecter le nombre de révisions en base :

```
SELECT COUNT(ID) as "Nombre de revisions :" FROM wp_posts WHERE post_type = 'revision';
```

Solution :

Limiter le nombre de révisions via la constante `WP_POST_REVISIONS` (2-3 suffisent généralement), à placer dans le fichier `wp-config.php` :

```
//Dans le fichier wp-config.php :
//Limité à 3 révisions
define( 'WP_POST_REVISIONS', 3 );
//Désactiver les révisions, si pas besoin d'historique
define( 'WP_POST_REVISIONS', false );
```

Supprimer toutes les révisions de chaque post **sauf la dernière** :

```
DELETE FROM wp_posts
WHERE post_type = 'revision'
  AND ID NOT IN (
    -- Sélectionne la dernière révision de chaque post
    SELECT id FROM (
      SELECT MAX(ID) AS id
      FROM wp_posts
      WHERE post_type = 'revision'
```

Tour du Core (suite et fin) `wp-content` : user space

Dans le répertoire `wp-content` :

- `themes` : liste des thèmes (templates et assets) ;
- `uploads` : fichiers téléversés via la Médiathèque (images, vidéos, documents, etc.). Organisé par année/mois par défaut ;
- `plugins` : plugins classiques, activables via l'admin WordPress. Étendre les fonctionnalités du core ;
- `mu-plugins` : *Must-use plugins* :
  - Plugins automatiquement activés, **sans possibilité de désactivation via l'admin**,
  - Initialement pour le multisite (*mu*), mais utilisé dans tout type d'installations pour des **fonctionnalités critiques ou globales**;
- `upgrades` : utilisé temporairement pour stocker les fichiers lors de mises à jour automatiques de WordPress, plugins ou thèmes,
- **Répertoires personnalisés** : certains plugins ou thèmes peuvent créer leurs propres dossiers pour stocker du code, des assets ou des caches.

Finaliser la préparation de l'environnement de développement

La configuration de WordPress se partage entre les sources *et* les données en base.

Dashboard : *Guide Tour* et quelques configurations

Se connecter en tant qu'administrateur·ice.

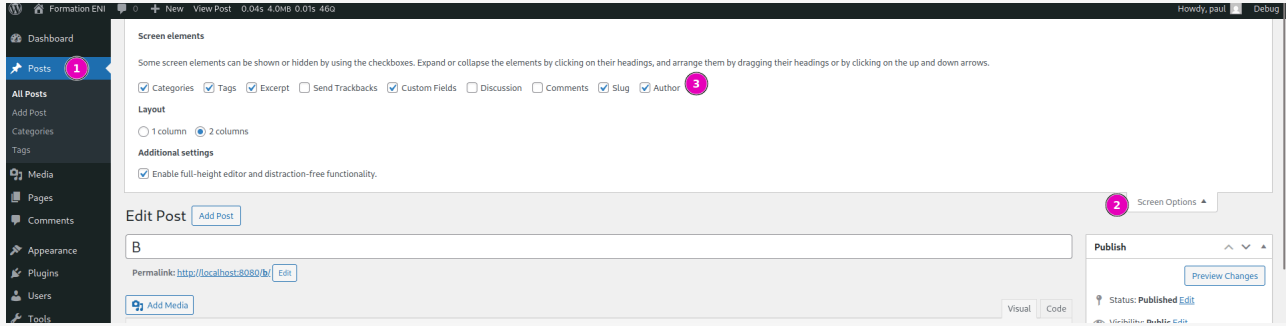
Guide Tour :

- Posts :
  - Taxonomies,
  - Status,
  - Visibilité;
- Media ;
- Pages ;
- Comments ;
- Appearance (Menus, Site Editing, Themes) ;
- Plugins (Chercher, Installer, Activer) ;
- Users ;
- Tools ;
- Settings (General, Reading, Writing, **Permalinks**).

Personnaliser le dashboard

{.marp-bg-img}

- Dans les *Screens Options* d'un Post :
  - **Afficher** Categories, Tags, **Excerpt**, **Custom Fields**, **Slug**, **Author**;
  - **Déplacer** les *Méta Box* à votre gré (disposition enregistrée dans votre profil)
- Dans l'édition de votre compte utilisateur :
  - **Choisir** un *color scheme* (utile pour différencier visuellement les environnements !)



Activer les permaliens

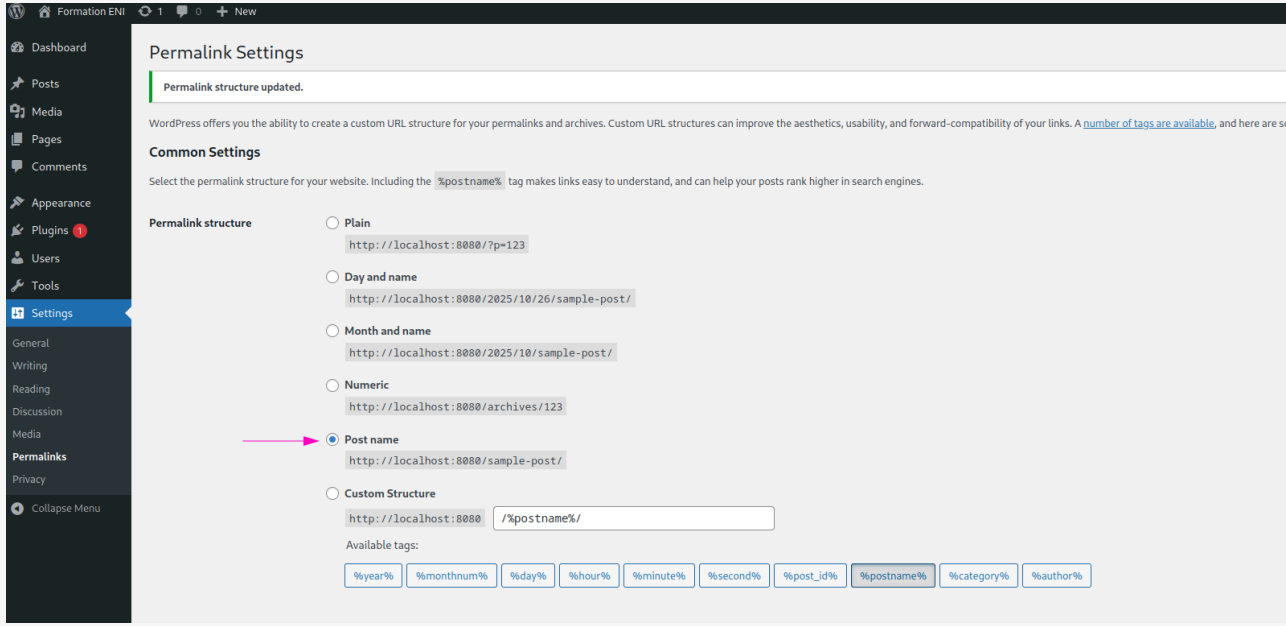
{.marp-bg-img}

Par défaut les URLs de WordPress sont de la forme `/?key=value`. Par exemple : `/?p=1`, `/?cat=2`, etc. Elles sont basées sur des paramètres d'url (*query part*).

Activer les permaliens (réécriture d'URL)

Avantages :

- Décrit une **hiérarchie des ressources** exposées par le site. Par ex : `/category/music` indique que `/category` est aussi une ressource valide;
- **URL lisibles** par des humains (*non opacité* des URL);
- **SEO** (keywords dans l'URL);
- Pool d'URL *indépendantes de la plateforme* WordPress;



Définir la timezone, le format de la date et de l'heure

Dans la section `Settings/General` :

- **Date** : Mettre le format `d/m/Y` ;
- **Heure** : Mettre le format `H:i` ;
- **Définir** la timezone (`Paris`).

> À adapter en fonction de la langue du site et de ses règles d'usage.

Installer des plugins de développement utiles

Installer les plugins suivants :

- Classic Editor, pour éditer facilement les champs customs et le contenu
- Query Monitor, pour monitorer (cache, requêtes HTTP, SQL, etc.)
- User Switching, pour changer rapidement de compte/role
- Faker Press, pour générer du faux contenu efficacement (peupler le site) pour tester la logique et les templates

> À utiliser **uniquement en environnement de développement** !

Créer un thème minimal (déjà fourni)

Fichiers requis :

- `style.css` ;
- `index.php`.

> On développera ce thème (et ses plugins) tout au long de la formation. On a ajouté `screenshot.png` (optionnel) pour que le thème ait un thumbnail.

Contenu du fichier `style.css`

```
/*
Theme Name: Mon Thème
Author: Prénom Nom
Description: Une description du thème
Version: 0.1.0
License: Infos sur la licence
Text Domain:
Tags:
*/
```

Contenu du fichier `index.php` (test)

```php
<?php

/**
 * Point d'entrée du thème
 */

//Modifier l'expéditeur de l'email par défaut via un filtre
add_filter('wp_mail_from', function ($email) {
    return 'test@mailhog.local';
});

//Envoi d'un email de test
if (wp_mail("test@example.com", "MailHog test", "Hello from WordPress")) {
  echo "email sent";
} else {
  echo "Error: email not sent.";
}

//Ecrire un log
error_log('some log');

//Afficher les informations sur la configuration de php utilisée par WordPress
phpinfo();
```

Activer le thème

1. **Activer le thème** *Mon Thème*. Ce thème servira de fil rouge pour une partie de la formation. C'est un **thème classique** (non basé sur des blocs, vu plus tard);
2. **Visiter** le site (`/`) et vérifier que l'email est bien présent dans le mailcatcher;
3. Vérifier que le fichier `wp-content/debug.log` existe et contient le log.

Pipeline assets CSS/JS

Le service `gulp` défini dans le fichier `compose.yaml` (basé sur gulpjs) :

1. Compile les sources SCSS (Sass) en CSS;
2. Minifie CSS et JS;
3. Copie les versions minifiées dans les sources du thème.

Lancer le service :

```
#ne pas le détacher pour debug (voir des erreurs)
docker compose exec gulp gulp
```

On s'en servira plus tard !

> À adapter/étendre à vos usages (intégrer vos frameworks CSS, comme Bootstrap, ou JavaScript favoris). On n'utilisera pas de framework CSS/JS dans cette formation pour ne pas introduire de complexité inutile.

Administration WordPress via WP-CLI

WP-CLI est un outil puissant (**programmable !**). Il permet de :

- Manipuler la base de données (dump, **search/replace URL**, etc.)
- Inspecter votre application WordPress;
- Installer des plugins;
- Mettre à jour WordPress;
- Etc.

```
#check la configuration de wp-cli
docker compose run --rm wpcli wp config list
docker compose run --rm wpcli cli info
docker compose run --rm wpcli db tables
```

> WP-CLI utilise les informations de connexion présents dans `wp-config.php` pour se connecter à la base de données.

Debug dans l'écosystème docker

```
#Lister les conteneurs actifs
docker compose ps
#Afficher en temps réel les logs du conteneur WordPress (log d'Apache ici)
docker compose logs -f wordpress
```

Fonctionnement de WordPress

Cycle de vie de l'application WordPress : de la requête à la réponse HTTP.

Cycle de vie de l'application WordPress : du serveur web au chargement de WordPress

1. **Requête HTTP est traitée par le serveur web** (Apache, Nginx, etc.)
    - Si l’URL correspond à un fichier existant (ex: `/wp-login.php`, `/wp-content/style.css`), le serveur **sert directement le fichier**,
    - Sinon, la requête est redirigée vers `index.php`* de WordPress (via `.htaccess` avec Apache ou la configuration Nginx).
2. **Initialisation de WordPress** : `wp-settings.php` est chargé :
    i. Initialisation du *Core*,
    ii. Chargement des plugins, MU-plugins et thèmes,
    iii. Définition des constantes, classes et fonctions globales;

> *Que la réécriture des permaliens soit activée ou non (voir slide suivante)

Cycle de vie de l'application WordPress : production de la réponse HTTP

3. **Contrôleur** : WP::main() **orchestre** la logique centrale :
    i. parse_request() : transforme l'URL en *query vars* pour préparer les requêtes SQL à effectuer (post_type, p, name, etc.). L'intention du client est dans l'URL, WordPress en déduit un ensemble de données à récupérer,
    ii. send_headers() : prépare et écrit les headers de la réponse HTTP (Content-Type, Status, **Cache HTP**, etc.). Il ne manque plus que le *body* (HTML) de la réponse à produire,
    iii. WP_Query->query() : **Exécute la/les requêtes SQL**, et construit le **contexte** `$wp_query` : objets $post, collections, flags (`is_single()`, `is_page()`, etc.) etc.
    iv. **Hooks** (actions/filters) : **injection de logique métier** par plugins ou thème (`pre_get_posts`, etc.).
    v. **Sélection** de la vue : *choix* du template basé sur la **Template Hierarchy**,
    vi. **Rendu** du template. Le template est exécuté avec **accès au contexte** => **Loop WordPress**. Produit le document HTML,
    vii. **Réponse HTTP** : HTML final est renvoyé au client (les headers et le statut HTTP ont été préparés par le contrôleur à l'étape 2).

Remarque sur la réécriture des permaliens

Par défaut, on l'a vu, les URL de WordPress concernent toujours la ressource racine (`/`) et en produit des variations avec des query parameters (ex `/?p=1`). Le fichier `index.php` est donc toujours le point d'entrée.

Réécrire les permaliens génère (avec Apache) le fichier `.htaccess` suivant :

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]
RewriteBase /
RewriteRule ^index\.php$ - [L]
# Si le fichier ou dossier existe physiquement, il est servi directement.
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# Sinon, redirige toute requête vers index.php, le point d’entrée WordPress.
RewriteRule . /index.php [L]
</IfModule>
```

Cette configuration dit :

- Sert directement le fichier s'il existe ;
- Redirige vers le fichier `index.php` sinon.

La réécriture des permaliens ne modifie donc pas le fonctionnement du cycle de vie de l'application.

Template hierarchy (un aperçu)

{.marp-bg-img}

La **template Hierarchy** est un **mécanisme central** du **fonctionnement des thèmes** à comprendre.

Elle détermine le choix de la vue et donc du script PHP de votre thème qui va être utilisé **en fonction** du contexte, et donc de l'**URL**.

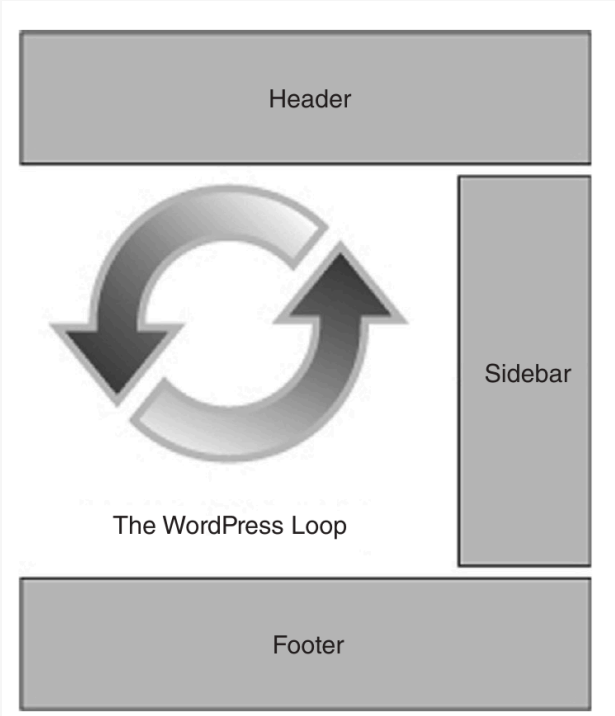
> On y reviendra plus en détail par la suite.

La (fameuse) "Loop WordPress"

{.marp-bg-img}

- Connexion entre la base de données et le template (**contexte**). **Par défaut**, un contexte *pertinent* (résultat de requêtes auprès de la base) est fourni au template;
- Utilisée principalement dans les **templates**. Mais aussi plugins, widgets, etc.
- Au coeur du thème, **contrôle comment le contenu est affiché** ;
- **À utiliser**, qu'il s'agisse d'**un item** (post, page, author, etc.) ou d'**une liste d'items** (posts, archive, etc.)
- Il est possible de **créer des loops customs** n'importe où.

```
//Dans un template (page, post, archive, etc.)
//have_posts() interroge le contexte
 <?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
    <div class="entry">
        <?php the_content(); ?>
    </div>
 <?php endwhile;?>
```



> Source de l'image, p 83.

Fonctionnement de la Loop

- `have_posts()` : vérifie si le `$wp_query` courant (**contexte**) contient encore des posts;
- `while ( have_posts() )` : itère sur tous les posts (posts au sens large, "contenu") du **contexte**;
- `the_post()` (Effets de bords !) :
  - **Passe au post suivant** dans `$wp_query->posts` (fait avancer un pointeur),
  - **Met à jour les variables globales** `$post` + ses métadonnées,
  - Usage des *template tags* (`the_title()` , `the_content()` , `get_the_ID()`) pour accéder au `$post` courant et à ses données.

```
//Dans un template (page, post, archive, etc.)
 <?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
    <div class="entry">
        <?php the_content(); ?>
    </div>
 <?php endwhile;?>
```

> On verra dans la suite de la formation comment créer nos propres loops customs.

Pattern *Model-View-Controller*

- *Model* = code métier. **Indépendant de tout le reste de l'application**, du code système ou du framework.
- *View* = présentation. Peut être réutilisée par plusieurs controller. Dépendance faible au model.
- *Controller* = traiter la requête et produit la réponse. Appelle le Model et la Vue.

WordPress, un framework *Model-View-Controller* ?

- **Controller** :
  - `WP::main()` : point d'**entrée** du traitement de la requête
    - rewriting + `parse_request` : **interprète l'URL** en requête logique
    - `WP_Query` (phase résolution) : construit le **contexte** et **récupère les données**
    - **Hooks** (*actions*, *filters*) : **injection de logique métier** selon des **évènements**
    - **Template Hierarchy**: **sélectionne le template** du thème pour présenter les données
- **Model** :
  - Objet `WP_Query` (côté lecture), WP_Post, WP_User, WP_Term, Meta API, Options API
  - **Logique métier applicative (Plugins / MU-plugins)**
- **View** :
  - Templates du **thème** (rendu)
  - Assets (CSS, JS, fonts media) définis pour le **thème**

> Idéalement, **on devrait pouvoir changer de thème tout en maintenant le bon fonctionnement du site** !

Au final, "concrètement", qu'est ce qu'un site web WordPress ?

Un programme écrit en PHP basé sur une base de données relationnelle (MySQL).

Pour migrer ou installer un site WordPress il faut donc :

- Installer WordPress (**core**);
- **Configurer** (éditer `wp-config.php`);
- Y (dé)placer vos **données** :
  - `wp-content` : themes, plugins, mu-plugins, media;
  - *Dump* de la base de données (contenu éditorial). Attention : URL dépendant !.

> On reviendra sur les *workflows* de déploiement plus tard dans la formation

Apprendre WordPress, utiliser les différentes documentations

Pour monter en compétences sur WordPress on peut utiliser les différents niveaux de documentation suivants :

- **Inline doc** : code source !
  - **WordPress impose des coding standards sur le core**. Tout fichier, fonction, variable **a une documentation** sous forme de commentaire (*PHPDoc block*),
  - **La source ultime de vérité** !
  - Permet de *découvrir* (pas à pas) le fonctionnement du core et d'autres fonctions,
- Code Reference :
  - **La doc de référence (web)**. Accéder à toutes les primitive de WordPress,
  - Générée à partir de l'*inline doc* et enrichie,
- Codex : Documentation d'origine sous forme de Wiki. **Plus maintenu**, **dépréciation en cours**, redirige vers la doc de référence. Toujours quelques ressources utiles. *Danger zone !*
- Guides officiels. Documentation de *haut niveau* sur différents sujets. Toujours utile et recommandé.

Apprendre WordPress : code source avec commentaires standardisés

Par exemple, dans le fichier `wp-includes/functions.php` :

```
/**
 * Converts given MySQL date string into a different format.
 *
 *  - `$format` should be a PHP date format string.
 *  - 'U' and 'G' formats will return an integer sum of timestamp with timezone offset.
 *  - `$date` is expected to be local time in MySQL format (`Y-m-d H:i:s`).
 *
 * Historically UTC time could be passed to the function to produce Unix timestamp.
 *
 * If `$translate` is true then the given date and format string will
 * be passed to `wp_date()` for translation.
 *
 * @since 0.71
 *
 * @param string $format    Format of the date to return.
 * @param string $date      Date string to convert.
 * @param bool   $translate Whether the return date should be translated. Default true.
 * @return string|int|false Integer if `$format` is 'U' or 'G', string otherwise.
 *                          False on failure.
 */
function mysql2date( $format, $date, $translate = true ) {
 if ( empty( $date ) ) {
  return false;
 }

  ...
```

Disposer de la doc de référence directement dans l'IDE

Recommandé !

Par exemple, dans VS Code, utiliser l'extension Hooks IntelliSense for WordPress, en plus de l'extension PHP Intelephense

Bonnes pratiques (première volée) 1/3

- Se préparer un environnement de développement dockerisé (fait);
- Configurer un *linter*. Par exemple, appliquer les *Coding Standards* de WordPress à vos sources, avec le linter PHP_CodeSniffer (ou PSR 12, etc.). On le fera après;
- **Versionner uniquement** :
  - vos themes,
  - vos mu-plugins,
  - vos plugins,
- Si développement régulier de sites avec WordPress, créer et maintenir un **starter theme** entièrement configuré, et automatiser la création d'un nouveau projet (avec `wp-cli`);

Bonnes pratiques (première volée) 2/3

- Préférer les **changements dans le code plutôt qu'en base de données** (versionnement). La base de données ne devrait pas contenir de données indispensables au fonctionnement du "thème" :
  - Base de données : Contenu éditorial,
  - *Codebase* : **Logique métier**/Présentation. Si données indispensables en base, **le "thème" doit vérifier leur existence et les générer**;
- **Adopter la façon de faire de WordPress** (même si cela parfois ne nous plait pas !) : conventions de nommage, utilisation des fichiers, schéma de la base de données, hooks, variable globales, etc. C'est un **framework**.

Bonnes pratiques (première volée) 3/3

- **S'appuyer sur le schéma WordPress existant au maximum**. Utiliser `posts`, `postmeta`, `terms`, `term_relationships`, `users`, `usermeta` pour **la majorité des besoins**. Par exemple :
  - Un *post* en WordPress = Un *objet* générique ;
  - Un modèle de données = post + meta données.
- Créer ses propres tables seulement si :
  - Le modèle ne rentre pas dans le pattern `post` + `meta` ;
  - Le domaine métier **impose des contraintes fortes non exprimables via la méta** (intégrité est fondamentale, méta indexable sur des gros volumes de données, etc.)

> Le schéma de base de WordPress (posts, metas, terms, users) est stabilisé, éprouvé sur des millions d'installations, et intégré au système de cache du coeur (objet cache, page cache, query caching). Ses tables ont des index connus, et l'API `WP_Query` sait les exploiter efficacement. Beaucoup de plugins se basent dessus, ce qui garantit compatibilité et prévisibilité du comportement. Créer ses tables = une dette à assumer !

En conclusion

Dans cette première séquence nous avons :

- Vu l'origine de WordPress, son mode de développement, son état actuel et son évolution ;
- (Pratique) **Mis en place un environnement de développement complet**, avec ses outils ;
- (Pratique) Configuré l'application WordPress en mode *dev* via la fichier `wp-config.php` ;
- Compris la système de fichiers de WordPress et son architecture (core, themes, plugins, mu-plugins) ;
- (Pratique) Pris connaissance du schéma de la base de données de WordPress ;
- Pris connaissance des différentes API de WordPress ;
- (Pratique) Crée et activé notre premier thème ;
- (Pratique) Compris les fonctionnalités natives du *dashboard* de WordPress (admin) ;
- Vu le **fonctionnement général de WordPress** : cycle requête-réponse, "Loop Wordpress", modèle "MVC" et rôle theme/plugins/*template hierarchy/hooks*;
- Vu comment se documenter sur WordPress;
- Vu les données primitives de Wordpress (*post*, *taxonomy*, *term*, *user*, *metadata*) ;
- Vu quelques bonnes pratiques;

Développer en WordPress avec PHP

- Avoir les *bases nécessaires* pour développer des sites WordPress en PHP, *the WordPress way* ;
- Développer du *bon* code métier avec le *PHP moderne* (>PHP 5)

> Vers le module 2

Liens utiles

- Common APIs Handbook, le point d'entrée de la documentation web de référence de WordPress