

WORDPRESS DEVELOPER EXPERT — MANUEL COMPLET 2026

Formation professionnelle en autonomie | 8 semaines | 80 heures | 4 blocs

Objectif de sortie : construire, sécuriser, tester, optimiser et déployer un thème WordPress classique natif et un plugin métier associé, sans constructeur de page tiers.

Public et prérequis

HTML5/CSS3 solides ; JavaScript DOM/fetch ; bases PHP (fonctions/tableaux), SQL, Git et terminal. Un rappel préalable PHP est recommandé pour les personnes ne maîtrisant pas encore tableaux, objets et fonctions.

Méthode de travail

Lire chaque explication, reproduire le code, réaliser le TP sans consulter le corrigé, provoquer une erreur pour apprendre à la diagnostiquer, puis remplir la grille d'auto-évaluation. Les commandes sont indicatives : adapter l'URL, le préfixe SQL et les chemins au site local.

Projet fil rouge

Le thème Agency Starter gère la présentation. Le plugin Agency Portfolio Manager gère les réalisations, services, taxonomies, réglages et API. Le kit CI/CD prépare leur livraison. L'indépendance thème/plugin est un critère de réussite.

Règles de sécurité pédagogique

Ne jamais exécuter les manipulations de formation sur un site de production. Ne jamais modifier le Core. Sauvegarder les données avant migration. Ne pas exposer de scripts d'inspection sur le web. Tester toute opération d'écriture avec un compte de permissions adaptées. Les sorties d'API publiques ne doivent pas exposer les brouillons.

Attention à la validation

Ce document comprend des corrigés de référence et un protocole de validation. Les contrôles statiques peuvent être exécutés localement, mais une certification de compatibilité WordPress nécessite une installation réelle et des tests sur plusieurs configurations, qui ne peuvent être inventés.

Semaine 1 — Core, bootstrap, données et WP_Query

Bloc : Architecture WordPress | Durée : 10 heures | Niveau : développeur

Objectif : Diagnostiquer le cycle de vie d'une requête et manipuler la base avec les API natives.

Déroulé horaire et pédagogie

- 0 h à 1 h : prérequis, démonstration guidée et vocabulaire.
- 1 h à 4 h : explications approfondies et manipulations.

- 4 h à 6 h : lecture de code commentée et micro-exercices.
- 6 h à 9 h : TP autonome avec points de contrôle.
- 9 h à 10 h : correction, diagnostic, quiz et revue de code.

Cours approfondi — concepts à maîtriser

1. Fichiers du Core

wp-admin sert à l'administration, wp-includes fournit les API et wp-content regroupe les extensions. Il ne faut jamais personnaliser le Core : les mises à jour écraseraient les changements.

Cas réel : Dans un site client, identifier le thème actif et tous les plugins avant de commencer un correctif.

Erreur fréquente : Modifier wp-includes directement ; créer plutôt un plugin ou un thème enfant.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

2. Bootstrap et requête principale

index.php appelle wp-blog-header.php, qui initialise WordPress puis gère la requête principale. Le contexte de l'URL détermine les variables de requête et le template. Les routes REST et les écrans admin suivent des chemins spécifiques.

Cas réel : Tracer une URL /blog/exemple/ et écrire les étapes de résolution.

Erreur fréquente : Confondre l'URL réécrite avec un fichier physique sur le serveur.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

3. Base de données et métadonnées

wp_posts stocke de nombreux types de contenus ; wp_postmeta conserve des paires clé-valeur liées à un identifiant de post. Les termes sont reliés par wp_terms, wp_term_taxonomy et wp_term_relationships.

Cas réel : Créer un article, ajouter une métadonnée, retrouver son ID via WP-CLI.

Erreur fréquente : Supposer que toutes les tables portent le préfixe wp_ ; utiliser la configuration réelle.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

4. Requêtes et The Loop

WP_Query récupère des contenus en fonction de paramètres. have_posts et the_post mettent à jour le contexte global utilisé par les Template Tags. Après une requête secondaire qui exécute the_post, wp_reset_postdata rétablit le contexte.

Cas réel : Afficher trois articles publiés, leurs titres et permaliens.

Erreur fréquente : Oublier wp_reset_postdata et provoquer des titres incorrects après une boucle secondaire.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

5. Sécurité des sorties

L'échappement s'applique au contexte final : `esc_html` pour le texte, `esc_attr` pour les attributs, `esc_url` pour les URL. Un contenu de post rendu par `the_content` suit des filtres différents.

Cas réel : Afficher correctement le titre et les liens issus de la base.

Erreur fréquente : Appliquer la même fonction d'échappement à toutes les sorties.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

Lecture de code, ligne par ligne

Le fragment suivant s'exécute dans un contexte WordPress chargé. Adapter les chemins et les noms de classes au projet.

```
$query = new WP_Query( array(
    'post_type' => 'post',
    'post_status' => 'publish',
    'posts_per_page' => 3,
) );
while ( $query->have_posts() ) {
    $query->the_post();
    echo '<a href="' . esc_url( get_permalink() ) . '">' . esc_html( get_the_title() ) . '</a>';
}
wp_reset_postdata();
```

1. `new WP_Query` crée une requête secondaire avec critères explicites.
2. `post_status` exclut les brouillons des résultats.
3. `have_posts` vérifie qu'il reste des résultats ; `the_post` prépare chaque objet.
4. `esc_url` protège l'attribut `href` ; `esc_html` protège le titre affiché.
5. `wp_reset_postdata` rétablit le contexte de la requête principale.

Travaux pratiques guidés — consignes apprenant

Préparation : travailler sur une installation locale isolée ; activer l'affichage de logs hors production ; effectuer un commit avant toute modification.

1. Installer WordPress dans un environnement local avec une base dédiée. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
2. Activer `WP_DEBUG_LOG` hors production et contrôler `wp-content/debug.log`. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
3. Créer 5 articles : 3 publiés, 2 brouillons. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
4. Exécuter WP-CLI pour afficher ID, titre et statut. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
5. Écrire un script d'inspection qui liste trois articles et une métadonnée. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
6. Documenter la différence entre la requête principale et une requête secondaire. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).

Commandes de référence

```
wp core version
wp post list --post_type=post --fields=ID,post_title,post_status
wp post meta update 42 project_code DEMO-001
wp post meta get 42 project_code
```

Résultat attendu

La liste n'expose que les trois articles publiés. Le code est exécuté dans un contexte WordPress, sans accès public direct au script.

Corrigé commenté et méthode de diagnostic

- Étape 1 — correction attendue : Installer WordPress dans un environnement local avec une base dédiée. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 2 — correction attendue : Activer WP_DEBUG_LOG hors production et contrôler wp-content/debug.log. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 3 — correction attendue : Créer 5 articles : 3 publiés, 2 brouillons. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 4 — correction attendue : Exécuter WP-CLI pour afficher ID, titre et statut. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 5 — correction attendue : Écrire un script d'inspection qui liste trois articles et une métadonnée. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 6 — correction attendue : Documenter la différence entre la requête principale et une requête secondaire. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.

Erreurs fréquentes et solutions

- Symptôme : Aucun article renvoyé. Diagnostic/correction : Contrôler post_status, post_type, base de données et présence de publications.
- Symptôme : L'identifiant 42 n'existe pas. Diagnostic/correction : Remplacer 42 par l'ID retourné par wp post list.

Exercices progressifs et corrigés

Niveau 1 — Reproduire

Reproduire le fragment de code sans modifier sa logique ; expliquer les cinq points commentés ci-dessus.

Corrigé : la sortie et la structure doivent correspondre à la démonstration ; toute erreur PHP doit être éliminée avant la suite.

Niveau 2 — Adapter

Adapter les noms, données, paramètres ou styles à un second cas métier sans copier le code de manière incohérente.

Corrigé : documenter précisément les modifications, la raison et les résultats fonctionnels.

Niveau 3 — Durcir

Créer un cas d'échec volontaire : absence de contenu, refus d'autorisation, mauvais format de données ou ressource indisponible.

Corrigé : aucune erreur fatale ; l'échec doit être géré, visible dans les logs appropriés et couvert par un test manuel ou automatisé.

Quiz de fin de module — questions et réponses

1. Question : Où réside un thème ? Réponse attendue : wp-content/themes.
2. Question : Que fait wp_reset_postdata ? Réponse attendue : Restaure le contexte global après une boucle secondaire.
3. Question : Quelle table contient les métadonnées des posts ? Réponse attendue : wp_postmeta.

Grille de validation pratique — 20 points

- 4 pts : installation reproductible, fichiers et configuration corrects.
- 6 pts : comportement fonctionnel démontré en condition réelle.
- 4 pts : sécurité et gestion des erreurs appropriées.
- 3 pts : tests et preuves observables.
- 3 pts : code lisible, historique Git et documentation.

Seuil : 14/20 minimum ; toute vulnérabilité critique non corrigée bloque la validation.

Semaine 2 — PHP orienté objet, Hooks, Composer, qualité

Bloc : Architecture WordPress | Durée : 10 heures | Niveau : développeur

Objectif : Structurer un socle PHP modulaire et mettre en place une discipline de qualité.

Déroulé horaire et pédagogie

- 0 h à 1 h : prérequis, démonstration guidée et vocabulaire.
- 1 h à 4 h : explications approfondies et manipulations.
- 4 h à 6 h : lecture de code commentée et micro-exercices.
- 6 h à 9 h : TP autonome avec points de contrôle.
- 9 h à 10 h : correction, diagnostic, quiz et revue de code.

Cours approfondi — concepts à maîtriser

1. Responsabilité unique

Une classe a un rôle cohérent. Séparer le chargeur de ressources, les routes REST et l'administration réduit les effets de bord et aide aux tests.

Cas réel : Créer AssetManager et ContentManager séparément.

Erreur fréquente : Une classe Application contenant toutes les fonctionnalités finit par devenir difficile à maintenir.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

2. Namespace et autoload PSR-4

Un namespace évite les collisions ; Composer associe un préfixe de namespace aux dossiers physiques. Un autoload permet de charger les classes sans `require_once` répété.

Cas réel : Mapper `Agency\` à `src/` et instancier `Agency\Frontend\AssetManager`.

Erreur fréquente : Namespace et chemin fichier incompatibles : Composer ne retrouve pas la classe.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

3. Actions et filtres

Une action exécute un callback lorsque WordPress déclenche un événement ; un filtre prend une valeur, éventuellement la transforme, puis retourne le résultat. Priorités les plus basses exécutées en premier.

Cas réel : Ajouter une classe `body_class` et enregistrer un Hook `after_setup_theme`.

Erreur fréquente : Oublier `return` dans un filtre, ce qui casse la valeur propagée.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

4. Chargement des assets

`wp_enqueue_style` et `wp_enqueue_script` gèrent dépendances et versions. Un appel conditionnel limite l'envoi de scripts inutiles et peut réduire le temps d'exécution côté client.

Cas réel : Charger `project.js` seulement pour `is_singular(agency_project)`.

Erreur fréquente : Insérer manuellement des balises script, contourner le système des dépendances.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

5. Qualité et tests

PHPCS et WPCS détectent des problèmes de conventions et certaines mauvaises pratiques ; PHPUnit exécute des assertions. La CI applique les contrôles à chaque changement. Cela ne prouve pas l'absence de bugs.

Cas réel : Écrire un test unitaire pour une classe indépendante de WordPress.

Erreur fréquente : Confondre analyse statique, test unitaire et test d'intégration sur WordPress.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

Lecture de code, ligne par ligne

Le fragment suivant s'exécute dans un contexte WordPress chargé. Adapter les chemins et les noms de classes au projet.

```
namespace Agency\Frontend;
class AssetManager {
    public function register(): void {
        add_action( 'wp_enqueue_scripts', array( $this, 'enqueue' ) );
    }
    public function enqueue(): void {
        if ( is_singular( 'agency_project' ) ) {
```

```

        wp_enqueue_script( 'agency-project', get_theme_file_uri( 'assets/js/project.js' ),
        array
        )
    }
}
}

```

1. Le namespace isole le nom complet de la classe.
2. register lie la méthode enqueue au Hook frontend.
3. is_singular teste le contexte de la requête.
4. wp_enqueue_script enregistre la ressource seulement pour les réalisations.
5. La version facilite l'invalidation du cache lors des livraisons.

Travaux pratiques guidés — consignes apprenant

Préparation : travailler sur une installation locale isolée ; activer l'affichage de logs hors production ; effectuer un commit avant toute modification.

1. Créer le fichier composer.json et la section autoload PSR-4. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
2. Créer une classe AssetManager sous src/Frontend/. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
3. Initialiser l'application depuis functions.php et composer vendor/autoload.php. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
4. Installer PHPCS/WPCS dans les dépendances de développement. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
5. Créer un test PHPUnit sur une classe de formatage pure. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
6. Pousser les changements vers un dépôt Git de formation. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).

Commandes de référence

```

composer install
composer dump-autoload
./vendor/bin/phpcs
./vendor/bin/phpunit --bootstrap vendor/autoload.php tests/Unit

```

Résultat attendu

Le chargement automatique fonctionne, l'asset est chargé seulement sur la page attendue, les tests s'exécutent.

Corrigé commenté et méthode de diagnostic

- Étape 1 — correction attendue : Créer le fichier composer.json et la section autoload PSR-4. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 2 — correction attendue : Créer une classe AssetManager sous src/Frontend/. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 3 — correction attendue : Initialiser l'application depuis functions.php et composer vendor/autoload.php. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.

- Étape 4 — correction attendue : Installer PHPCS/WPCS dans les dépendances de développement. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 5 — correction attendue : Créer un test PHPUnit sur une classe de formatage pure. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 6 — correction attendue : Pousser les changements vers un dépôt Git de formation. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.

Erreurs fréquentes et solutions

- Symptôme : Class not found. Diagnostic/correction : Contrôler le préfixe PSR-4, la casse du fichier et composer dump-autoload.
- Symptôme : Le JS manque. Diagnostic/correction : Vérifier wp_head/wp_footer dans le thème et le Hook wp_enqueue_scripts.

Exercices progressifs et corrigés

Niveau 1 — Reproduire

Reproduire le fragment de code sans modifier sa logique ; expliquer les cinq points commentés ci-dessus.

Corrigé : la sortie et la structure doivent correspondre à la démonstration ; toute erreur PHP doit être éliminée avant la suite.

Niveau 2 — Adapter

Adapter les noms, données, paramètres ou styles à un second cas métier sans copier le code de manière incohérente.

Corrigé : documenter précisément les modifications, la raison et les résultats fonctionnels.

Niveau 3 — Durcir

Créer un cas d'échec volontaire : absence de contenu, refus d'autorisation, mauvais format de données ou ressource indisponible.

Corrigé : aucune erreur fatale ; l'échec doit être géré, visible dans les logs appropriés et couvert par un test manuel ou automatisé.

Quiz de fin de module — questions et réponses

1. Question : Que retourne un filtre ? Réponse attendue : La valeur modifiée ou inchangée.
2. Question : À quoi sert PSR-4 ? Réponse attendue : Associer namespaces et chemins.
3. Question : Un PHPCS vert garantit-il un site sans faille ? Réponse attendue : Non.

Grille de validation pratique — 20 points

- 4 pts : installation reproductible, fichiers et configuration corrects.
- 6 pts : comportement fonctionnel démontré en condition réelle.
- 4 pts : sécurité et gestion des erreurs appropriées.
- 3 pts : tests et preuves observables.
- 3 pts : code lisible, historique Git et documentation.

Seuil : 14/20 minimum ; toute vulnérabilité critique non corrigée bloque la validation.

Semaine 3 — Structure, header, footer, The Loop, responsive

Bloc : Thèmes From Scratch | Durée : 10 heures | Niveau : développeur

Objectif : Construire un thème classique activable et accessible sans constructeur tiers.

Déroulé horaire et pédagogie

- 0 h à 1 h : prérequis, démonstration guidée et vocabulaire.
- 1 h à 4 h : explications approfondies et manipulations.
- 4 h à 6 h : lecture de code commentée et micro-exercices.
- 6 h à 9 h : TP autonome avec points de contrôle.
- 9 h à 10 h : correction, diagnostic, quiz et revue de code.

Cours approfondi — concepts à maîtriser

1. Thème classique minimum

Un thème classique nécessite un style.css avec en-tête de thème et un index.php. functions.php est facultatif mais utile pour enregistrer les supports et ressources.

Cas réel : Activer un thème minimal depuis Apparence.

Erreur fréquente : Copier du contenu depuis le Core plutôt que créer un dossier de thème.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

2. Template Hierarchy

WordPress choisit le template le plus spécifique disponible puis redescend vers index.php. front-page.php peut servir à la page d'accueil, home.php à l'index des articles.

Cas réel : Comparer accueil statique et index du blog.

Erreur fréquente : Confondre home.php avec front-page.php.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

3. En-tête et pied de page

wp_head, wp_body_open et wp_footer sont des points d'extension majeurs. language_attributes, bloginfo(charset) et body_class complètent le HTML.

Cas réel : Ajouter des fichiers header.php et footer.php valides.

Erreur fréquente : Oublier wp_footer et casser des scripts.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

4. Menus et images

register_nav_menus déclare des emplacements ; wp_nav_menu affiche un menu administrable.
add_theme_support(post-thumbnails) active les images mises en avant.

Cas réel : Configurer navigation, logo et image d'article.

Erreur fréquente : Écrire des liens statiques non administrables.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

5. Accessibilité responsive

HTML sémantique, lien d'évitement, contrastes, focus visible et navigation au clavier sont nécessaires. Une grille responsive doit éviter les débordements latéraux.

Cas réel : Tester au clavier et à 320px, 768px et 1440px.

Erreur fréquente : Faire disparaître le focus clavier avec outline:none sans alternative.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

Lecture de code, ligne par ligne

Le fragment suivant s'exécute dans un contexte WordPress chargé. Adapter les chemins et les noms de classes au projet.

```
<?php get_header(); ?>
<main id="main-content">
<?php if ( have_posts() ) : ?>
    <?php while ( have_posts() ) : the_post(); ?>
        <article <?php post_class(); ?>>
            <h1><?php the_title(); ?></h1>
            <?php the_content(); ?>
        </article>
    <?php endwhile; ?>
<?php endif; ?>
</main>
<?php get_footer(); ?>
```

1. get_header et get_footer incluent les éléments globaux.
2. main fournit un point principal repérable par les aides techniques.
3. have_posts et the_post parcourent la requête principale.
4. post_class fournit des classes utiles au style.
5. the_content applique les filtres de rendu WordPress.

Travaux pratiques guidés — consignes apprenant

Préparation : travailler sur une installation locale isolée ; activer l'affichage de logs hors production ; effectuer un commit avant toute modification.

1. Créer style.css et index.php pour rendre le thème visible. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).

2. Configurer les supports post-thumbnails, title-tag et menu primary. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
3. Créer header.php, footer.php, single.php, page.php et front-page.php. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
4. Créer un composant article-card dans template-parts/. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
5. Développer un CSS mobile-first et tester l'accès au menu au clavier. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
6. Comparer rendu articles, pages et page d'accueil. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).

Commandes de référence

```
wp theme list
wp theme activate agency-starter
wp option get show_on_front
```

Résultat attendu

Les contenus sont modifiables dans WordPress ; le menu et les liens fonctionnent sur mobile et au clavier.

Corrigé commenté et méthode de diagnostic

- Étape 1 — correction attendue : Créer style.css et index.php pour rendre le thème visible. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 2 — correction attendue : Configurer les supports post-thumbnails, title-tag et menu primary. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 3 — correction attendue : Créer header.php, footer.php, single.php, page.php et front-page.php. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 4 — correction attendue : Créer un composant article-card dans template-parts/. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 5 — correction attendue : Développer un CSS mobile-first et tester l'accès au menu au clavier. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 6 — correction attendue : Comparer rendu articles, pages et page d'accueil. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.

Erreurs fréquentes et solutions

- Symptôme : Thème absent. Diagnostic/correction : Contrôler l'en-tête Theme Name dans style.css et index.php.
- Symptôme : Menu invisible. Diagnostic/correction : Enregistrer le bon emplacement et affecter un menu dans Appearance.

Exercices progressifs et corrigés

Niveau 1 — Reproduire

Reproduire le fragment de code sans modifier sa logique ; expliquer les cinq points commentés ci-dessus.

Corrigé : la sortie et la structure doivent correspondre à la démonstration ; toute erreur PHP doit être éliminée avant la suite.

Niveau 2 — Adapter

Adapter les noms, données, paramètres ou styles à un second cas métier sans copier le code de manière incohérente.

Corrigé : documenter précisément les modifications, la raison et les résultats fonctionnels.

Niveau 3 — Durcir

Créer un cas d'échec volontaire : absence de contenu, refus d'autorisation, mauvais format de données ou ressource indisponible.

Corrigé : aucune erreur fatale ; l'échec doit être géré, visible dans les logs appropriés et couvert par un test manuel ou automatisé.

Quiz de fin de module — questions et réponses

1. Question : Quel est le template fallback classique ? Réponse attendue : index.php.
2. Question : Quel Hook doit exister dans head ? Réponse attendue : wp_head().
3. Question : À quoi sert un skip-link ? Réponse attendue : Accéder directement au contenu au clavier.

Grille de validation pratique — 20 points

- 4 pts : installation reproductible, fichiers et configuration corrects.
- 6 pts : comportement fonctionnel démontré en condition réelle.
- 4 pts : sécurité et gestion des erreurs appropriées.
- 3 pts : tests et preuves observables.
- 3 pts : code lisible, historique Git et documentation.

Seuil : 14/20 minimum ; toute vulnérabilité critique non corrigée bloque la validation.

Semaine 4 — Archives, pagination, taxonomies, Gutenberg

Bloc : Thèmes From Scratch | Durée : 10 heures | Niveau : développeur

Objectif : Créer des templates réutilisables, une pagination fiable et maîtriser theme.json.

Déroulé horaire et pédagogie

- 0 h à 1 h : prérequis, démonstration guidée et vocabulaire.
- 1 h à 4 h : explications approfondies et manipulations.
- 4 h à 6 h : lecture de code commentée et micro-exercices.
- 6 h à 9 h : TP autonome avec points de contrôle.
- 9 h à 10 h : correction, diagnostic, quiz et revue de code.

Cours approfondi — concepts à maîtriser

1. Archives et query principale

Pour l'archive des posts, les paramètres de la requête principale devraient être modifiés via `pre_get_posts` avec des `is_admin` et `is_main_query` ; créer une requête secondaire ne doit pas se faire sans raison.

Cas réel : Afficher une archive paginée de réalisations.

Erreur fréquente : Remplacer la requête principale par `query_posts`.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

2. Pagination

`the_posts_pagination` utilise la requête principale. Sur `WP_Query` secondaire, transmettre `paged` et `max_num_pages` à `paginate_links` ; ne pas mettre `no_found_rows` à `true`.

Cas réel : Naviguer de page 1 à page 2.

Erreur fréquente : Avoir des liens de pagination identiques ou vides.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

3. Taxonomies

`single-{post_type}.php`, `archive-{post_type}.php` et `taxonomy-{taxonomy}.php` ciblent les données enregistrées dans un plugin. Les taxonomies classent des contenus ; un thème gère leur affichage.

Cas réel : Afficher `archive-agency_project` et `taxonomy-agency_sector`.

Erreur fréquente : Déclarer le CPT uniquement dans le thème : le contenu disparaît fonctionnellement au changement de thème.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

4. Template Parts

`get_template_part` évite la duplication. Extraire les composants `card`, `breadcrumb` et `empty-state` facilite les corrections globales.

Cas réel : Réutiliser `content-card.php` dans l'accueil et l'archive.

Erreur fréquente : Copier-coller dix variantes d'un même composant.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

5. Gutenberg et theme.json

`theme.json` version 3 centralise notamment `palette` et `dimensions` pour les versions WordPress qui la prennent en charge. Un thème classique peut contenir `theme.json` ; un Block Theme utilise des templates HTML de blocs.

Cas réel : Déclarer `palette` et `layout`, puis créer un contenu avec des blocs.

Erreur fréquente : Penser que `theme.json` transforme automatiquement un thème classique en Block Theme.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

Lecture de code, ligne par ligne

Le fragment suivant s'exécute dans un contexte WordPress chargé. Adapter les chemins et les noms de classes au projet.

```
add_action( 'pre_get_posts', function ( $query ) {
    if ( is_admin() || ! $query->is_main_query() ) {
        return;
    }
    if ( $query->is_post_type_archive( 'agency_project' ) ) {
        $query->set( 'posts_per_page', 6 );
    }
} );
```

1. `pre_get_posts` permet de modifier les paramètres avant l'exécution SQL.
2. `is_admin` exclut les requêtes d'administration.
3. `is_main_query` évite de modifier les boucles secondaires.
4. `is_post_type_archive` limite l'effet aux réalisations.
5. `set` ajuste la pagination à six contenus.

Travaux pratiques guidés — consignes apprenant

Préparation : travailler sur une installation locale isolée ; activer l'affichage de logs hors production ; effectuer un commit avant toute modification.

1. Créer le template `archive-agency_project.php`. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
2. Ajouter au moins 13 réalisations pour tester trois pages. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
3. Développer les Template Parts et l'état sans contenu. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
4. Créer un template de taxonomie `agency_sector`. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
5. Ajouter `theme.json` et contrôler les couleurs de l'éditeur. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
6. Tester recherche sans résultats et URL 404. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).

Commandes de référence

```
wp rewrite flush
wp post list --post_type=agency_project --fields=ID,post_title,post_status
```

Résultat attendu

Archives et pagination conservent le bon contexte, sans duplicate content non souhaité ni erreurs 404.

Corrigé commenté et méthode de diagnostic

- Étape 1 — correction attendue : Créer le template `archive-agency_project.php`. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 2 — correction attendue : Ajouter au moins 13 réalisations pour tester trois pages. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 3 — correction attendue : Développer les Template Parts et l'état sans contenu. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 4 — correction attendue : Créer un template de taxonomie `agency_sector`. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 5 — correction attendue : Ajouter `theme.json` et contrôler les couleurs de l'éditeur. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 6 — correction attendue : Tester recherche sans résultats et URL 404. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.

Erreurs fréquentes et solutions

- Symptôme : Archive renvoie 404. Diagnostic/correction : Vérifier CPT `has_archive`, slug et régénérer les permaliens UNE fois.
- Symptôme : Page 2 répète page 1. Diagnostic/correction : Vérifier `paged` et la nature principale/secondaire de la query.

Exercices progressifs et corrigés

Niveau 1 — Reproduire

Reproduire le fragment de code sans modifier sa logique ; expliquer les cinq points commentés ci-dessus.

Corrigé : la sortie et la structure doivent correspondre à la démonstration ; toute erreur PHP doit être éliminée avant la suite.

Niveau 2 — Adapter

Adapter les noms, données, paramètres ou styles à un second cas métier sans copier le code de manière incohérente.

Corrigé : documenter précisément les modifications, la raison et les résultats fonctionnels.

Niveau 3 — Durcir

Créer un cas d'échec volontaire : absence de contenu, refus d'autorisation, mauvais format de données ou ressource indisponible.

Corrigé : aucune erreur fatale ; l'échec doit être géré, visible dans les logs appropriés et couvert par un test manuel ou automatisé.

Quiz de fin de module — questions et réponses

1. Question : Où placer un CPT métier ? Réponse attendue : Dans un plugin.
2. Question : Quelle fonction pagine la requête principale ? Réponse attendue : `the_posts_pagination()`.
3. Question : `theme.json` fait-il un Block Theme ? Réponse attendue : Non.

Grille de validation pratique — 20 points

- 4 pts : installation reproductible, fichiers et configuration corrects.
- 6 pts : comportement fonctionnel démontré en condition réelle.
- 4 pts : sécurité et gestion des erreurs appropriées.
- 3 pts : tests et preuves observables.
- 3 pts : code lisible, historique Git et documentation.

Seuil : 14/20 minimum ; toute vulnérabilité critique non corrigée bloque la validation.

Semaine 5 — Plugin, hooks, CPT, taxonomies, Settings API

Bloc : Plugins et sécurité | Durée : 10 heures | Niveau : développeur

Objectif : Développer un plugin métier indépendant du thème et protégé.

Déroulé horaire et pédagogie

- 0 h à 1 h : prérequis, démonstration guidée et vocabulaire.
- 1 h à 4 h : explications approfondies et manipulations.
- 4 h à 6 h : lecture de code commentée et micro-exercices.
- 6 h à 9 h : TP autonome avec points de contrôle.
- 9 h à 10 h : correction, diagnostic, quiz et revue de code.

Cours approfondi — concepts à maîtriser

1. Cycle de vie du plugin

En-tête Plugin Name, Hooks d'activation et de désactivation, et `uninstall.php` ont des responsabilités distinctes. Les fichiers de contenus restent normalement en base lors de la désactivation.

Cas réel : Activer et désactiver sans perdre les réalisations.

Erreur fréquente : Effacer les données à chaque désactivation.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

2. CPT et taxonomie

`register_post_type` sur `init` déclare `agency_project` et `agency_service`. `register_taxonomy` classe les projets. `show_in_rest` rend l'éditeur et l'API disponibles selon configuration.

Cas réel : Créer les deux menus administrateur et des termes Secteurs.

Erreur fréquente : Déclarer les objets après le moment utile pour WordPress.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

3. Métadonnées

`register_post_meta` définit le type, la sanitization, `show_in_rest` et `auth_callback` ; `custom-fields` doit être supporté par le CPT pour une exposition REST cohérente.

Cas réel : Ajouter année et client à une réalisation.

Erreur fréquente : Exposer en écriture des champs privés sans autorisation appropriée.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

4. Settings API

`register_setting` déclare une option et sa validation ; les `fields/sections` construisent un formulaire. Les accès à la page sont protégés par une capacité.

Cas réel : Créer un réglage entre 1 et 24.

Erreur fréquente : Utiliser `$_POST` brut dans `update_option`.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

5. Sécurité

Un nonce défend une intention contre CSRF, mais n'est ni authentification ni autorisation. `current_user_can` contrôle les droits ; l'entrée doit être validée et sortie échappée.

Cas réel : Tester accès anonyme et auteur non autorisé.

Erreur fréquente : Autoriser une modification parce que le nonce est valide, sans vérifier `edit_post`.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

Lecture de code, ligne par ligne

Le fragment suivant s'exécute dans un contexte WordPress chargé. Adapter les chemins et les noms de classes au projet.

```
register_post_type( 'agency_project', array(
    'public' => true,
    'show_in_rest' => true,
    'has_archive' => true,
    'supports' => array( 'title', 'editor', 'thumbnail', 'custom-fields' ),
) );
```

1. `agency_project` est une clé technique stable et courte.
2. `public` autorise la présence publique des contenus selon les autres réglages.
3. `show_in_rest` permet l'exposition native aux routes adaptées.
4. `has_archive` active une archive pour le type.
5. `custom-fields` permet l'utilisation du système de métadonnées exposé au REST.

Travaux pratiques guidés — consignes apprenant

Préparation : travailler sur une installation locale isolée ; activer l’affichage de logs hors production ; effectuer un commit avant toute modification.

1. Installer le plugin Agency Portfolio Manager dans staging. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
2. Créer CPT Réalisations et Services. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
3. Ajouter taxonomy Secteurs et des métadonnées sécurisées. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
4. Créer une page Settings API et enregistrer un nombre compris entre 1 et 24. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
5. Tester un utilisateur qui ne peut pas enregistrer le réglage. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
6. Désactiver/réactiver et vérifier conservation des contenus. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).

Commandes de référence

```
wp plugin activate agency-portfolio-manager
wp post-type get agency_project
wp option get agency_pm_items_per_page
```

Résultat attendu

Les données métier survivent au changement de thème et les accès administratifs non autorisés sont refusés.

Corrigé commenté et méthode de diagnostic

- Étape 1 — correction attendue : Installer le plugin Agency Portfolio Manager dans staging. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 2 — correction attendue : Créer CPT Réalisations et Services. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 3 — correction attendue : Ajouter taxonomy Secteurs et des métadonnées sécurisées. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 4 — correction attendue : Créer une page Settings API et enregistrer un nombre compris entre 1 et 24. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 5 — correction attendue : Tester un utilisateur qui ne peut pas enregistrer le réglage. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 6 — correction attendue : Désactiver/réactiver et vérifier conservation des contenus. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.

Erreurs fréquentes et solutions

- Symptôme : Champ custom invisible via REST. Diagnostic/correction : Vérifier `show_in_rest` et supports `custom-fields`.
- Symptôme : Les CPT disparaissent. Diagnostic/correction : S’assurer que le plugin est actif et que les Hooks init se déclenchent.

Exercices progressifs et corrigés

Niveau 1 — Reproduire

Reproduire le fragment de code sans modifier sa logique ; expliquer les cinq points commentés ci-dessus.

Corrigé : la sortie et la structure doivent correspondre à la démonstration ; toute erreur PHP doit être éliminée avant la suite.

Niveau 2 — Adapter

Adapter les noms, données, paramètres ou styles à un second cas métier sans copier le code de manière incohérente.

Corrigé : documenter précisément les modifications, la raison et les résultats fonctionnels.

Niveau 3 — Durcir

Créer un cas d'échec volontaire : absence de contenu, refus d'autorisation, mauvais format de données ou ressource indisponible.

Corrigé : aucune erreur fatale ; l'échec doit être géré, visible dans les logs appropriés et couvert par un test manuel ou automatisé.

Quiz de fin de module — questions et réponses

1. Question : Nonce = autorisation ? Réponse attendue : Non.
2. Question : Où déclare-t-on un CPT ? Réponse attendue : `register_post_type()` sur `init`.
3. Question : Doit-on supprimer au `deactivate` ? Réponse attendue : Non, généralement pas.

Grille de validation pratique — 20 points

- 4 pts : installation reproductible, fichiers et configuration corrects.
- 6 pts : comportement fonctionnel démontré en condition réelle.
- 4 pts : sécurité et gestion des erreurs appropriées.
- 3 pts : tests et preuves observables.
- 3 pts : code lisible, historique Git et documentation.

Seuil : 14/20 minimum ; toute vulnérabilité critique non corrigée bloque la validation.

Semaine 6 — REST CRUD, JavaScript, permissions et PHPUnit

Bloc : Plugins et sécurité | Durée : 10 heures | Niveau : développeur

Objectif : Créer, lire, modifier et supprimer logiquement des contenus via des endpoints testés.

Déroulé horaire et pédagogie

- 0 h à 1 h : prérequis, démonstration guidée et vocabulaire.
- 1 h à 4 h : explications approfondies et manipulations.
- 4 h à 6 h : lecture de code commentée et micro-exercices.

- 6 h à 9 h : TP autonome avec points de contrôle.
- 9 h à 10 h : correction, diagnostic, quiz et revue de code.

Cours approfondi — concepts à maîtriser

1. REST et verbes HTTP

Les ressources sont exposées à des routes versionnées. GET lit les données ; POST crée, PATCH modifie partiellement, DELETE supprime ou met à la corbeille selon le contrat.

Cas réel : Tester GET /agency/v1/projects.

Erreur fréquente : Ne pas documenter la signification réelle de DELETE.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

2. Contrôleurs et arguments

register_rest_route est appelé sur rest_api_init ; des paramètres args déclarent les types, la sanitization et la validation. WP_REST_Response et WP_Error matérialisent résultats et erreurs.

Cas réel : Retourner 400 pour une entrée invalide.

Erreur fréquente : Ne pas traiter les cas inexistantes et produire un 500.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

3. Permission callback

Une route publique peut renvoyer seulement les contenus publiés ; les écritures requièrent une permission précise, et un objet existant requiert edit_post ou delete_post sur son ID.

Cas réel : Tenter une modification en anonyme et en utilisateur autorisé.

Erreur fréquente : Réutiliser __return_true sur une route d'écriture.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

4. Frontend avec fetch

Le navigateur demande du JSON. Pour éviter des injections HTML, afficher le texte avec textContent et des éléments DOM créés explicitement. Une base URL REST est injectée via mécanisme WordPress.

Cas réel : Construire un tableau filtrable de projets.

Erreur fréquente : Construire innerHTML avec des valeurs externes non filtrées.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

5. Tests WordPress

Les tests unitaires isolés vérifient les fonctions pures ; les tests d'intégration chargent WP et les plugins, fabriquent des utilisateurs et inspectent les réponses REST. Il faut tester succès et refus.

Cas réel : Vérifier GET 200 et refus POST anonyme.

Erreur fréquente : Dire que des tests PHPUnit sont réussis alors que seul php -l a été exécuté.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

Lecture de code, ligne par ligne

Le fragment suivant s'exécute dans un contexte WordPress chargé. Adapter les chemins et les noms de classes au projet.

```
register_rest_route( 'agency/v1', '/projects', array(
    'methods' => WP_REST_Server::READABLE,
    'callback' => 'agency_list_projects',
    'permission_callback' => '__return_true',
) );
```

1. Le namespace agency/v1 fixe la version du contrat.
2. projects correspond au chemin de la ressource.
3. READABLE correspond à un accès de lecture.
4. callback exécute la fonction de récupération des contenus publiés.
5. __return_true est acceptable ici seulement si le résultat est réellement public.

Travaux pratiques guidés — consignes apprenant

Préparation : travailler sur une installation locale isolée ; activer l'affichage de logs hors production ; effectuer un commit avant toute modification.

1. Ajouter route GET publique ne renvoyant que les projets publiés. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
2. Déclarer arguments per_page bornés entre 1 et 24. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
3. Créer une route POST qui crée un brouillon pour un utilisateur autorisé. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
4. Ajouter PATCH/DELETE et vérifier edit_post/delete_post objet par objet. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
5. Créer des tests REST de lecture anonyme, écriture anonyme et écriture autorisée. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
6. Afficher les données via fetch dans un composant sans innerHTML non fiable. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).

Commandes de référence

```
curl -i http://localhost:8080/wp-json/agency/v1/projects
wp user list --fields=ID,user_login,roles
```

Résultat attendu

GET ne divulgue pas de brouillon ; POST anonyme est refusé ; une écriture autorisée crée un brouillon.

Corrigé commenté et méthode de diagnostic

- Étape 1 — correction attendue : Ajouter route GET publique ne renvoyant que les projets publiés. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 2 — correction attendue : Déclarer arguments per_page bornés entre 1 et 24. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 3 — correction attendue : Créer une route POST qui crée un brouillon pour un utilisateur autorisé. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 4 — correction attendue : Ajouter PATCH/DELETE et vérifier edit_post/delete_post objet par objet. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 5 — correction attendue : Créer des tests REST de lecture anonyme, écriture anonyme et écriture autorisée. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 6 — correction attendue : Afficher les données via fetch dans un composant sans innerHTML non fiable. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.

Erreurs fréquentes et solutions

- Symptôme : rest_no_route. Diagnostic/correction : Vérifier rest_api_init, permaliens et méthode HTTP.
- Symptôme : POST retourne 403 pour admin. Diagnostic/correction : Vérifier authentication REST et nonce/capacités, pas uniquement rôle nominal.

Exercices progressifs et corrigés

Niveau 1 — Reproduire

Reproduire le fragment de code sans modifier sa logique ; expliquer les cinq points commentés ci-dessus.

Corrigé : la sortie et la structure doivent correspondre à la démonstration ; toute erreur PHP doit être éliminée avant la suite.

Niveau 2 — Adapter

Adapter les noms, données, paramètres ou styles à un second cas métier sans copier le code de manière incohérente.

Corrigé : documenter précisément les modifications, la raison et les résultats fonctionnels.

Niveau 3 — Durcir

Créer un cas d'échec volontaire : absence de contenu, refus d'autorisation, mauvais format de données ou ressource indisponible.

Corrigé : aucune erreur fatale ; l'échec doit être géré, visible dans les logs appropriés et couvert par un test manuel ou automatisé.

Quiz de fin de module — questions et réponses

1. Question : Où enregistrer la route ? Réponse attendue : rest_api_init.
2. Question : Une route POST peut-elle être publique ? Réponse attendue : Pas pour une écriture métier sans contrôle.
3. Question : Quel test prouve le refus d'accès ? Réponse attendue : Un test d'intégration avec utilisateur non autorisé.

Grille de validation pratique — 20 points

- 4 pts : installation reproductible, fichiers et configuration corrects.
- 6 pts : comportement fonctionnel démontré en condition réelle.
- 4 pts : sécurité et gestion des erreurs appropriées.
- 3 pts : tests et preuves observables.
- 3 pts : code lisible, historique Git et documentation.

Seuil : 14/20 minimum ; toute vulnérabilité critique non corrigée bloque la validation.

Semaine 7 — Crawl, indexabilité, canonicals et JSON-LD

Bloc : SEO technique | Durée : 10 heures | Niveau : développeur

Objectif : Auditer les signaux techniques SEO et publier des données structurées non dupliquées.

Déroulé horaire et pédagogie

- 0 h à 1 h : prérequis, démonstration guidée et vocabulaire.
- 1 h à 4 h : explications approfondies et manipulations.
- 4 h à 6 h : lecture de code commentée et micro-exercices.
- 6 h à 9 h : TP autonome avec points de contrôle.
- 9 h à 10 h : correction, diagnostic, quiz et revue de code.

Cours approfondi — concepts à maîtriser

1. Crawl / index / classement

Exploration = découverte et récupération ; indexation = sélection des pages ; classement = réponse à une requête.
Un statut 200 ne garantit ni indexation ni trafic.

Cas réel : Inspecter dix URL du site.

Erreur fréquente : Assimiler présence en sitemap à une garantie d'indexation.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

2. HTTP et canonicals

Gérer 200, 301, 302, 404, 410 et 5xx. Une canonical est un signal de préférence, non une redirection ; une URL peut encore être choisie autrement par Google.

Cas réel : Vérifier les en-têtes et canonical sur une archive paginée.

Erreur fréquente : Ajouter deux balises canonical concurrentes.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

3. Robots et sitemap

robots.txt régule le crawl ; noindex est une directive d'indexation qui doit être accessible à un robot. Un sitemap propose les URL canoniques que l'on souhaite faire découvrir.

Cas réel : Comparer robots.txt, wp-sitemap.xml, et meta robots.

Erreur fréquente : Bloquer en robots.txt une page dont on exige la lecture du noindex.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

4. HTML et métadonnées

Title pertinent, structure de titres cohérente, liens explicites, textes alternatifs adaptés et structure HTML claire facilitent lecture et accessibilité.

Cas réel : Écrire des titres uniques pour 10 URL.

Erreur fréquente : Abuser du keyword stuffing ou multiplier les H1 sans logique éditoriale.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

5. Schema.org JSON-LD

Les balisages BlogPosting, Organization ou BreadcrumbList doivent refléter le contenu réel. Éviter les balisages concurrents du thème et d'un plugin SEO. Les rich results ne sont jamais garantis.

Cas réel : Ajouter BlogPosting sur un article puis valider la sortie JSON-LD.

Erreur fréquente : Ajouter un LocalBusiness fictif ou des avis non présents.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

Lecture de code, ligne par ligne

Le fragment suivant s'exécute dans un contexte WordPress chargé. Adapter les chemins et les noms de classes au projet.

```
$schema = array(
    '@context' => 'https://schema.org',
    '@type' => 'BlogPosting',
    'headline' => get_the_title( $post ),
    'datePublished' => get_the_date( 'c', $post ),
    'url' => get_permalink( $post ),
);
echo '<script type="application/ld+json">';
echo wp_json_encode( $schema, JSON_HEX_TAG | JSON_HEX_AMP );
echo '</script>';
```

1. @context fixe le vocabulaire utilisé.
2. @type décrit la nature réelle du contenu.
3. headline est récupéré depuis la publication courante.
4. datePublished utilise un format ISO 8601 via get_the_date(c).
5. wp_json_encode sérialise les données WordPress de façon appropriée.

Travaux pratiques guidés — consignes apprenant

Préparation : travailler sur une installation locale isolée ; activer l’affichage de logs hors production ; effectuer un commit avant toute modification.

1. Créer un inventaire de 10 URL et mesurer statut/canonical. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
2. Contrôler titres, méta description, H1, liens et alternatifs. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
3. Vérifier la cohérence sitemap/robots/noindex. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
4. Activer le JSON-LD uniquement s’il n’est pas déjà émis par le SEO plugin. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
5. Valider BlogPosting et BreadcrumbList selon pertinence. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
6. Rédiger un journal des anomalies classées par impact. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).

Commandes de référence

```
curl -I https://example.com/  
curl -s https://example.com/wp-sitemap.xml | head  
curl -s https://example.com/robots.txt
```

Résultat attendu

Toutes les anomalies ont preuve, gravité, correction et test de régression ; aucune promesse de classement.

Corrigé commenté et méthode de diagnostic

- Étape 1 — correction attendue : Créer un inventaire de 10 URL et mesurer statut/canonical. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 2 — correction attendue : Contrôler titres, méta description, H1, liens et alternatifs. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 3 — correction attendue : Vérifier la cohérence sitemap/robots/noindex. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 4 — correction attendue : Activer le JSON-LD uniquement s’il n’est pas déjà émis par le SEO plugin. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 5 — correction attendue : Valider BlogPosting et BreadcrumbList selon pertinence. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 6 — correction attendue : Rédiger un journal des anomalies classées par impact. Le formateur vérifie l’exécution effective et la cohérence des données, pas uniquement la présence du fichier.

Erreurs fréquentes et solutions

- Symptôme : Rich Results Test refuse un schéma. Diagnostic/correction : Consulter les champs exigés pour ce résultat enrichi, et les données réelles.
- Symptôme : Canoniques doublées. Diagnostic/correction : Désactiver une source avant de corriger la configuration globale.

Exercices progressifs et corrigés

Niveau 1 — Reproduire

Reproduire le fragment de code sans modifier sa logique ; expliquer les cinq points commentés ci-dessus.

Corrigé : la sortie et la structure doivent correspondre à la démonstration ; toute erreur PHP doit être éliminée avant la suite.

Niveau 2 — Adapter

Adapter les noms, données, paramètres ou styles à un second cas métier sans copier le code de manière incohérente.

Corrigé : documenter précisément les modifications, la raison et les résultats fonctionnels.

Niveau 3 — Durcir

Créer un cas d'échec volontaire : absence de contenu, refus d'autorisation, mauvais format de données ou ressource indisponible.

Corrigé : aucune erreur fatale ; l'échec doit être géré, visible dans les logs appropriés et couvert par un test manuel ou automatisé.

Quiz de fin de module — questions et réponses

1. Question : Une canonical impose-t-elle le choix de Google ? Réponse attendue : Non.
2. Question : robots.txt supprime-t-il l'indexation ? Réponse attendue : Non.
3. Question : Un JSON-LD garantit-il un rich result ? Réponse attendue : Non.

Grille de validation pratique — 20 points

- 4 pts : installation reproductible, fichiers et configuration corrects.
- 6 pts : comportement fonctionnel démontré en condition réelle.
- 4 pts : sécurité et gestion des erreurs appropriées.
- 3 pts : tests et preuves observables.
- 3 pts : code lisible, historique Git et documentation.

Seuil : 14/20 minimum ; toute vulnérabilité critique non corrigée bloque la validation.

Semaine 8 — Core Web Vitals, SQL, CI/CD et rollback

Bloc : Performance et production | Durée : 10 heures | Niveau : développeur

Objectif : Améliorer des métriques mesurées et livrer de façon reproductible.

Déroulé horaire et pédagogie

- 0 h à 1 h : prérequis, démonstration guidée et vocabulaire.
- 1 h à 4 h : explications approfondies et manipulations.
- 4 h à 6 h : lecture de code commentée et micro-exercices.

- 6 h à 9 h : TP autonome avec points de contrôle.
- 9 h à 10 h : correction, diagnostic, quiz et revue de code.

Cours approfondi — concepts à maîtriser

1. LCP, INP, CLS

LCP décrit le rendu du contenu principal, INP la réactivité et CLS les mouvements inattendus. Les seuils bons sont 2,5s / 200ms / 0,1 évalués en données terrain au 75e percentile quand disponibles.

Cas réel : Enregistrer trois mesures et identifier une cause dominante.

Erreur fréquente : Confondre score Lighthouse et résultats terrain.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

2. Images et priorités

Les dimensions préviennent CLS ; srcset et sizes servent des versions adaptées ; l'image LCP ne doit pas être lazy-loadée par défaut. fetchpriority=high peut être utile sur le média principal confirmé.

Cas réel : Inspecter l'image hero et le réseau mobile.

Erreur fréquente : Appliquer fetchpriority high à toutes les images.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

3. CSS et JS

Chargement conditionnel, différé et découpage des ressources réduisent travail inutile. Toujours vérifier interactions, notamment navigation mobile et formulaires.

Cas réel : Supprimer le JS inutile d'une page simple et comparer.

Erreur fréquente : Déferer un script dépendant sans tester sa chronologie.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

4. Base de données et cache

Mesurer le nombre et le coût des requêtes ; transients permettent un cache temporaire mais nécessitent une invalidation. Un objet cache persistant dépend de l'hébergement.

Cas réel : Créer un transient avec invalidation lors d'une sauvegarde de CPT.

Erreur fréquente : Mettre en cache une donnée critique sans procédure d'invalidation.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

5. Release et rollback

Une livraison comprend sauvegarde restaurable, build traçable, tests, staging, validation, publication et surveillance. Restaurer uniquement les fichiers ne rétablit pas les données si une migration a eu lieu.

Cas réel : Exécuter check.sh, build.sh, SHA256SUMS puis staging-smoke.sh sur WordPress.

Erreur fréquente : Prétendre que php -l démontre la compatibilité WordPress.

Vérification de compréhension : l'apprenant explique à voix haute pourquoi cette notion existe, la fonction WordPress correspondante et le risque métier si elle est ignorée.

Lecture de code, ligne par ligne

Le fragment suivant s'exécute dans un contexte WordPress chargé. Adapter les chemins et les noms de classes au projet.

```
$count = get_transient( 'agency_project_count' );
if ( false === $count ) {
    $counts = wp_count_posts( 'agency_project' );
    $count = (int) ( $counts->publish ?? 0 );
    set_transient( 'agency_project_count', $count, HOUR_IN_SECONDS );
}
return $count;
```

1. get_transient tente d'obtenir le résultat enregistré.
2. false indique l'absence du cache ou son expiration.
3. wp_count_posts lit les statistiques WordPress pour le CPT.
4. set_transient conserve temporairement la valeur calculée.
5. Il faut ajouter une invalidation lors des changements de publication.

Travaux pratiques guidés — consignes apprenant

Préparation : travailler sur une installation locale isolée ; activer l'affichage de logs hors production ; effectuer un commit avant toute modification.

1. Relever un état avant optimisation : URLs, timings, captures et configurations. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
2. Identifier la ressource LCP et corriger si nécessaire le chargement. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
3. Tester images responsive et réservations d'espace pour limiter CLS. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
4. Réduire les ressources inutiles en préservant les fonctionnalités. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
5. Exécuter la CI statique puis une matrice réelle PHP/WordPress en staging. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).
6. Créer et restaurer une sauvegarde de validation, puis documenter le rollback. Point de contrôle : enregistrer la preuve correspondante (capture, sortie CLI ou commit).

Commandes de référence

```
bash scripts/check.sh
bash scripts/build.sh
sha256sum -c dist/SHA256SUMS
WP_PATH=/chemin/wordpress bash scripts/staging-smoke.sh
```

Résultat attendu

La comparaison avant/après est documentée ; les opérations critiques sont reproductibles et le rollback a été testé.

Corrigé commenté et méthode de diagnostic

- Étape 1 — correction attendue : Relever un état avant optimisation : URLs, timings, captures et configurations. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 2 — correction attendue : Identifier la ressource LCP et corriger si nécessaire le chargement. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 3 — correction attendue : Tester images responsive et réservations d'espace pour limiter CLS. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 4 — correction attendue : Réduire les ressources inutiles en préservant les fonctionnalités. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 5 — correction attendue : Exécuter la CI statique puis une matrice réelle PHP/WordPress en staging. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.
- Étape 6 — correction attendue : Créer et restaurer une sauvegarde de validation, puis documenter le rollback. Le formateur vérifie l'exécution effective et la cohérence des données, pas uniquement la présence du fichier.

Erreurs fréquentes et solutions

- Symptôme : Bon score local, mauvais terrain. Diagnostic/correction : Vérifier cache, réseau, audience et données CrUX ; ne pas assimiler lab et field.
- Symptôme : Restauration incomplète. Diagnostic/correction : Restaurer aussi la base et contrôler URLs, médias et migrations.

Exercices progressifs et corrigés

Niveau 1 — Reproduire

Reproduire le fragment de code sans modifier sa logique ; expliquer les cinq points commentés ci-dessus.

Corrigé : la sortie et la structure doivent correspondre à la démonstration ; toute erreur PHP doit être éliminée avant la suite.

Niveau 2 — Adapter

Adapter les noms, données, paramètres ou styles à un second cas métier sans copier le code de manière incohérente.

Corrigé : documenter précisément les modifications, la raison et les résultats fonctionnels.

Niveau 3 — Durcir

Créer un cas d'échec volontaire : absence de contenu, refus d'autorisation, mauvais format de données ou ressource indisponible.

Corrigé : aucune erreur fatale ; l'échec doit être géré, visible dans les logs appropriés et couvert par un test manuel ou automatisé.

Quiz de fin de module — questions et réponses

1. Question : À quoi correspond INP ? Réponse attendue : Réactivité aux interactions.

2. Question : Quelle métrique évalue les déplacements ? Réponse attendue : CLS.
3. Question : Un contrôle syntaxique prouve-t-il la compatibilité ? Réponse attendue : Non.

Grille de validation pratique — 20 points

- 4 pts : installation reproductible, fichiers et configuration corrects.
- 6 pts : comportement fonctionnel démontré en condition réelle.
- 4 pts : sécurité et gestion des erreurs appropriées.
- 3 pts : tests et preuves observables.
- 3 pts : code lisible, historique Git et documentation.

Seuil : 14/20 minimum ; toute vulnérabilité critique non corrigée bloque la validation.

ÉVALUATION FINALE — PROJET INTÉGRATEUR

Sujet professionnel

Une agence souhaite administrer ses services et ses réalisations. Livrer un thème sans constructeur visuel, un plugin indépendant, une API lisible, une interface de consultation, un audit SEO et des preuves de performance.

Cahier des charges vérifiable

1. Site local reproductible, dépôt Git et README ; aucune modification du Core.
2. Templates : accueil, page, single post, home/blog, archive de réalisations, recherche, 404 ; menu utilisable au clavier et rendu responsive.
3. CPT projets/services, taxonomie secteur, métadonnées, réglages sécurisés ; conservation des données après changement de thème.
4. API GET publique sans brouillon, écritures interdites aux anonymes, permissions par objet ; tests de réponses 200, 400, 401/403 et 404 selon cas.
5. Audit de 10 URL : HTTP, canonical, noindex, sitemap, titre, structure Hn, JSON-LD et liens.
6. Mesures avant/après LCP, INP et CLS quand données terrain disponibles ; pour les données de laboratoire, déclarer l'outil et le protocole.
7. Livraison en ZIP, SHA256SUMS, sauvegarde, test de restauration et procédure de rollback.

Barème sur 100 et corrigé attendu

- Architecture & qualité — 20 pts : arborescence claire 5, Git et documentation 5, absence de modifications du Core 5, CI/statique 5.
- Thème — 25 pts : hiérarchie et boucles 10, responsive 6, accessibilité/navigation 5, composants réutilisables 4.
- Plugin & API — 30 pts : CPT/taxonomies 7, métadonnées/options 6, permissions/nonces/validation 9, tests REST 8.
- SEO & performance — 25 pts : indexation 6, JSON-LD 5, analyse CWV 6, optimisation mesurée 4, rollback 4.

Validation finale : 70/100 minimum ET pas de faille critique non corrigée ET démonstration pratique exécutée sur WordPress.

Jeu d'essai obligatoire

- 3 projets publiés, 2 brouillons, 1 projet en corbeille ; 2 services ; 2 termes de taxonomie ; au moins deux comptes avec des capacités différentes.
- Résultats attendus : les projets en brouillon n'apparaissent pas dans GET public ; un anonyme ne peut créer ni modifier ; les permissions objets sont vérifiées ; les pages d'archives n'affichent pas d'erreur 404 ; l'interface responsive demeure navigable.

Protocole de tests WordPress (à exécuter en staging)

1. Préparer une installation WordPress vierge sur une base temporaire et saisir sa version exacte.
2. Installer le thème et le plugin depuis les ZIP. Conserver le journal du processus.
3. Vérifier les dépendances PHP et versions WordPress recommandées ; ouvrir plusieurs pages réelles.
4. Générer le jeu d'essai ; parcourir front-page, single, home, archive et 404.
5. Exécuter les tests PHPUnit d'intégration si la suite de test WordPress est configurée.
6. Exécuter les requêtes REST GET/POST/PATCH/DELETE avec profils anonyme, auteur et administrateur.
7. Tester à 320, 768 et 1440 pixels, au clavier, en navigation privée et sans cache.
8. Exécuter curl, Rich Results Test/Schema Validator et Lighthouse ; noter les résultats (pas uniquement score global).
9. Tester restauration et rollback d'une version ; préserver la base de données.
10. Enregistrer date, versions, procédure, sorties de commandes et verdict PASS/FAIL/BLOQUÉ dans le procès-verbal.

Tableau de PV de recette — à remplir

- Test · Environnement · Preuve · Résultat · Correction ·
- Installation thème/plugin · À renseigner · URL/log · Non testé · — ·
- CPT et permissions · À renseigner · Capture/CLI · Non testé · — ·
- REST anonyme / authentifié · À renseigner · JSON HTTP · Non testé · — ·
- Templates responsive · À renseigner · Capture · Non testé · — ·
- SEO / JSON-LD · À renseigner · Validation · Non testé · — ·
- Core Web Vitals · À renseigner · Rapport · Non testé · — ·
- Rollback · À renseigner · Journal · Non testé · — ·

Grille d'autonomie de l'apprenant

La compétence est acquise lorsque l'apprenant peut expliquer pourquoi un choix technique est fait, reproduire la solution sans copier, corriger un échec provoqué et démontrer un résultat devant un tiers.

Sources officielles

- <https://developer.wordpress.org/themes/classic-themes/basics/template-hierarchy/>
- <https://developer.wordpress.org/plugins/>

- <https://developer.wordpress.org/rest-api/>
- <https://developer.wordpress.org/block-editor/reference-guides/theme-json-reference/>
- <https://developer.wordpress.org/themes/releasing-your-theme/testing/>
- <https://developers.google.com/search/docs>
- <https://web.dev/articles/vitals>