

# WPCookBook

Vincent Dubroeucq



# WPCookBook

Apprenez à développer, the WordPress Way

Vincent Dubroeucq

# WPCookBook

Copyright 2021 Vincent Dubroeucq

Le contenu de cet ebook reste la propriété de l'auteur. Aucun contenu ne peut être publié, modifié, revendu ou partagé sans l'autorisation écrite de l'auteur.

# Preface



Vincent s'est lancé dans une sacrée aventure en écrivant ce livre. Présenter les bonnes pratiques de développement de WordPress n'est pas une mince affaire. Pourtant, c'était nécessaire.

En effet, la plupart des livres pertinents sur le sujet sont en anglais et datent de plusieurs années. La documentation officielle propose aussi pas mal de ressources mais on ne sait pas vraiment par où commencer quand on se lance...

On peut tout trouver sur Internet et c'est justement ça le problème. On est submergé d'informations. Ce qui fait qu'on ne passe jamais à l'action.

Et pour avoir rédigé un ebook par le passé et conçu plusieurs formations en ligne, je sais que les lecteurs/étudiants souhaitent suivre un fil conducteur. Avoir quelque chose à quoi se raccrocher dans leur apprentissage.

C'est exactement ce que fait le WPCookBook de Vincent. De plus, il nous transmet les fondamentaux du développement pour WordPress dans la bonne humeur et avec passion (vous ne verrez pas passer les 400 pages).

Avec ces nouvelles compétences, vous aurez de quoi faire des sites WordPress qui tiennent la route. C'est certain.

Bonne lecture !

Alex Borto de [WPMarmite](#)

# Avant-propos



En 2009 je me suis lancé dans l'aventure WordPress, l'envie d'arrêter de créer des sites sous Notepad.exe pour gagner du temps était mon premier argument.

En 2010 je commence à créer mes propres extensions. Problème, je n'ai pas compris les hooks WordPress...

En 2011 je me lance officiellement à mon compte avec le peu de connaissances que j'ai, mais assez pour y arriver. Au fur à mesure du temps, on apprends, on touche, on bidouille. Aider les gens m'a aidé à améliorer mon code, ma façon de voir le code et de toucher à WP.

Ce qu'il m'a manqué durant ces années, c'est un guide le plus complet possible, écrit par un humain pour les humains. Même si on peut trouver des ressources sur Internet, on n'est jamais certain que c'est ce qu'il faut faire. Mais pour vous chers lecteurs et chères lectrices, c'est maintenant chose faite !

Vincent a écrit de ses petites mains pleines de doigts sa façon de développer avec WordPress, ou plutôt LA bonne façon de le faire, et avec ses mots. C'est ce qui rend ce livre encore plus humain : on se sent proche de ce qu'il fait, il a non seulement mis son temps mais aussi sa passion du développement bien fait.

J'ai acheté et lu l'intégralité de l'ebook avant de vous faire cette préface, je peux vous assurer qu'entre ce qui est déjà dedans et les prévisions des prochaines mises à jour, vous n'avez pas fini de l'ouvrir et le réouvrir.

Et vous savez quoi ? J'ai appris des choses ! J'ai vraiment été surpris de la profondeur à laquelle on va avec ce livre, sans pour autant qu'on s'y perde dans des futilités. Cela va sans dire que ses exemples sont sécurisés, il vous apprend aussi justement à sécuriser votre travail tout au long de vos développements.

Je vous laisse maintenant entre ses mains, vous avez du travail, vous n'êtes pas ici pour me lire ni pour penser à un éléphant avec un chapeau (mais vous venez de le faire...).

Julio Potier, Consultant en Sécurité Web, Ingénieur WordPress

<https://secupress.fr/>

# Introduction

Hey ! Salut à vous, bienvenue dans WPCookBook ! Je suis honoré de votre présence avec moi !

Dans cet ebook, vous apprendrez tout ce dont vous avez besoin pour développer pour WordPress de la “bonne” manière.

Je mets “bonne” entre guillements car évidemment, il n'y a pas qu'une seule et unique façon de coder pour WordPress, mais certaines actions ou certains développements doivent se faire d'une certaine façon, pour maximiser la compatibilité avec vos extensions et pour garantir que votre code fonctionne de façon attendue.

WordPress est une super plateforme. Elle est simple à utiliser, permet de publier un site très rapidement, d'en adapter le design et d'en étendre les fonctionnalités en quelques clics. Le nouvel éditeur permet d'écrire du contenu riche et complexe, aux mises en page variées assez facilement.

C'est juste énorme.

Pour les développeurs, WordPress met à notre disposition de nombreuses APIs pour effectuer les tâches de développement les plus courantes.

Pourtant, je suis parfois appelé pour intervenir sur des sites (maintenance ou évolution) et quand j'ouvre le site dans mon éditeur de code, j'ai parfois un petit moment de désespoir ou d'horreur selon le cas.

— Ah ok, le lien vers la page de réglages a été rajouté avec jQuery. Nice !

Je ne veux absolument pas jeter la pierre. On ne sait pas ce qu'on ne sait pas. Mais ajouter un menu avec jQuery montre une méconnaissance de WordPress, simplement. Et c'est pour cette raison que j'écris ce guide.

Il y a des APIs et outils pour tout ! Widgets, code courts (shortcodes), API REST, AJAX, types de contenu personnalisés (CPT), taxonomies, réglages, HeartBeat, etc...

Pourtant je trouve des développements “maison” pour des fonctionnalités qui existent déjà ou qui pourraient utiliser bien mieux les fonctions natives de WordPress !

Par exemple, pourquoi faire une requête SQL à la main pour un type de contenu alors qu'on peut faire une `WP_Query` ou utiliser la fonction `get_posts()` ? Pourquoi risquer la faille de sécurité injection SQL (SQLi) ? Plus personnellement je trouve qu'écrire une requête SQL est bien plus compliqué et pénible qu'appeler une fonction, pas vous ?

Une balise `<script>` directement dans le fichier du thème peut fonctionner, mais non ce n'est pas comme ça que WordPress s'attend à ce qu'on charge les scripts.

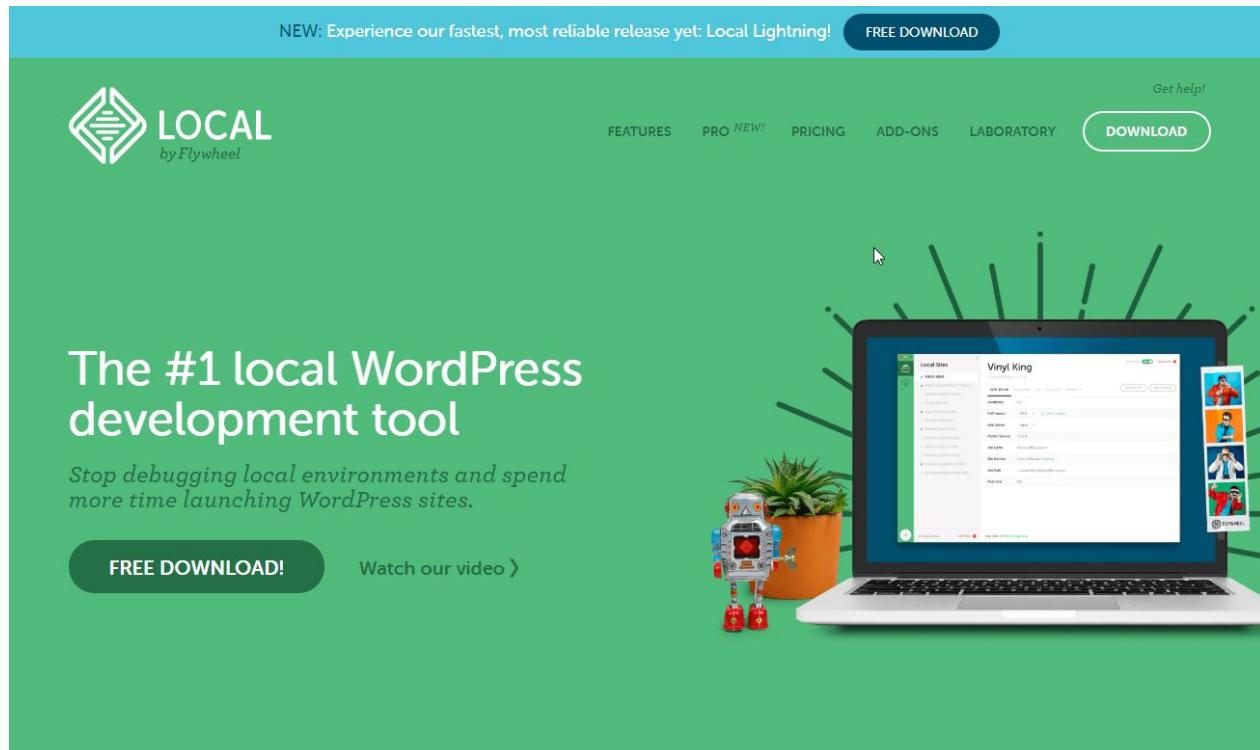
WordPress ne fonctionne simplement pas comme ça. Et ce n'est pas parce que cela fonctionne que c'est bien.

Ensemble, dans ce guide, notre objectif est de prendre conscience de la quantité d'outils que WordPress met à notre disposition et d'apprendre à utiliser au maximum ces outils en respectant les bonnes pratiques de développement pour cette plateforme.

## Votre environnement de développement

Pour suivre tout au long de ce guide, et faire vos expériences, vous allez avoir besoin d'un environnement de travail local, c'est-à-dire un site non pas sur un serveur en production, mais sur votre ordinateur.

Pour ce faire, je recommande Local By Flywheel, ou Mamp ou encore Lamp, bref, un Xamp.



Local by Flywheel

Ces outils font sensiblement la même chose et s'installent sensiblement de la même façon : on télécharge un installateur qu'on exécute, simplement. Cela installe tout le nécessaire pour faire tourner le serveur Web local et gérer la base de données. On n'a plus qu'à installer WordPress et go.

## Une installation fraîche de WordPress

Pour ce guide, je vais utiliser Local by Flywheel. La procédure d'installation de WordPress est un poil plus complexe pour MAMP, car il faut créer manuellement la base de données, donc on va aller au plus simple et rapide.

- Téléchargez l'installateur sur <https://local.getflywheel.com/>
- Lancer l'installateur.
- Suivez la procédure d'installation.
- Sur l'écran d'accueil, ajoutez un site. Le mien s'appellera WPCookBook.
- Choisissez vos paramètres. Personnellement, je mets la dernière version de PHP, et Apache.
- C'est bon, le site est créé !

## Thèmes et extensions

Sur ce site, on ne va toucher à rien du tout pour le moment ! Pas d'extensions, pas de thème spécial à installer !

On va utiliser le thème TwentyNineteen par défaut, et on ne va installer aucune extension.

Le but du jeu est d'apprendre à exploiter WordPress, donc on va garder la base la plus propre possible, sans aucune autre interaction que vos extensions tests. Tout ce qu'on va faire, c'est créer un thème enfant dans un prochain chapitre.

## Menus

On va simplement ajouter un menu, avec un lien vers une page (la page Exemple suffit), vers un article (Hello World) et une catégorie. Aussi, veillez à ce que la page d'accueil du site liste les derniers articles et ce sera tout pour l'instant. L'objectif est juste de pouvoir naviguer vers les templates de page courants.

Votre site local devrait ressembler à ça :



# Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

• vincent • septembre 12, 2019 • Uncategorized • Un commentaire  
✍ [Modifier](#)

Recherche...

**Rechercher**

## Articles récents

[Hello world!](#) septembre 12, 2019

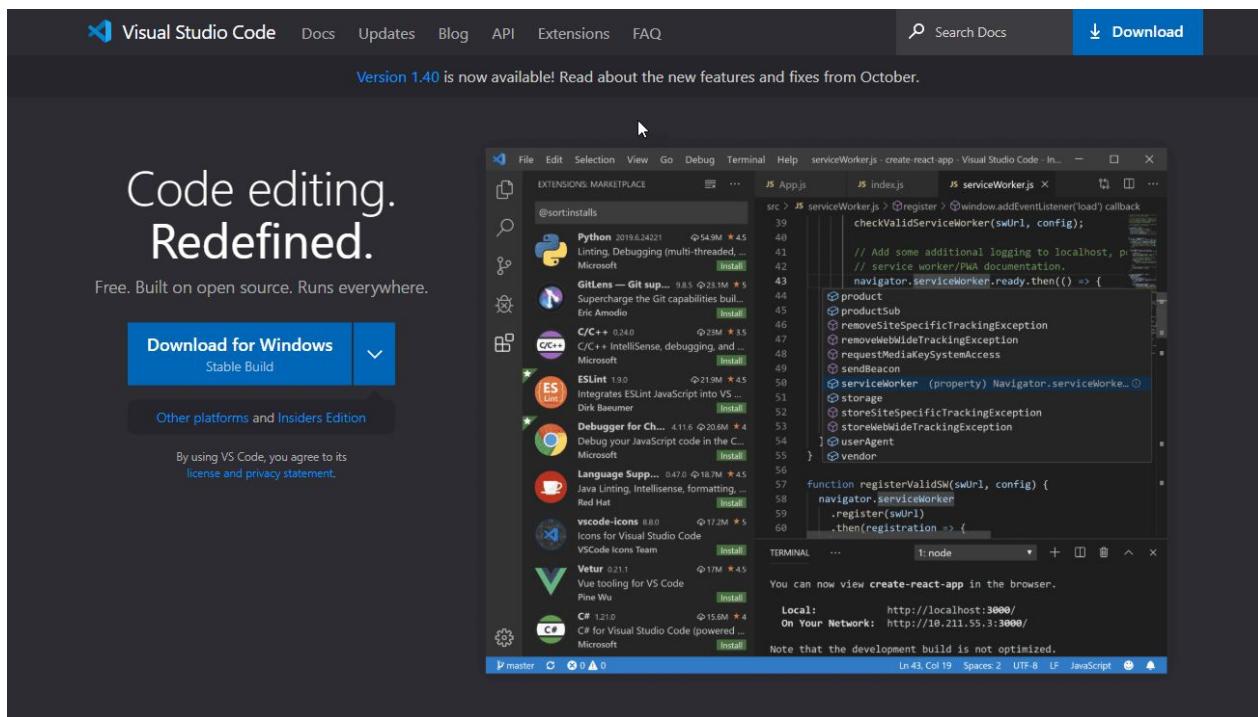
Notre site local est prêt !

### Un éditeur de code

Pour éditer nos thèmes et extensions, il faudra un éditeur de texte. Pas juste un bloc note, un vrai éditeur de code. Il y en existe plein : des gratuits et des payants.

Parmi les meilleurs gratuits, il y a :

- VS Code (<https://code.visualstudio.com/>)
- Atom (<https://atom.io/>)
- Brackets (<http://brackets.io/>)



VS Code

Je vais utiliser VS Code, mais vous pouvez utiliser n'importe quel éditeur de texte. Il suffit d'aller sur le site et de le télécharger, de l'installer et go. Easy.

La seule manoeuvre qu'on va effectuer est d'activer le mode debug de WordPress.

Il suffit d'éditer le fichier `wp-config.php` de WordPress situé à la racine de l'installation, et d'y ajouter la ligne suivante, juste avant le commentaire `/* That's all, stop editing! Happy blogging. */`:

```
define( 'WP_DEBUG', true );
```

Cela aura pour effet d'afficher les erreurs PHP, les notices et warnings pour l'utilisation de fonctions dépréciées, l'utilisation de variables non définies, etc.

Cela vous permettra de produire un code plus sûr et plus propre. Gardez simplement en tête que notre objectif est zéro notice !

## La documentation de WordPress

La meilleure documentation pour WordPress que l'on puisse trouver est l'officielle !

Chaque fonction et hook du cœur de WordPress est documenté dans la [Code Reference](#), avec des exemples d'utilisation pour la plupart des cas.

Il suffit de taper la fonction / classe / méthode que l'on cherche dans la barre de recherche pour obtenir toutes les fonctions et hooks pertinents listés.

Chaque page de la documentation donne une description rapide de la fonction ou du hook, le code source que l'on peut lire et des liens utiles qui permettent de trouver où la fonction est utilisée dans WordPress et quelles autres fonctions/hooks elle utilise. On peut aussi parfois trouver des exemples fournis par des contributeurs en dessous.

C'est juste énorme. Un outil indispensable et on va y revenir plusieurs fois au long de ce guide.

Dès que votre site local est installé, ouvrez votre éditeur de code, et go !

# La plugin API de WordPress

La Plugin API (API des extensions) est l'API fondamentale de WordPress. C'est sur celle-ci que se basent toutes les interactions entre WordPress et ses extensions, mais aussi ses thèmes.

C'est simplement un ensemble de fonctions qui permettent aux extensions et thèmes de se greffer sur WordPress et de venir modifier son comportement.

C'est l'API qui rend ce système de thèmes et extensions possible, et qui fait de WordPress le CMS puissant et personnalisable que nous connaissons, avec ses millions d'extensions et thèmes.

Wow. Ca doit être un truc bien compliqué, non ? Eh bien en fait, pas du tout. Ou du moins, pas tant que ça.

Note : ce premier chapitre risque d'être long, mais il pose les fondements nécessaires pour comprendre WordPress et tout le reste de l'ebook ! Accrochez-vous !

## Comment fonctionne WordPress

Si on y réfléchit bien, WordPress est juste un gros script PHP. C'est tout. C'est un programme, qui s'exécute de façon plus ou moins linéaire, mais c'est juste un programme PHP.

Là où WordPress est magique c'est que lors de son exécution, il nous donne (ou plutôt aux extensions et thèmes), la possibilité d'agir sur presque tout, presque n'importe quand.

Pour schématiser, à chaque fois qu'il fait quelque chose (aller chercher une donnée en base ou l'afficher, ou avant d'afficher un modèle de page, ou alors pendant l'affichage d'un modèle, etc.), il s'arrête et demande :

— Ok, les extensions et thèmes activés ! Dites, je vais faire une requête pour aller chercher des articles. Je compte chercher les 10 derniers articles, toutes catégories confondues. Est-ce que quelqu'un a une objection ? Un détail à ajouter ? Non ? Bon ben j'y vais !

...

— C'est bon les gars j'ai les articles ! Les voilà. Vous avez un truc à faire ? Non ? Bon ben je les affiche alors. Je compte utiliser *home.php* dans le thème actif. Ça vous va ?

Etc.

Et pour lever la main et dire "Oui ! J'ai un truc à faire" au bon moment, on utilise la plugin API : le système de hooks de WordPress.

## Les fameux hooks

Si vous avez déjà entendu parler des hooks de WordPress, ou utilisé un hook, alors vous avez déjà vu ou utilisé la plugin API. Celle-ci est constituée des fonctions de ce système de hooks.

Les hooks (crochets en anglais) sont les points d'accroche que WordPress met à notre disposition pour

intervenir à différents moments de son exécution.

Il y a deux types de hooks et chaque type de hooks a sa série de fonctions.

- Les **filtres** vont permettre aux extensions ou thèmes de modifier une donnée fournie par WordPress. On nous donne la donnée, on la traite comme on veut et on la retourne pour que le programme continue de s'exécuter.
- Les **actions** vont permettre d'agir à un certain moment, mais ne nécessitent pas qu'on retourne une valeur. Cependant, une ou plusieurs valeurs peuvent nous être fournies, pour qu'on puisse faire notre travail correctement.

Pour que vous compreniez de suite comment ça marche, on va taper un peu de code.

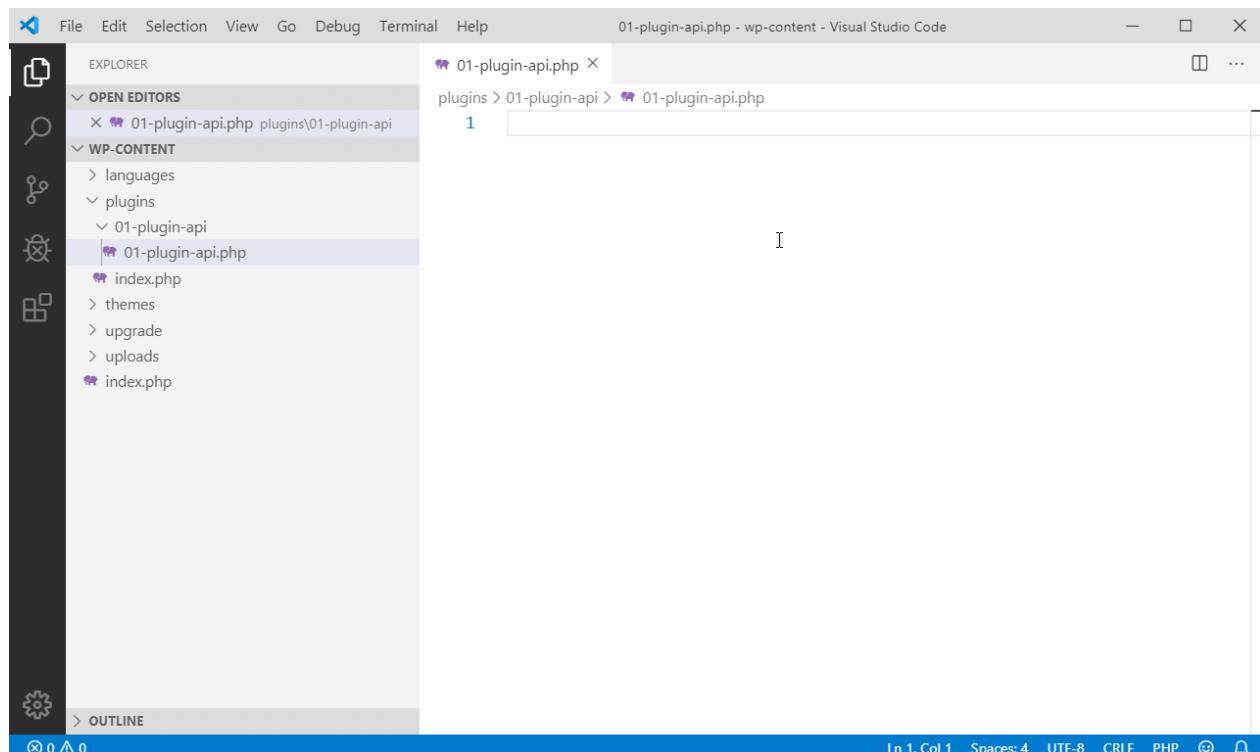
## Comment créer une extension

Wow ! On va déjà créer une extension ?! Eh ouais.

Sur votre installation fraîche de WordPress, naviguez dans le dossier `wp-content/plugins/`, puis créez un dossier `01-plugin-api/`

Ouvrez le dossier `wp-content/` entier dans votre éditeur de code. Comme ça, vous aurez accès rapidement à toutes les petites extensions que l'on va créer tout au long de ce guide. Puis créez un fichier PHP ayant le même nom que le dossier, donc `01-plugin-api.php`.

Normalement, vous devriez avoir ceci :



Structure du dossier dans l'éditeur

Pour que WordPress reconnaisse notre extension, il faut ajouter une entête spéciale dans notre fichier, en

déclarant son nom ainsi que d'autres métadonnées utiles.

On va donc ajouter ceci:

```
<?php
/**
 * Plugin Name: Plugin API - Basic hook examples
 */
```

L'entête `Plugin Name` est la seule entête obligatoire pour que tout fonctionne. Si vous allez dans votre administration et vérifiez la liste de vos extensions, vous pourrez voir que notre extension apparait !

Vous pouvez l'activer de suite, mais vu que notre fichier est vide, elle ne fera rien, évidemment.

The screenshot shows the WordPress admin interface under the 'Extensions' tab. On the left sidebar, 'Extensions' is selected. The main area displays a list of installed extensions. One extension is listed: 'Plugin API - Basic hook examples.' Below the extension name are two buttons: 'Activer' (Enable) and 'Supprimer' (Delete). At the bottom of the list, there is a section for 'Actions groupées' (Grouped actions) with a 'Appliquer' (Apply) button.

Page des extensions avec la nôtre.

On peut compléter l'entête en ajoutant d'autres informations utiles :

```

<?php
/**
 * Plugin Name:      Plugin API - Basic hook examples.
 * Plugin URI:     https://example.com/plugins/01-plugin-api/
 * Description:    This registers a few basic hooks, just to see how hooks work.
 * Version:        1.0
 * Requires at least: 5.2
 * Requires PHP:    7.2
 * Author:         Vincent Dubroeucq
 * Author URI:    https://vincentdubroeucq.com/
 * License:        GPL v2 or later
 * License URI:   https://www.gnu.org/licenses/gpl-2.0.html
 * Text Domain:   01-plugin-api
 * Domain Path:  languages/
 */

```

- **Plugin URI** est l'adresse à laquelle l'extension sera disponible. Cela peut être une URL sur votre site, ou celle du répertoire des extensions de [WordPress.org](#).
- **Description** est une description courte apparaissant dans l'admin (140 caractères max).
- **Version** est la version de votre extension. Utilisez un numéro de version sémantique: majeure.mineure.correctif.
- **Requires at least** et **Requires PHP** sont la version minimale de WordPress et de PHP nécessaires pour que l'extension fonctionne correctement.
- **Author** et **Author URI**: C'est vous (et votre site web) ! Si vous travaillez à plusieurs sur l'extension, vous pouvez mettre plusieurs noms séparés par des virgules.
- **Licence** et **Licence URI** sont l'intitulé et le lien vers la licence sous laquelle votre extension sera distribuée. Même si techniquement, on peut utiliser n'importe quelle licence compatible avec la GPL, on utilise en général la même que WordPress: GPL 2 ou plus.
- **Text Domain**: C'est la clé qui va être utilisée pour associer les bonnes traductions aux chaînes à traduire de votre extension.
- **Domain Path**: C'est le chemin que WordPress va utiliser pour chercher les traductions de votre extension.

Je parlerai plus en détails du **Text Domain** et **Domain Path** dans un autre chapitre, pas de panique.

On va juste ajouter une ligne après ce bloc de commentaires.

```
defined( 'ABSPATH' ) || die();
```

Cela permet simplement de vérifier si WordPress est bien chargé, en vérifiant si une de ses constantes est définie. Si quelqu'un essaie d'accéder directement au fichier sans passer par WordPress, on exécute **die()** et donc on arrête le script.

## Notre premier hook

Notre extension est en place, on va pouvoir (enfin !) commencer à s'amuser.

Voilà pour commencer un exemple un peu bidon. Dans votre extension, ajoutez ceci:

```
// in main plugin file
add_filter( 'the_title', 'wpcookbook_title' );
function wpcookbook_title( $title ){
    $title = 'TOTO';
    return $title;
}
```

Que fait-on ici ?

Avec la fonction `add_filter()`, on ... ajoute un filtre. Plus précisément, on enregistre une fonction (`wpcookbook_title`) pour qu'elle soit exécutée à un moment précis (`the_title`). Elle est simplement ajoutée en mémoire à la liste des fonctions à exécuter à ce moment-là.

Note : Les fonctions déclarées hors d'une classe PHP sont disponibles globalement. Ce qui veut dire qu'elles peuvent rentrer en conflit avec des fonctions de WordPress ou des autres extensions (essayez de déclarer une fonction appelée `the_title` pour voir !). Pour éviter tout conflit de nom, il faut impérativement préfixer vos fonctions. Par convention, le préfixe devrait être le slug de votre extension. Donc ici, notre fonction devrait normalement s'appeler `01_plugin_api_title()`. Mais, commencer une fonction par un chiffre est non valide, donc pour faire plus simple j'utiliserais le préfixe `wpcookbook_` tout au long de ce guide.

En faisant ainsi, on a "hooké" notre fonction sur le hook `the_title`. C'est-à-dire qu'on dit à WordPress :

— Ok WordPress. A chaque fois que tu tombes sur le hook appelé `the_title`, tu pourras exécuter ma fonction `wpcookbook_title()` s'il te plaît ?

Maintenant, si on va sur le devant du site, voilà ce qu'on a :

WPCookBook — Apprenez à développer pour WordPress  
**TOTO TOTO TOTO**

## TOTO

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

vincent septembre 12, 2019 Uncategorized Un commentaire Edit

Recherche...

Rechercher

## Articles récents

TOTO

Tous les titres sont changés en TOTO

Magnifique. Notre extension remplace tous les titres par “TOTO”. Articles, pages, dans les widgets, etc...

À chaque fois que WordPress tombe sur le hook `the_title`, notre fonction est exécutée : WordPress lui donne le titre, et on lui rend “TOTO”.

## Mais comment ça marche ? — Version courte

Le thème actif `twentynineteen` utilise la fonction `the_title()` pour afficher le titre de chaque publication. Cette fonction utilise `get_the_title()` pour récupérer ce titre.

Et dans cette fonction, on trouve la ligne suivante :

```
return apply_filters( 'the_title', $title, $id );
```

La fonction `get_the_title()` ne fait pas que retourner le titre, mais le passe d'abord dans la fonction `apply_filters()`.

La fonction `apply_filters()` permet d'intervenir à ce moment précis du déroulement du script car elle va déclencher l'exécution de toutes les fonctions prévues pour s'exécuter sur le hook `the_title` en leur passant comme paramètres le titre de la publication (`$title`) et son identifiant (`$id`).

Relisez bien ce dernier paragraphe ! `apply_filters()` va déclencher TOUTES les fonctions associées au hook `the_title`.

Si vous vous souvenez bien, c'est sur ce hook que l'on a greffé la fonction `wpcookbook_title()` de notre

plugin en utilisant `add_filter()`.

Donc à chaque fois que notre thème utilise `the_title()` pour afficher un titre, WordPress va tomber sur cette ligne et va déclencher toutes les fonctions enregistrées pour s'exécuter à ce moment précis (“hookées” sur `the_title`) en leur passant le titre et l'identifiant de la publication en question.

Sur la page d'accueil qui liste les articles de blogs, notre fonction est donc déclenchée plusieurs fois !

## `add_filter()` et `apply_filters()` en détails — Version longue

```
apply_filters( string $tag, mixed $value [, mixed $...] )
```

Cette fonction exécute toutes les fonctions de rappel associées au hook passé en premier paramètre, en leur passant les autres paramètres.

La fonction prend donc au moins deux paramètres : `$tag`, et `$value`.

`$tag` est une chaîne de caractères représentant le nom du hook.

`$value` est la valeur que l'on va passer aux fonctions à exécuter sur ce hook, et qui doit être retournée. C'est notre valeur principale, ce qu'on cherche à modifier.

Les autres paramètres sont optionnels et sont des données dont on pourrait avoir besoin pour faire ce que l'on souhaite, mais WordPress ne s'attend pas à ce qu'on les retourne.

```
return apply_filters( 'the_title', $title, $id );
```

Ici, `apply_filters()` va prendre toutes les fonctions “hookées” sur `the_title` et va les exécuter en leur passant un paramètre `$title` et un paramètre `$id`.

C'est cette fonction `apply_filters()` qui permet aux extensions WordPress de se greffer à cet endroit précis du déroulement du script, c'est à dire à chaque fois que `the_title()` ou `get_the_title()` est utilisée.

```
add_filter( string $tag, callable $function_to_add, int $priority = 10, int $accepted_args = 1 )
```

Cette fonction sert à “hooker” une autre fonction. C'est-à-dire à la mettre en mémoire dans la liste des fonctions à exécuter sur le hook précisé par le premier paramètre.

La fonction prend 4 paramètres.

`$tag` est le nom du hook sur lequel on veut exécuter notre fonction.

`$function_to_add` est le nom de la fonction à hooker.

`$priority` est optionnel. C'est la priorité à laquelle la fonction va être exécutée. Plus le nombre est grand, plus c'est tard, et plus c'est petit, plus c'est tôt.

Si par exemple vous devez absolument toucher à une donnée après toutes les autres extensions, alors

mettez un grand nombre. Si vous devez être le premier à avoir accès à la donnée, mettez 1.

Par défaut la priorité est 10 et les fonctions sur une même priorité s'exécutent dans l'ordre dans lequel elles ont été ajoutées à la file d'attente.

Donc si une extension est chargée après la vôtre et se hooke au même endroit avec la même priorité que vous, alors sa fonction sera exécutée après la vôtre. C'est simplement le principe du premier arrivé premier servi.

Dans 95% des cas, vous n'avez pas besoin de modifier la priorité. Les 5% restants, c'est quand plusieurs extensions se hookent au même endroit, car elles agissent sur les mêmes choses et donc se marchent sur les pieds !

\$accepted\_args est optionnel. C'est le nombre de paramètres qui vont être passés à la fonction de rappel à exécuter. Par défaut, c'est 1.

Vous vous souvenez ? La fonction `apply_filters()` passe deux paramètres aux fonctions de rappel à exécuter sur le hook `the_title`.

```
return apply_filters( 'the_title', $title, $id );
```

Dans notre extension, nous avons omis les deux derniers paramètres et donc utilisé leur valeur par défaut.

Avec tous les paramètres, cela donne :

```
add_filter( 'the_title', 'wpcookbook_title', 10, 2 );
/**
 * Changes the post and page titles to 'TOTO'
 *
 * @param string $title The title of the post
 * @param int    $id    The post id
 * @return string $title
 */
function wpcookbook_title( $title, $id ){
    $title = 'TOTO ' . $id;
    return $title;
}
```

On indique dans `add_filter()` que l'on va passer deux paramètres à notre fonction de rappel `wpcookbook_title()`, et on lui donne bien les deux paramètres `$title` et `$id` que nous fournit l'appel à `apply_filters()` dans la fonction `get_the_title()`, où le hook se trouve.

On en profite au passage pour ajouter un commentaire qui va décrire brièvement ce que fait la fonction, ce à quoi correspondent chaque paramètre, leur type attendu et la valeur de retour de la fonction.

Pour tester, on va simplement concaténer l'identifiant au titre "TOTO".

WPCookBook — Apprenez à développer pour WordPress  
[TOTO 6](#) [TOTO 7](#) [TOTO 8](#)

## TOTO 1

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

vincent septembre 12, 2019 Uncategorized Un commentaire Modifier

Recherche...

Rechercher

## Articles récents

[TOTO 1](#)

Les titres affichent maintenant l'identifiant

Attention ! Notre fonction doit absolument retourner son premier paramètre \$title.

C'est très important. Un filtre permet de manipuler une donnée que WordPress nous a "prêtée" en quelque sorte, mais il a besoin qu'on la lui rende pour qu'il puisse poursuivre son travail et s'exécuter correctement.

## Résumé

Nous avons, avec add\_filter(), enregistré une fonction pour s'exécuter à chaque fois que WordPress rencontre le hook the\_title.

Ce hook se trouve dans la fonction get\_the\_title(), qui est utilisée par the\_title(), qui est elle-même utilisée partout dans notre thème pour afficher les titres. Donc notre fonction est déclenchée partout !

- la fonction apply\_filters() permet d'exécuter les fonctions associées au hook passé en premier paramètre. Elle passe au moins une valeur à chaque fonction de rappel.
  - La fonction add\_filter() déclare une fonction de rappel à déclencher à un moment précis.
- Attention à bien passer les paramètres nécessaires à notre fonction de rappel.

Ça va ? Vous suivez ? Prenez 5 minutes de pause et un café, vous l'avez bien mérité !

On continue.

## do\_action() et add\_action()

Maintenant, ajoutons à notre extension une autre fonction que l'on va hooker sur wp\_footer.

```
add_action( 'wp_footer', 'wpcookbook_footer' );
/**
 * Prints a custom message in the footer.
 */
function wpcookbook_footer(){
    ?>
        <p>This is a custom message to print in the footer! </p>
    <?php
}
```

Rafraîchissez la page de votre site.

## Catégories

Uncategorized

## Méta

[Admin. du site](#)  
[Déconnexion](#)  
[Flux RSS des articles](#)  
[RSS des commentaires](#)  
[Site de WordPress-FR](#)

WPCookBook, Fièrement propulsé par WordPress.

This is a custom message to print in the footer !



Notre message apparaît dans le pied de page.

Vous vous souvenez ? Un peu plus haut, on avait vu qu'il y avait deux types de hooks: les filters et les actions.

Les filters permettent de modifier une donnée, qu'il faut donc absolument retourner (eh, je radote là, non ?), alors que les actions permettent simplement d'agir à ce moment-là, sans nécessairement nous fournir de données, ni en attendre en retour.

Ici, avec la fonction add\_action(), on va enregistrer en mémoire notre fonction wpcookbook\_footer() pour qu'elle s'exécute quand WordPress rencontrera le hook wp\_footer.

Si on ouvre le code du fichier footer.php du thème twentynineteen, on voit qu'il utilise la fonction wp\_footer(), qui contient cette ligne :

```
do_action( 'wp_footer' );
```

```
do_action( string $tag, $arg = '' )
```

La fonction `do_action()` prend au moins un paramètre et nous permet d'intervenir à cet endroit précis en exécutant toutes les fonctions de rappel prévues pour être exécutées sur le hook `$tag` passé en premier paramètre. Si on lui passe d'autres paramètres, ces derniers sont passés à chaque fonction de rappel.

Donc ici, `do_action( 'wp_footer' )` va simplement exécuter toutes les fonctions hookées sur `wp_footer`, sans rien leur donner de spécial.

Cela ne vous rappelle rien ?

```
add_action( string $tag, callable $function_to_add, int $priority = 10, int $accepted_args = 1 )
```

`add_action()` permet de déclarer une fonction `$function_to_add` à exécuter quand WordPress rencontrera le hook `$tag`.

`$priority` est la priorité (optionnelle), `$accepted_args` (optionnel aussi) est le nombre de paramètres qui vont être passés à la fonction de rappel `$function_to_add`. Par défaut, c'est 1.

Cela ne vous rappelle rien ?

`add_action()` fonctionne exactement comme `add_filter()`, et `do_action()` fonctionne de façon très similaire à `apply_filters()` ! La seule différence est que `apply_filters()` exige qu'on lui renvoie une valeur !

Comme pour l'exemple précédent, on a omis tous les paramètres optionnels dans notre code. Avec tous les détails, cela donne :

```
add_action( 'wp_footer', 'wpcookbook_footer', 10, 1 );
/**
 * Prints a custom message in the footer.
 *
 * @param mixed $args Arguments passed to the callback by corresponding do_action() call.
 * Defaults to empty string.
 */
function wpcookbook_footer( $args ){
    ?>
        <p>This is a custom message to print in the footer! </p>
        <p><?php var_dump( $args );?></p>
    <?php
}
```

J'ai ajouté un `var_dump()` pour afficher le contenu de `$args`.

Note : J'ai fermé les balises `<?php ?>` pour pouvoir plus facilement écrire du code HTML, parce que je savais que ce hook était utilisé pour afficher quelque chose. Selon si le hook est prévu pour afficher quelque chose ou pas et selon la quantité de code HTML à afficher, on pourra préférer ne pas les fermer et utiliser des fonctions PHP comme `echo`, `printf` par exemple.

Note (encore) : le texte à afficher est codé en dur dans le code HTML. C'est juste pour l'exemple, ok ?  
On ne fait jamais ça ! J'explique tout cela plus en détails dans la section sur l'internationalisation.

## Catégories

[Uncategorized](#)

## Méta

[Admin. du site](#)  
[Déconnexion](#)  
[Flux des publications](#)  
[Flux des commentaires](#)  
[Site de WordPress-FR](#)

WPCookBook, Fièrement propulsé par WordPress.



This is a custom message to print in the footer !

string(0) ""



On voit bien la valeur par défaut de \$args

## En résumé

C'est au final assez simple.

`add_filter()` et `add_action()` servent à hooker nos fonctions pour qu'elles s'exécutent au bon moment, avec les bonnes données.

`apply_filters()` et `do_action()` déclenchent toutes les fonctions hookées, en leur passant les informations nécessaires.

Si vous avez compris comment fonctionnent ces quatre fonctions, vous avez effectivement compris comment vous greffer sur le fonctionnement de WordPress, et donc comment fonctionnent toutes les extensions et thèmes !

## Les autres fonctions de l'API

Cela dit, il y a d'autres fonctions disponibles dans la Plugin API :

- `has_action()`
- `has_filter()`
- `do_action_ref_array()`

- `apply_filters_ref_array()`
- `did_action()`
- `current_filter()`
- `current_action()`
- `remove_action()`
- `remove_filter()`
- `remove_all_actions()`
- `remove_all_filters()`

Ces fonctions sont utiles, mais beaucoup moins courantes que les quatre décrites précédemment, donc on va rapidement les passer en revue.

`has_filter( string $tag, callable|bool $function_to_check = false )` et `has_action()` vont simplement vérifier si une fonction particulière `$function_to_check` est hookée sur un hook `$tag` particulier. Si on omet `$function_to_check`, la fonction vérifie simplement si une fonction est hookée sur `$tag`.

`do_action_ref_array( string $tag, array $args )` fonctionne exactement comme `do_action()` sauf que les paramètres à passer aux fonctions hookées le sont sous forme d'un tableau unique au lieu d'être listées. Idem pour `apply_filters_ref_array()` et `apply_filters()`.

`did_action( string $tag )` retourne le nombre de fois qu'un hook `$tag` a été déclenché.

`current_filter()` et `current_action()` renvoient le nom du filtre sur lequel la fonction qui l'appelle est hookée.

`remove_action()` et `remove_filter()` permettent de "déhooker" une fonction précise, sur un hook précis, avec une priorité précise. C'est-à-dire qu'on va faire en sorte que la fonction précédemment enregistrée pour s'exécuter ne le fasse pas.

Par exemple, si on ajoute cette ligne dans notre extension, après notre fonction :

```
remove_action( 'wp_footer', 'wpcookbook_footer', 10 );
```

Notre fonction `wpcookbook_footer()` ne s'exécutera pas, et on n'aura pas le message dans le pied de page, simplement parce qu'on a enlevé la fonction de la liste d'attente sur ce hook.

Evidemment, le moment où cette ligne s'exécute est important. Si vous utilisez `remove_action()` ou `remove_filter()` AVANT que la fonction en question ne soit hookée (c'est-à-dire avant l'appel à `add_action()` ou `add_filter()` correspondant), cela ne fera rien.

Idem si vous les utilisez APRÈS que la fonction en question soit déclenchée. Il est trop tard.

Aussi, si la fonction à déhooker a été prévue à une certaine priorité différente de 10 (valeur par défaut), alors il vous faut passer la priorité correspondante en paramètre.

Dans notre exemple, modifions la priorité de notre fonction:

```
add_action( 'wp_footer', 'wpcookbook_footer', 15, 1 );
/** ... */
function wpcookbook_footer( $args ){
    ...
}

remove_action( 'wp_footer', 'wpcookbook_footer', 10 );
```

Sur le devant du site, cela donne :

## Catégories

[Uncategorized](#)

## Méta

[Admin. du site](#)  
[Déconnexion](#)  
[Flux des publications](#)  
[Flux des commentaires](#)  
[Site de WordPress-FR](#)

WPCookBook, Fièrement propulsé par WordPress.

This is a custom message to print in the footer !

string(0) ""



Malgré le *remove\_action()*, on a toujours notre message.

WordPress cherche à enlever de la file d'attente de `wp_footer` une fonction `wpcookbook_footer()`, à la priorité 10. Or, cette fonction a été enregistrée pour s'exécuter à la priorité 15 !

Pour lister les fonctions prévues pour le hook courant et trouver leur priorité, vous pouvez utiliser l'astuce suivante :

```
add_action( 'wp_footer', 'wpcookbook_footer', 15, 1 );
/** ... */
function wpcookbook_footer( $args ){
    ?>
        <p>This is a custom message to print in the footer! </p>
        <pre>
            <?php global $wp_filter; print_r($wp_filter[current_filter()]); ?>
        </pre>
    <?php
}

remove_action( 'wp_footer', 'wpcookbook_footer', 10 );
```

Tous les fonctions hookées sont rangées dans la variable globale `$wp_filter`, donc on peut les lister en vérifiant ce qu'il y a dans ce tableau global pour le hook courant `current_filter()`.

This is a custom message to print in the footer !

```
WP_Hook Object
(
    [callbacks] => Array
        (
            [15] => Array
                (
                    [plugin_api_footer] => Array
                        (
                            [function] => plugin_api_footer
                            [accepted_args] => 1
                        )
                )
            )
        )
    [20] => Array
        (
            [wp_print_footer_scripts] => Array
                (
                    [function] => wp_print_footer_scripts
                    [accepted_args] => 1
                )
        )
)
Liste des fonctions et leur priorité
```

On peut voir qu'il y a une fonction prévue à la priorité 15 (la nôtre), une autre à 20, etc.

En changeant la priorité à 15 dans notre `remove_action()`, notre fonction n'est plus exécutée.

`remove_all_actions()` et `remove_all_filters()` déhookent toutes les fonctions hookées sur un hook passé en paramètre. Bim. Plus personne.

## Ce qu'il faut retenir

Phew ! C'était long !

WordPress est littéralement rempli d'appels aux fonctions `apply_filters()` et `do_action()`. C'est-à-dire qu'il met à notre disposition des centaines de hooks à des centaines de moments différents de son déroulement pour que nous puissions modifier ou traiter des données à notre sauce, afficher du code HTML supplémentaire ou remplacer le code censé s'afficher, etc.

Rares sont les fonctions de WordPress ne nous laissant pas la main à un moment ou à un autre.

La majorité des extensions bien codées utilisent aussi `apply_filters()` et `do_action()` en de multiples endroits pour nous permettre de modifier leur comportement et pour que d'autres extensions puissent s'y greffer.

Comment croyez-vous que WooCommerce et toutes ses extensions fonctionnent ? Woocommerce se hooke sur WordPress, et les extensions pour WooCommerce se greffent sur WooCommerce !

En codant vos propres extensions comme WordPress et en ajoutant des hooks dans vos développements, vous permettez aux autres développeurs utilisant vos extensions de les étendre !

Je vous assure que si vous avez compris comment fonctionnent `add_filter()`, `add_action()`, `apply_filters()` et `do_action()`, vous avez compris comment vous greffer sur WordPress ou sur ses extensions, à n'importe quel moment.

Maintenant, la difficulté restante est de trouver le bon hook pour faire ce dont vous avez besoin !

Rassurez-vous, c'est l'objectif de cet ebook ! Dans les prochains chapitres, vous allez apprendre à faire tous les développements courants pour WordPress, je vais vous montrer tous les hooks les plus utiles, mais vous allez aussi voir comment "creuser" dans le code pour y trouver le hook dont vous avez besoin.

Quand vous saurez faire ça, plus rien ne vous fera peur ! Vous serez autonomes, vous saurez trouver où vous hooker pour effectuer telle ou telle action et pourrez développer toutes les fonctionnalités nécessaires pour vos projets.

Prêts ? Go.

# Comment créer un thème enfant

Dans le premier chapitre sur la Plugin API, on a vu comment créer une extension, et cela va nous servir tout au long de ce guide. Dans la même veine, on va souvent intervenir sur notre thème, donc on va voir tout de suite ce que sont les thèmes enfants, pourquoi ils sont indispensables, comment tout cela fonctionne et comment en créer un.

Dans à peu près 100% des cas, vous aurez besoin de personnaliser votre thème, donc vous aurez absolument besoin de créer un thème enfant. Donc autant savoir le faire correctement tout de suite, pour vous éviter des crises de nerfs plus tard.

## Comment les mises à jour peuvent ruiner votre site

Si utilisez un thème du répertoire officiel de WordPress, alors vous avez peut-être constaté que vous avez parfois des notifications de mises à jour. Cela signifie simplement que l'auteur a mis à jour son thème, en corrigeant une faille de sécurité, en ajoutant un peu de CSS pour corriger un souci d'affichage ou en ajoutant une fonctionnalité. Puis il l'a réuploadé sur le répertoire officiel, et WordPress a comme par magie découvert que votre thème pouvait être mis à jour.

Ce serait dommage de ne pas profiter de cette nouvelle version améliorée ou corrigée, non ?

Mais si vous avez eu la mauvaise idée de modifier vos fichiers de thème directement — par exemple, si vous avez copié-collé un snippet de code trouvé sur internet dans votre fichier `functions.php` — vous allez avoir la surprise de voir que votre modification a disparu !

Car en mettant à jour votre thème, WordPress a simplement écrasé votre ancien thème (avec vos modifications) par le nouveau ! Eh oui !

Ici je parle des thèmes du répertoire, mais c'est aussi vrai pour les thèmes premium achetés. Si vous les mettez à jour de façon automatisée alors vous aurez le même souci.

## Qu'est-ce qu'un thème enfant ?

Un thème enfant va simplement vous éviter les crises de nerfs après mise à jour. C'est un thème qui référence un autre thème (son parent) et qui sera activé à sa place.

Il va hériter de toutes les fonctionnalités de son parent. Mais il peut aussi contenir ses propres fonctionnalités, styles et modèles de page. C'est juste que tout ce qui n'est pas spécifique à l'enfant sera hérité du parent. C'est aussi simple que ça.

## Comment créer notre thème enfant

C'est assez simple. Pour travailler sur nos recettes, on va créer un enfant de Twenty Nineteen, le thème par défaut de WordPress.

Commencez par ouvrir votre dossier des thèmes dans votre éditeur de code. Dans une installation classique de WordPress, les thèmes sont situés dans `wp-content/themes`. Vous trouverez un dossier nommé `twentynineteen` si vous avez le thème installé. Sinon, téléchargez-le via l'interface d'administration des thèmes.

Créez un dossier `twentynineteen-child` pour notre thème enfant. C'est dans ce dossier que l'on va travailler. Vous pouvez choisir n'importe quel nom pour votre thème, mais par convention, on ajoute simplement le suffixe `-child` au nom du dossier du thème parent.

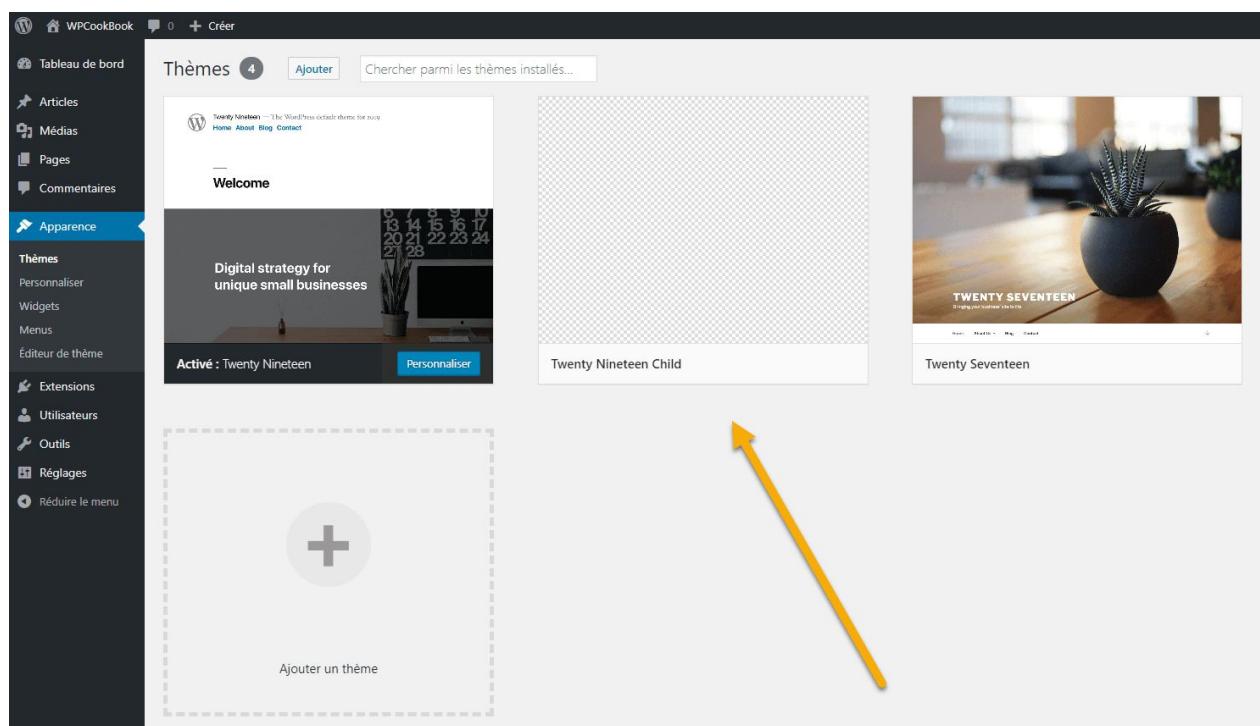
Pour que WordPress reconnaise notre thème, il faut ajouter un fichier `style.css` qui contient une entête permettant de l'identifier et de fournir d'autres métadonnées.

Créez donc un fichier `style.css` à la racine de votre dossier, et copiez-collez l'entête suivante.

```
// in style.css
/*
Theme Name: Twenty Nineteen Child
Template: twentynineteen
*/
```

Theme Name est le nom du thème enfant, et avec Template vous indiquez à quel parent vous faites référence.

Et voilà ! Notre thème enfant apparaît dans l'administration de WordPress !



Notre thème est disponible dans l'administration !

Allez-y, vous pouvez l'activer !

[Aller au contenu](#)

# WPCookBook

Apprenez à développer pour WordPress

- [Plus](#)
  - [Retour](#)
  - [Page](#)
  - [Article](#)
  - [Catégorie](#)

## Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

Publié par [vincen](#) le [septembre 12, 2019](#) Publié dans [Uncategorized](#) Un commentaire sur Hello world! [Modifier Hello world!](#)  
Rechercher :

## Articles récents

- [Hello world!](#)

## Commentaires récents

- [A WordPress Commenter](#) dans [Hello world!](#)

## Archives

- [septembre 2019](#)

Pour l'instant, ce n'est pas très joli.

## Charger les styles

À ce stade, le thème fonctionne. C'est-à-dire qu'il utilise les bons modèles de page et génère l'HTML correspondant, mais les styles du thème parent ne sont simplement pas chargés.

Ajoutez un fichier `functions.php` à la racine du thème, et ajoutez-y ce snippet :

```
<?php
// in functions.php
add_action( 'wp_enqueue_scripts', 'twentynineteen_child_scripts' );
/**
 * Loads parent styles
 */
function twentynineteen_child_scripts(){
    wp_enqueue_style( 'parent-styles', get_parent_theme_file_uri( 'style.css' ) );
}
```

Vous vous souvenez du chapitre sur les hooks (J'espère quand même !) ? Ici, la fonction `twentynineteen_child_scripts()` se hooke au moment où WordPress charge les ressources (CSS et JS) nécessaires sur le devant du site (le hook `wp_enqueue_scripts`).

Elle utilise la fonction `wp_enqueue_style()` pour charger la feuille de styles du parent, en allant la chercher grâce à la fonction `get_parent_theme_file_uri()`, qui renvoie l'url du fichier dans le thème parent.

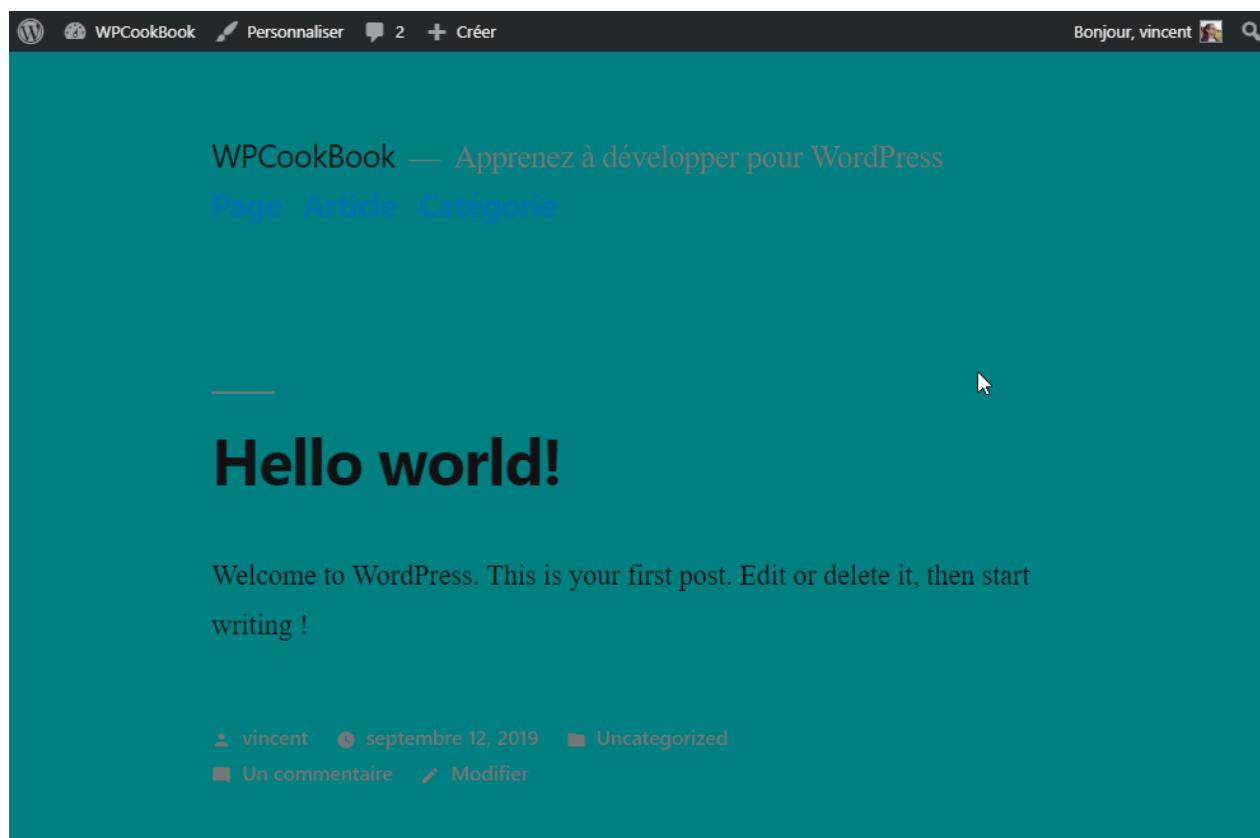
Note : pas de panique si vous ne connaissez pas ces fonctions. On va en parler un peu plus loin et apprendre à bien naviguer dans les dossiers de nos thèmes et extensions pour charger les bons fichiers. Aussi, on verra comment charger le JS et CSS correctement. Suivez simplement et faites-moi confiance, on verra tous les détails en temps voulu.

Le fichier `functions.php` est automatiquement reconnu et chargé par WordPress ce qui est plutôt pratique, et le fichier `functions.php` du thème parent est lui aussi chargé après celui de l'enfant.

Dans ce fichier (`functions.php` du parent), la feuille de style du thème actif est chargée en utilisant `get_stylesheet_uri()` (qui renvoie l'url du fichier `style.css` du thème actif, donc du thème enfant dans notre cas.)

C'est pourquoi avec `twentynineteen`, la feuille de styles de l'enfant est directement chargée !

Donc vous pouvez directement ajouter vos styles personnalisés dans `style.css` de votre thème enfant et ça marche ! Testez simplement en ajoutant ceci : `body { background-color: teal; }`



Au moins, nos styles fonctionnent.

Voilà. Ce n'est pas très joli, mais on a pu vérifier que nos styles enfants fonctionnent.

Donc, pour résumer un peu ce qui se passe en coulisses :

- Le fichier `functions.php` du thème actif (l'enfant) est chargé.
- Notre fonction est hookée pour charger les styles du parent.
- Le thème enfant fait référence au parent `twentynineteen`. Donc son fichier `functions.php` est lui aussi chargé.
- Celui-ci hooke le chargement de la feuille de styles du thème actif, donc de l'enfant.

Au final, on a bien les styles du parent chargés avant ceux de l'enfant. Tout va bien.

## Ca ne marche pas !

Si les styles de votre thème enfant ne sont pas pris en compte, c'est peut-être parce que le thème parent charge ses propres styles, sans regarder ce qu'il se passe chez l'enfant.

Si le thème parent utilise la fonction `get_template_directory_uri()` ou `get_parent_theme_file_uri()` pour aller chercher les styles, il ne va trouver que les siens, car cette fonction ne regarde pas dans le thème enfant !

Dans le fichier `functions.php` de l'enfant, il faut donc charger les styles de l'enfant, car ceux du parent sont déjà chargés ! On va donc plutôt utiliser ceci :

```
// in functions.php
<?php
add_action( 'wp_enqueue_scripts', 'twentynineteen_child_scripts' );
/**
 * Loads child styles
 */
function twentynineteen_child_scripts(){
    wp_enqueue_style( 'child-styles', get_stylesheet_uri(), array( 'twentynineteen-style' ) );
}
```

Ici, on charge les styles de l'enfant, en s'assurant que les styles du parent soient bien déclarés comme une dépendance (3e paramètre de la fonction `wp_enqueue_style()`), car il faut que les styles de l'enfant soient chargés après ceux du parent. En déclarant une dépendance, WordPress s'occupe tout seul de les charger dans le bon ordre.

Pour trouver l'identifiant de la feuille de style du parent, on peut soit ouvrir le fichier `functions.php` du parent et y chercher comment les styles sont déclarés, ou on peut aussi inspecter la source, mais attention, WordPress ajoute le suffixe `-css` au nom de la ressource.

```

30      <link rel='stylesheet' id='wp-block-library-css' href='http://wpcookbook.local/wp-
31 includes/css/dist/block-library/style.min.css?ver=5.3' type='text/css' media='all' />
32 <link rel='stylesheet' id='wp-block-library-theme-css' href='http://wpcookbook.local/wp-
33 includes/css/dist/block-library/theme.min.css?ver=5.3' type='text/css' media='all' />
34 <link rel='stylesheet' id='parent-styles-css' href='http://wpcookbook.local/wp-
content/themes/twentynineteen/style.css?ver=5.3' type='text/css' media='all' />
35 <link rel='stylesheet' id='twentynineteen-style-css' href='http://wpcookbook.local/wp-
content/themes/twentynineteen-child/style.css?ver=1.0' type='text/css' media='all' />
36 <link rel='stylesheet' id='twentynineteen-print-style-css' href='http://wpcookbook.local/wp-
content/themes/twentynineteen/print.css?ver=1.0' type='text/css' media='print' />
37 <link rel='https://api.w.org/' href='https://wpcookbook.local/wp-json/' />           ↗
38 <link rel="EditURI" type="application/rsd+xml" title="RSID"
39 href="https://wpcookbook.local/xmlrpc.php?rsd" />
40 <link rel="wlwmanifest" type="application/wlwmanifest+xml" href="http://wpcookbook.local/wp-
includes/wlwmanifest.xml" />
41 <meta name="generator" content="WordPress 5.3" />
42 </head>

```

Code source du thème enfant.

On voit bien dans le code source que les styles sont chargés dans le bon ordre.

Si vous rencontrez un souci, posez-vous simplement deux questions:

- Est-ce que la feuille de style de mon thème enfant est bien chargée ?
- Si oui, est-ce qu'elle l'est après celle du parent ?

## Personnaliser un modèle de page

Votre thème est bien installé et les styles fonctionnent. Maintenant, comment faire vos personnalisations ?

Le thème enfant hérite tout de son parent. Quand WordPress cherche un modèle de page, il va d'abord le chercher dans l'enfant, et s'il ne le trouve pas, il va chercher dans le parent.

La manière la plus simple de personnaliser un modèle est donc de dupliquer le modèle parent dans votre enfant et de le modifier.

Copiez-collez le fichier header.php de twenty-nineteen dans twenty-nineteen-child, et modifiez-le pour ajouter simplement un titre bidon.

```

<?php
/**
 * The header for our theme
 *
 * This is the template that displays all of the <head> section and everything up until <div id="content">
 *
 * @link https://developer.wordpress.org/themes/basics/template-files/#template-partials
 *
 * @package WordPress
 * @subpackage Twenty_Nineteen
 * @since 1.0.0
 */
?><!doctype html>
<html <?php language_attributes(); ?>>
<head>
    <meta charset=<?php bloginfo( 'charset' ); ?>" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="profile" href="https://gmpg.org/xfn/11" />
    <?php wp_head(); ?>
</head>

<body <?php body_class(); ?>>

    ...

    <div class="site-branding-container">
        <?php get_template_part( 'template-parts/header/site', 'branding' ); ?>
    </div><!-- .site-branding-container -->

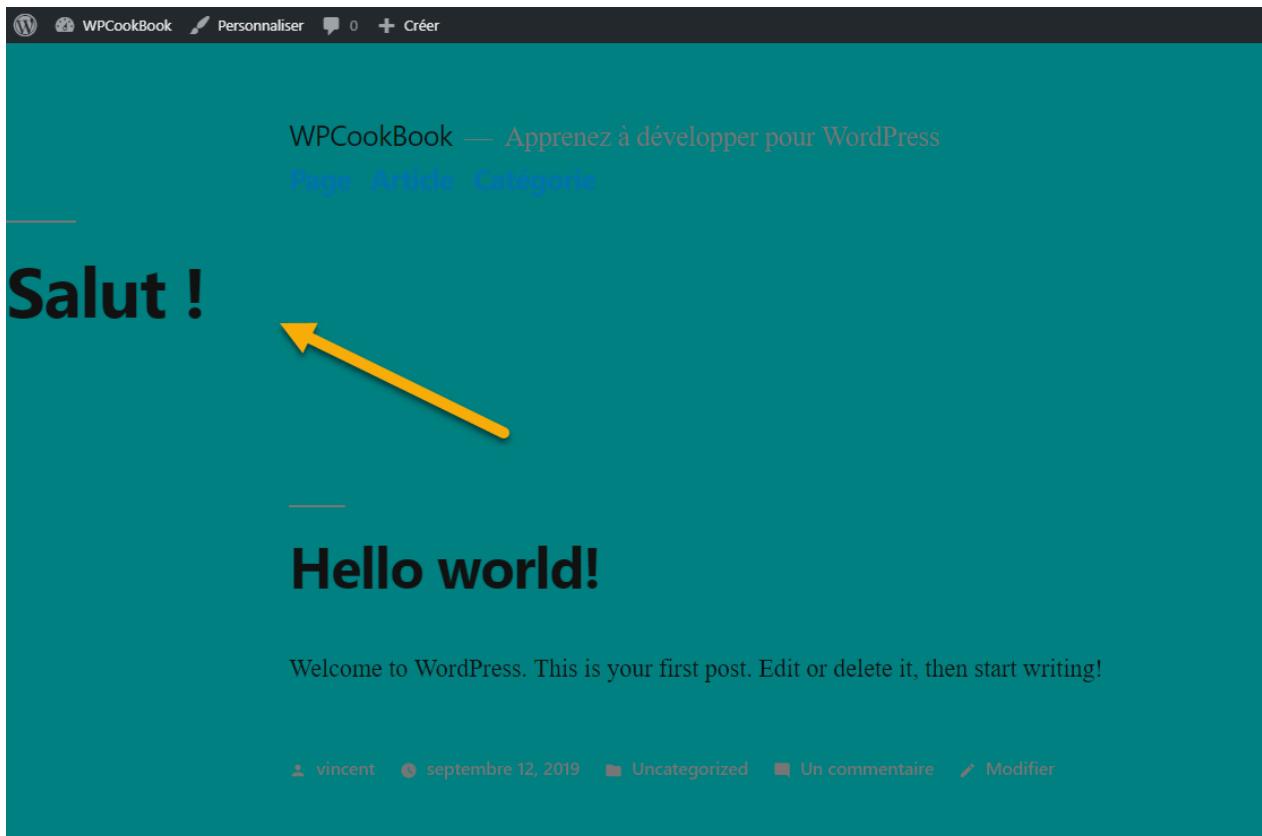
    <h1>Salut !</h1>

    <?php if ( is_singular() && twentynineteen_can_show_post_thumbnail() ) : ?>
        ...
    <?php endif; ?>

    ...

```

J'ai juste ajouté un `<h1>Salut !</h1>` après la ligne 31, l'objectif étant juste de tester si notre fichier est bien chargé en lieu et place de celui du parent.



Y avait-il vraiment besoin d'une flèche ?

Maintenant, vu que c'est toujours le fichier `header.php` du thème enfant qui va être utilisé, je vais avoir ce titre sur toutes mes pages.

Remarquez que le `header` appelle un modèle partiel (un « template part ») nommé `site-branding` dans le dossier `template-parts/header/` à l'aide de la fonction `get_template_part()`, ligne 30.

Ce qu'il faut savoir, c'est que cette fonction vérifie aussi d'abord si le fichier demandé existe dans l'enfant !

Donc si votre thème fait bon usage de cette fonction, vous pouvez aussi dupliquer les `template-parts` dans votre thème enfant pour les personnaliser. La seule contrainte est de respecter la structure des dossiers, pour que la fonction puisse retrouver le bon fichier au bon endroit.

Créez un dossier `template-parts/` avec un sous-dossier `header/`. Puis copiez-collez y le fichier `site-branding.php` du thème parent.

Ce fichier est responsable de l'affichage de la barre de navigation. Par exemple, si vous voulez enlever le slogan du site, il vous suffit de commenter quelques lignes ou de les supprimer.

```
<?php
/**
 * Displays header site branding
 *
 * @package WordPress
 * @subpackage Twenty_Nineteen
 * @since 1.0.0
 */
?>
<div class="site-branding">

    ...

<?php
// $description = get_bloginfo( 'description', 'display' );
// if ( $description || is_customize_preview() ) :
// ?>
<!-- <p class="site-description">
    <?php // echo $description; ?> -->
<!-- </p> -->
<!-- <?php //endif; ?> -->
...

```

En commentant les lignes à partir de `$description = ...`, la description n'apparaît plus.

## Complétons l'entête de notre thème

Pour fournir plus d'informations sur notre thème dans l'administration, on va compléter l'entête du fichier `style.css`. Voici un exemple plus complet d'entête.

```
/*
Theme Name: Twenty Nineteen Child
Template: twentynineteen
Theme URI: https://example.com/themes/twentynineteen-child/
Author: Vincent Dubroeucq
Author URI: https://vincentdubroeucq.com/
Description: This is a simple child theme used throughout all the examples given in the WPCookBook guide.
Version: 1.0
Requires at least: WordPress 4.9.6
License: GNU General Public License v2 or later
License URI: http://www.gnu.org/licenses/gpl-2.0.html
Text Domain: twentynineteen-child
Domain Path: /languages
Tags: one-column, flexible-header, accessibility-ready, custom-colors, custom-menu, custom-logo, editor-style, featured-images, footer-widgets, rtl-language-support, sticky-post, threaded-comments, translation-ready

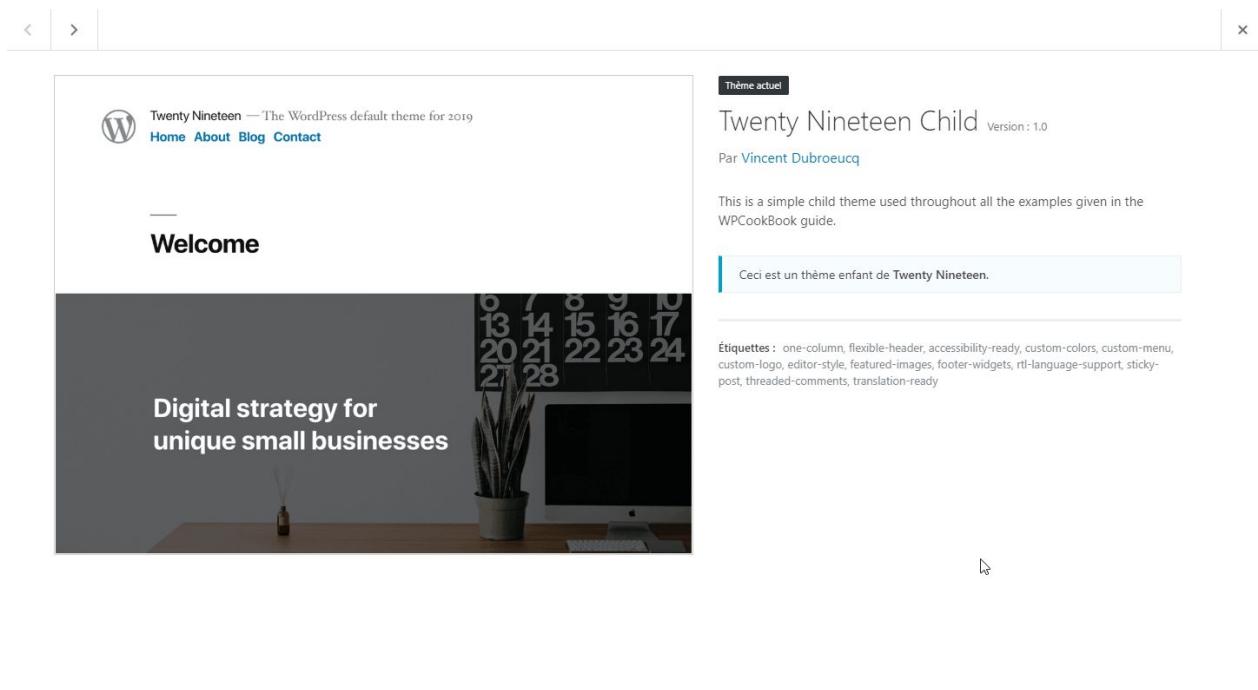
Add copyrights and other attributions here.
*/
```

- **Theme URI** est l'adresse de la page d'accueil du thème. Si vous publiez votre thème sur le répertoire officiel de WordPress, alors cette url doit pointer vers [https://wordpress.org/themes/\\${slug}](https://wordpress.org/themes/${slug})
- **Author**: Votre nom. Vous pouvez utiliser votre nom d'utilisateur sur<https://wordpress.org> si vous le souhaitez.
- **Author URI**: L'adresse de votre site web. Si vous l'indiquez, alors votre nom sera cliquable dans l'administration.
- **Description**: C'est la description qui apparaît dans l'administration.
- **Version**: Le numéro de version de votre thème. Il est fortement recommandé d'utiliser le format x.y.z, où x est la version majeure, y la version mineure, et z le numéro de correctif.
- **Requires at least**: Version minimale de WordPress pour que le thème fonctionne. Si votre thème utilise des fonctions récentes de WordPress qui ne sont donc pas disponibles dans d'anciennes versions, il vaut mieux le signaler ici.
- **Licence**: La licence de votre thème. Ce sera très probablement la même que WordPress, donc GPL.
- **Licence URI**: URL à laquelle on peut lire la licence. Vous pouvez aussi inclure un fichier `licence.txt` qui contient le texte de la licence.
- **Text Domain** : C'est un mot-clé associé aux traductions de votre thème. Pas de panique, on verra ça en détails dans [internationaliser ses développements](#).
- **Domain Path** : C'est le chemin vers les traductions.
- **Tags**: Ce sont les mots-clés associés à votre thème correspondants aux filtres sur <https://wordpress.org/themes>. Vous pouvez trouver la liste des tags sur <https://make.wordpress.org/themes/handbook/review/required/theme-tags/>

Techniquement, seul **Theme Name** est requis pour qu'un thème puisse être activé et **Template** pour qu'un thème soit déclaré comme thème enfant, et hérite donc des fonctionnalités d'un autre thème.

Pour faire plus joli, on peut aussi inclure un fichier `screenshot.png` ou `screenshot.jpg` à la racine du site. Ce fichier devrait faire 1200px par 900px pour s'afficher au mieux dans l'administration de WordPress et sur <https://wordpress.org/themes>. Notez qu'un screenshot est obligatoire si vous publiez votre thème sur le répertoire officiel.

Pour faire simple, on va copier celui du thème parent `twentynineteen`.



Avec les informations complètes, c'est beaucoup mieux, non ?

## Ce qu'il faut retenir

Pour le moment, ce qu'il faut retenir, c'est :

- Créer un thème enfant est une des premières choses à faire quand on travaille sur un site !
- Un thème enfant hérite tout de son parent.
- Il suffit de créer un dossier du même nom que le parent, avec le suffixe `-child`, y ajouter un fichier `style.css` et `functions.php`
- Il faut faire attention à bien charger les styles du parent, puis de l'enfant dans `functions.php`
- On duplique les modèles que l'on veut personnaliser, en respectant la même organisation des fichiers.

**Note importante :** A partir de maintenant, tous les exemples de ce guide utiliseront ce thème enfant de *twentynineteen*. Donc si on ne travaille pas dans une extension, mais dans le thème, on va travailler dans le thème enfant que l'on vient juste de créer ! Ce qui veut dire que si je vous propose d'ajouter une fonction dans `functions.php`, je parle du thème enfant de *twentynineteen*.

## La suite

Dans ce chapitre, on a volontairement survolé certains aspects importants. On n'a pas parlé en détails de la hiérarchie des templates, des fichiers du thème, etc...

Mais maintenant, vous êtes prêts pour attaquer le reste de ce guide : vous avez un thème enfant, et vous savez créer des extensions. On verra dans les détails comment fonctionnent les thèmes dans le chapitre Comprendre les thèmes WordPress.

Les chapitres suivants de cette section Fondements sont assez théoriques, mais ils permettent de mieux comprendre comment fonctionne WordPress dans les coulisses.

Les recettes sont classées en deux grandes sections : Thèmes et Extensions, selon leur utilisation qui est plutôt du domaine des thèmes ou du domaine des extensions. Aussi, elles sont organisées plus ou moins selon leur complexité ou selon certains prérequis.

Par exemple, c'est une bonne idée de comprendre le système des fichiers du thème avant de personnaliser la boucle. Aussi, déclarer et gérer une metabox dans l'administration est plus complexe que créer un code court, donc le chapitre vient plus tard.

A partir d'ici, vous avez plusieurs possibilités:

- Soit vous voulez du concret, de l'action ! Rendez-vous alors directement à la recette qui vous intéresse.
- Soit vous voulez apprendre en détails comment fonctionne WordPress. Lisez attentivement le reste de cette section "Fondements", puis rendez-vous à la section Thème et lisez dans l'ordre.
- Si vous êtes vainqueur, rendez-vous à la section Conclusion.

# Domaine des thèmes et des extensions

Quand vous devez développer une nouvelle fonctionnalité pour votre site ou le site d'un client, il faut vous poser la question suivante :

— Est-ce que ma fonctionnalité est plutôt du ressort du thème ou d'une extension ?

Quand on parcourt Internet à la recherche du snippet dont on a besoin, souvent les articles nous disent de coller le snippet dans le fichier `functions.php` du thème enfant.

Ok, c'est rapide et c'est chargé automatiquement. Donc on est sûr que notre code sera exécuté.

Mais le souci c'est que ce fichier `functions.php` peut rapidement être rempli de fonctions disparates, agissant sur tout et rien sur le site. Et si vous changez de thème, il faudra porter toutes ces fonctionnalités vers le nouveau thème.

Tout d'abord, il faut savoir que ce fichier n'est pas un fourre-tout ! On y déclare les fonctionnalités du thème, et quand je parle de fonctionnalités, je veux parler des fonctionnalités de WordPress relatives à l'affichage de notre contenu, mais surtout pas de types de contenus personnalisés, des fonctions déclenchant des envois de mails, etc...

Donc pour clarifier les choses, on va redéfinir rapidement ce que sont les thèmes et extensions, ainsi que leur rôle.

## Votre thème

Un thème est censé s'occuper uniquement de tout ce qui concerne l'affichage du contenu. Point.

Il ne doit pas déclarer de widgets, ni de types de contenu personnalisés, ni de taxonomies, ni de shortcodes, ni de blocs pour le nouvel éditeur, etc..., ni aucune fonctionnalité complexifiant l'administration du site.

Le thème structure votre contenu et lui donne les styles nécessaires.

Il va afficher votre logo, déclarer vos zones de menu, vos zones de widgets, votre zone de contenu principale, va s'occuper d'afficher vos images aux bonnes dimensions, va lister vos articles, en créant les liens vers les pages simples correspondantes et vers les archives correspondantes, va s'occuper d'afficher la pagination, d'afficher la liste des commentaires et le formulaire accompagnant, votre pied de page, etc...

Le thème n'est censé être que la cosmétique des fonctionnalités de base présentes dans WordPress.

C'est tout. Il n'est pas censé complexifier l'administration du site, donc ne devrait pas ajouter d'éléments de menu dans l'administration de WordPress, ni nécessiter l'utilisation d'une ou plusieurs extensions.

Beaucoup de thèmes premium que l'on achète sont trop complexes, ajoutent des fonctionnalités à WordPress et exigent l'utilisation de certaines extensions.

Ce n'est pas ce qu'est censé faire un thème. Ces thèmes-là sont plus des templates de sites complets ou des boîtes à outils pour tel ou tel type de besoin précis. Ils sont bien trop spécifiques.

Le souci avec ce genre de thème est que c'est tellement spécifique que ça en devient difficile à personnaliser. Beaucoup vantent des dizaines de fonctionnalités additionnelles dont on a rarement besoin. Donc on installe une usine à gaz pour pas grand chose.

Par contre, un thème peut parfaitement déclarer un bon support pour telle ou telle extension populaire, comme par exemple des modèles de page produits personnalisés pour WooCommerce. Si on utilise WooCommerce pour vendre un produit, on aura un beau template. Sinon, il y aura juste plus de travail de personnalisation.

## Les extensions

Une extension étend les fonctionnalités de WordPress.

C'est tout. Un site WordPress peut faire a, b, et c aussi. Avec telle extension, je peux faire d. C'est aussi simple que ça.

Avec une extension, on va ajouter des fonctionnalités e-commerce, un formulaire de contact, des boutons de partage, un outil de réservation de séance pour nos cours de yoga, un formulaire d'inscription pour devenir membre, etc.

Les extensions peuvent compliquer sensiblement l'interface d'administration en ajoutant de multiples menus et réglages en tout genre.

## Et les styles ?

Evidemment, une extension a souvent un effet visible sur le devant du site. Il faut bien l'afficher notre formulaire de contact.

Mais est-ce que pour autant l'extension peut charger ses propres styles pour le devant du site ? Oui, à condition de ne pas marcher sur les pieds du thème.

Quand vous créez des styles pour une extension, évitez de redéfinir les polices des textes, les marges des paragraphes, les bordures des <input />, et héritez au maximum des styles du thème.

Dans l'idéal, il ne devrait y avoir que des styles mineurs de mise en page, mais le moins possibles de styles "cosmétiques" (couleurs, typographie, etc.)

## Ni noir ni blanc

Ce n'est pas forcément simple de déterminer si tel ou tel bout de code a sa place dans un thème ou dans une extension. Mais essayez de vous poser les questions suivantes :

- Si je change de thème, est-ce que je perds la fonctionnalité ?
- Si je change de thème, est-ce que l'affichage de ma fonctionnalité sera cassé ?
- Est-ce que ma fonctionnalité implique des réglages dans l'administration ?
- De quels styles va avoir besoin tout utilisateur de mon extension ? Quels styles puis-je omettre car ils sont censé être définis dans le thème (je pense surtout aux formulaires) ?

Essayez de garder cela en tête quand vous devez développer une fonctionnalité, pour vous faciliter la maintenance plus tard. Votre futur vous ne vous sera que plus reconnaissant d'avoir mis la demande client dans une extension quand il vous demandera une refonte graphique du site.

## Les mu-plugins (extensions indispensables)

Les mu-plugins sont des extensions qui ne sont pas désactivables, et qui sont chargées avant les extensions traditionnelles.

Si vous avez besoin de certaines fonctions à la fois dans le thème et dans des extensions, cela peut être une bonne idée de les inclure dans un mu-plugin. De ce fait, elles seront toujours disponibles, qu'elles soient utilisées dans une extension ou dans votre thème.

### Ce qu'il faut retenir

- Un thème est censé s'occuper de l'affichage uniquement, et ne pas déclarer de fonctionnalité.
- Une extension ajoute des fonctionnalités.
- Une extension devrait contenir des styles minimaux pour s'appuyer au maximum sur le thème.

# Comprendre le chargement de WordPress

J'ai mentionné dans le chapitre sur la plugin API que WordPress n'était qu'un simple programme PHP.

C'est juste un gros script qui doit se charger et s'exécuter à chaque chargement de page. C'est tout.

Quand une requête est faite à votre serveur, le fichier `.htaccess` redirige tout ça vers `index.php` et c'est parti, la machine démarre à partir de ce fichier.

On ne va pas parcourir de façon linéaire le chargement de WordPress en lisant ensemble le code à partir de ce fichier. Je pense que vous pouvez tout seul lire le fichier `index.php` et suivre les `require` pour savoir quel fichier est chargé ensuite.

Cela dit, si vous faites ça vous allez vite abandonner quand vous allez voir la quantité de fichiers chargés par `wp-settings.php` !

Donc non, on ne va pas s'infliger ça.

L'objectif est simplement d'avoir une vue d'ensemble du chargement de WordPress pour pouvoir comprendre quand sont disponibles les hooks principaux, et éviter de hooker ses fonctions au mauvais moment, pour s'étonner ensuite que cela ne marche pas.

## Côté front

Donc voilà une petite roadmap assez générale des hooks principaux, dans l'ordre dans lequel ils sont rencontrés :

| Hook                           | Détails   |
|--------------------------------|---|
| <code>mu_plugins_loaded</code> | Les mu-plugins sont les premières extensions chargées. Une fois que c'est fait, ce hook est déclenché. A ce stade, les extensions "traditionnelles" ne sont pas encore chargées, donc le seul endroit où nous pouvons placer du code pour intervenir sur ce hook, c'est dans un mu-plugin ! |
| <code>plugins_loaded</code>    | Il est rencontré quand les extensions sont chargées. C'est le premier hook disponible pour les extensions.  |
| <code>after_setup_theme</code> | Il est rencontré juste après que le fichier <code>functions.php</code> du thème soit chargé. C'est le premier hook disponible pour le thème et c'est sur lui que beaucoup de fonctionnalités du thème sont déclarées, à l'aide de <code>add_theme_support()</code>                          |

| Hook               | Détails   |
|--------------------|---|
| init               | C'est un hook très utilisé, car WordPress est en grande partie chargé à ce stade. Les extensions et le thème sont chargés, et l'utilisateur est authentifié et disponible. C'est sur ce hook que l'on va déclarer nos types de contenu personnalisés et nos taxonomies, par exemple. WordPress continue à se charger car il se passe beaucoup de choses sur ce hook. Par exemple, les widgets se chargent sur ce hook et déclenchent <code>widget_init</code> . |
| wp_loaded          | Ce hook est rencontré juste après <code>init</code> . WordPress est complètement chargé, mais la requête n'a pas encore été traitée.  |
| parse_request      | Les variables de la requête HTTP entrante sont interprétées. Elles sont filtrables via le filtre <code>apply_filters( 'request', \$this-&gt;query_vars )</code> ; juste avant.  |
| send_headers       | Les entêtes HTTP sont envoyés à ce stade. Ils sont filtrables juste avant, grâce au filtre <code>apply_filters( 'wp_headers', \$headers, \$this )</code> ;  |
| parse_query        | Les variables pour la requête principale <code>WP_Query</code> sont prêtes ! La requête pour aller chercher le bon contenu n'est pas encore faite cependant.  |
| pre_get_posts      | Ce hook est très utile car il permet de manipuler les paramètres de la requête <code>WP_Query</code> avant que celle-ci ne soit effectuée. Si vous avez des paramètres à changer, que ce soit de tri, de taxonomie, de métadonnées ou autre, c'est maintenant ou jamais !   |
| wp                 | A ce stade, tout est prêt. L'objet <code>\$wp</code> est complet.   |
| template_redirect  | Ce hook est rencontré juste avant que WordPress n'aille chercher le bon modèle de page dans votre thème pour afficher le contenu. C'est le dernier moment où vous pouvez effectuer une redirection proprement. Pour filtrer le modèle de page à utiliser, vous pouvez utiliser le filtre <code>apply_filters( 'template_include', \$template )</code> ;   |
| get_header         | Ce hook est rencontré juste avant d'aller chercher le fichier <code>header.php</code> du thème.   |
| wp_enqueue_scripts | C'est sur ce hook que l'on greffe pour ajouter des styles et scripts à charger.   |
| wp_head            | On utilise ce hook pour écrire dans la balise <code>&lt;head&gt;</code> de la page.   |

| Hook                                   | Détails   |
|--|---|
| wp_print_styles et<br>wp_print_scripts | se déclenchent juste avant d'afficher les balises <link> et <script> dans <head>.   |
| the_post                               | se déclenche quand l'objet \$post est prêt. On peut donc le manipuler avant de l'afficher.  |
| pre_get_comments                       | Comme pre_get_posts, mais pour les commentaires. C'est-à-dire qu'il est déclenché juste avant d'aller chercher les commentaires.  |
| get_sidebar                            | On le trouve avant que le modèle pour la zone de widget soit chargé.  |
| get_footer                             | Comme get_header, mais juste avant d'aller chercher le pied de page du thème.   |
| wp_footer                              | Déclenché dans le pied de page, pour charger les <script> des extensions, principalement.   |
| admin_bar_menu                         | Charge le menu de la barre d'admin, si besoin.  |
| shutdown                               | On rencontre ce hook juste avant que l'exécution PHP s'arrête. C'est le dernier hook disponible. Attention ce hook se déclenche quand PHP s'arrête, donc il est forcément déclenché sur la page, mais rien ne vous assure que vous avez passé tel ou tel hook car une erreur fatale ou un die() va le déclencher. |

## Ouf !

Vous vous doutez bien que la liste a été (très) raccourcie et que toutes les actions et filtres n'ont pas été listés. Il y a des centaines de filtres et autres actions déclenchés sur lesquels on peut se greffer ! C'est juste impossible de tous les lister !

Si la liste précédente vous donne mal à la tête, voici la même liste, mais ultra allégée.

| Hook              | Détails   |
|-------------------|---|
| mu_plugins_loaded | Les mu-plugins sont chargés.                                      |
| plugins_loaded    | Extensions chargées. Premier hook disponible pour les extensions. |
| after_setup_theme | Thème chargé. Premier hook disponible pour le thème.              |
| init              | Extensions et thèmes chargés, utilisateur disponible.             |
| wp_loaded         | Juste après init. WordPress est complètement chargé.              |

| Hook               | Détails  |
|--------------------|--|
| pre_get_posts      | Juste avant d'aller en base de données chercher le contenu.            |
| wp                 | Tout est prêt.   |
| template_redirect  | On va chercher le modèle de page. Dernier moment pour une redirection. |
| get_header         | Juste avant d'aller chercher header.php.                               |
| wp_enqueue_scripts | On charge les styles et scripts.                                       |
| wp_head            | On écrit dans la balise <head> de la page.                             |
| get_footer         | Juste avant d'aller chercher le pied de page du thème.                 |
| wp_footer          | On charge les <script> en bas de page.                                 |

Ces hooks sont essentiels et couvrent une bonne partie de ce dont vous aurez besoin de manière générale.

- Besoin d'une nouvelle taxonomie ? C'est sur init.
- Besoin de modifier les publications à aller chercher ? C'est sur pre\_get\_posts.
- Besoin d'ajouter une fonctionnalité au thème ? C'est sur after\_setup\_theme.
- Besoin d'ajouter des scripts ? C'est sur wp\_enqueue\_scripts, etc.

## Un peu de fun

Pour s'amuser, on va se hooker sur un exemple simple et regarder ce dont on dispose à ce moment. On ne va pas faire du code super propre mais l'idée c'est juste d'expérimenter un peu.

Créez une extension puis activez-là. Si vous ne vous souvenez plus comment on crée une extension, relisez rapidement le 1er chapitre sur la Plugin API.

Dans notre fichier racine, ajoutez le snippet suivant :

```
// in main plugin file
add_action( 'pre_get_posts', 'wpcookbook_query' );
function wpcookbook_query( $query ){
    echo '<pre>';
    wp_die( print_r( $query ) );
}
```

Ici, on se greffe sur pre\_get\_posts, donc juste avant d'aller chercher le contenu et vu que le hook nous fournit la \$wp\_query en paramètre, on peut simplement la passer à notre fonction et l'afficher dans un <pre>, en arrêtant WordPress avec un wp\_die().

C'est loin d'être joli comme code, mais au moins cela permet d'avoir rapidement un aperçu de ce que contient une variable.

```

WP_Query Object
(
    [query] => Array
    (
        [
    )

    [query_vars] => Array
    (
        [error] =>
        [m] =>
        [p] => 0
        [post_parent] =>
        [subpost] =>
        [subpost_id] =>
        [attachment] =>
        [attachment_id] => 0
        [name] =>
        [static] =>
        [pagename] =>
        [page_id] => 0
        [second] =>
        [minute] =>
        [hour] =>
        [day] => 0
        [monthnum] => 0
        [year] => 0
        [w] => 0
        [category_name] =>
        [tag] =>
        [cat] =>
        [tag_id] =>
        [author] =>
        [author_name] =>
        [feed] =>
        [tb] =>
        [paged] => 0
        [meta_key] =>
        [meta_value] =>
        [preview] =>
        [s] =>
        [sentence] =>
        [title] =>
        [fields] =>
        [menu_order] =>
        [embed] =>
        [category__in] => Array
        (
        )
    )

    [category__not_in] => Array
    (
    )

    [category__and] => Array
    (
    )

    [post__in] => Array
    (
    )
)

```

Contenu de \$wp\_query

Du coup, par exemple, on peut dire à WordPress de ne pas afficher notre article Hello World, en modifiant les variables de la requête. On utilise `$query->set()` avec le paramètre `post__not_in` pour exclure la publication ayant pour identifiant 1, et en commentant le `wp_die()` pour voir le résultat.

```
add_action( 'pre_get_posts', 'wpcookbook_query' );
// in main plugin file
function wpcookbook_query( $query ){
    $query->set( 'post__not_in', array( 1 ) );
    // echo '<pre>';
    // wp_die( print_r( $query ) );
}
```

Sur la page d'accueil de notre blog, cela donne :



WPCookBook — Apprenez à développer pour WordPress

[Page](#) [Article](#) [Catégorie](#)

## Aucun résultat

Prêt·e à publier votre premier article ? [Lancez-vous ici.](#)

L'article Hello World a disparu de nos résultats

L'article Hello World a disparu de nos résultats, y compris dans les archives par catégorie et a aussi fait disparaître le widget Articles récents !

Vous pouvez même aller faire un tour en administration, notre article a disparu. Cela nous donne une information importante : le hook `pre_get_posts` est aussi rencontré en backend !

Si l'article a disparu, c'est parce que notre code a exclu l'article avec l'identifiant 1 dans TOUTES les requêtes, sur le devant du site ET dans l'administration.

Si l'on exclut l'article uniquement dans la requête pour le contenu principal à l'aide de la méthode `is_main_query()`, le widget Articles Récents réapparaît sur le devant du site.

```
// in main plugin file
add_action( 'pre_get_posts', 'wpcookbook_query' );
function wpcookbook_query( $query ){
    if( $query->is_main_query() ){
        $query->set( 'post__not_in', array( 1 ) );
    }
}
```

## Aucun résultat

Prêt-e à publier votre premier article ? [Lancez-vous ici.](#)

## Commentaires récents

[A WordPress Commenter](#) dans [Hello world!](#)

## Articles récents

[Hello world!](#)



## Archives

[septembre 2019](#)

L'article Hello World est bien dans nos articles récents

Par contre, il est présent dans les widgets du tableau de bord de l'administration, mais pas sur la page listant les articles. Argh ! Il faut ajouter une condition pour qu'il ne disparaisse pas dans l'administration.

```
add_action( 'pre_get_posts', 'wpcookbook_query' );
function wpcookbook_query( $query ){
    if( $query->is_main_query() && ! is_admin() ){
        $query->set( 'post__not_in', array( 1 ) );
    }
}
```

Vous voyez que WordPress nous donne beaucoup de contrôle sur énormément d'éléments sur notre site. Mais il faut être prudent ! Rassurez-vous, on verra plus en détails comment personnaliser vos requêtes de contenu plus tard.

## Côté administration

Dans l'administration de WordPress, la plupart des hooks listés plus haut se déclenchent aussi, sauf ceux réservés à l'affichage sur le devant du site.

A la place, on a des hooks correspondant pour l'administration.

| Hook                  | Détails  |
|-----------------------|--|
| mu_plugins_loaded     | Les mu-plugins sont chargés.   |
| plugins_loaded        | Extensions chargées. Premier hook disponible pour les extensions.                      |
| after_setup_theme     | Thème chargé. Premier hook disponible pour le thème.                                   |
| init                  | Extensions et thèmes chargés, utilisateur disponible.                                  |
| wp_loaded             | Juste après init. WordPress est complètement chargé.                                   |
| admin_menu            | Utilisé pour ajouter ou modifier des éléments du menu de l'administration de WordPress |
| admin_init            | L'administration est prête.  |
| load-{\$page}         | Déclenché quand une certaine \$page est chargée.                                       |
| admin_enqueue_scripts | On charge les styles et scripts.   |
| admin_head            | On écrit dans la balise <head> de la page.   |
| admin_notices         | Pour ajouter des notices.  |
| admin_footer          | Pour écrire des <script> en bas de page de l'administration                            |

Selon le contenu de la page, d'autres hooks vont être déclenchés. Par exemple, la page d'édition des articles va déclencher de nombreux autres hooks qui vont nous permettre d'ajouter nos metaboxes, d'ajouter des blocs pour l'éditeur, etc.

Aussi, chaque processus en backend a sa série de hooks. Par exemple, sauvegarder une publication déclenche toute une série d'événements sur lesquels on peut venir se greffer et intervenir.

On peut par exemple changer le titre de nos publications en "Bonjour Toto" chaque fois qu'elles sont sauvegardées! Cool, non ? Bon ok...

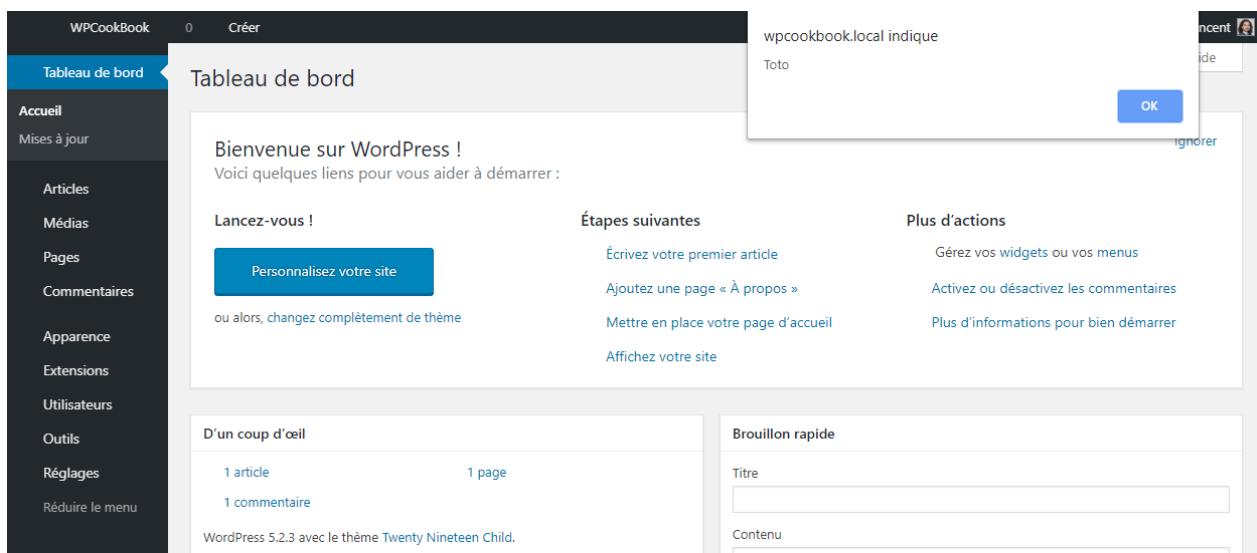
Enfin, il existe aussi des hooks dynamiques, qui nous permettent de déclencher nos fonctions au bon moment, et pas sur toutes les pages. Par exemple "admin\_footer-{\$hook\_suffix}" va permettre de charger des scripts uniquement pour la page correspondante au \$hook\_suffix.

## Encore du fun

Encore pour le fun, voici un snippet :

```
// in main plugin file
add_action( 'admin_footer', 'wpcookbook_admin_footer' );
function wpcookbook_admin_footer(){
    echo '<script>alert( \'Toto\' );</script>';
}
```

Si vous allez dans votre administration, notre balise `<script>` apparaît dans le code source, et notre script s'exécute... sur chaque page de l'administration.



Notre script s'exécute sur chaque page de l'administration.

## Ce qu'il faut retenir

- WordPress est un script PHP qui se charge de façon plus ou moins linéaire.
- Il va déclencher de nombreux hooks et certains sont communs à toutes les pages.
- Il charge d'abord les mu-plugins, puis les extensions, puis le thème.
- Chaque page a ses propres hooks.
- Chaque processus en administration a ses hooks.
- Il existe des hooks dynamiques pour plus de contrôle.

Que ce soit sur le devant du site ou dans l'administration, WordPress nous permet de toucher à presque tout, presque n'importe quand !

Tout au long de ce guide et de ses recettes, on va utiliser des tonnes de hooks. Vous aurez donc des dizaines d'exemples de ce type, et vous saurez exactement quand vous hooker, et comment.

# Comment trouver les hooks dont nous avons besoin ?

Ce chapitre est assez théorique. Cela dit, si vous avez compris l'esprit / les astuces qui sont proposées, alors vous aurez tous les outils nécessaires pour aborder n'importe quel développement de fonctionnalité, que ce soit pour étendre WordPress ou pour étendre une extension déjà existante comme WooCommerce, Learndash ou autre.

## Nous avons tout le code source

Cela paraît évident, mais nous avons tout le code presque sous les yeux. Il suffit d'ouvrir un éditeur de code et de regarder.

Pour apprendre, il ne faut surtout pas hésiter à ouvrir WordPress, et regarder ce qu'il a dans le ventre.

Le code de WordPress est bien commenté. Pour chaque hook ou fonction, on a tous les paramètres utilisés, et la valeur de retour quand la fonction retourne quelque chose.

## Nous avons toute la documentation

La documentation officielle disponible à <https://developer.wordpress.org> est très bien faite. La Code Reference est un gros moteur de recherche. Il suffit de lui donner le nom de la fonction, du hook ou de la classe que l'on recherche.

Chaque page de recherche nous donne une explication de la fonction et de ses paramètres, le code source directement (ou le lien vers Trac), dans quel fichier on la trouve, quelles autres fonctions l'utilisent, et des exemples d'utilisation pour les plus courantes. Une vraie mine d'or !

## Comment creuser dans le code de WordPress

Il y a plusieurs astuces que l'on peut utiliser pour trouver les hooks dont nous avons besoin. La plupart du temps il faudra utiliser une combinaison de ces techniques. En voici quelques unes :

### La structure des fichiers

Les développeurs sont en général logiques et organisés. Les fichiers de WordPress sont organisés en trois dossiers :

- **wp-includes** : contient la majorité des fonctions nécessaires sur le devant du site ou en administration.
- **wp-admin** : contient tous les fichiers nécessaires à l'administration de WordPress.
- **wp-content** : contient tout le contenu utilisateur, c'est-à-dire les thèmes, les extensions et les uploads, entre autres.

À la racine se trouvent les fichiers contenant les fonctions de chargement, connexion et inscription.

La majorité des extensions sont organisées aussi de façon logique et on y trouve souvent plusieurs dossiers distincts pour les fonctionnalités générales, pour les fonctionnalités du devant du site, pour l'administration, pour les templates, pour les fichiers CSS et JS, etc.

Observez simplement et attentivement les noms des dossiers et des fichiers peut grandement vous aider à trouver ce que vous cherchez.

## La recherche dans la documentation

Si vous cherchez un hook ou une fonction, mais ne savez pas trop par où commencer, parfois une simple recherche dans la Code reference peut vous mettre sur la voie.

Cochez la case correspondante au type de résultat souhaité (fonction, hook, ou classe) et entrez votre mot clé. Par exemple, recherchez tous les hooks contenant le mot post. Vous serez surpris du nombre de résultats !

## La recherche dans le code source

Imaginons que l'on ait besoin de trouver où une certaine fonction est hookée. C'est-à-dire qu'on a le nom de la fonction mais on ne sait pas sur quel hook elle est utilisée.

En gros, on cherche la ou les ligne(s) :`add_action( 'hook' , 'ma_fonction' )` ou `add_filter( 'hook' , 'ma_fonction' )`, hook étant le hook recherché.

En utilisant les fonctionnalités de recherche de l'éditeur de code, on peut chercher 'ma\_fonction', entre single quotes, car c'est comme ça qu'elle apparaîtrait dans un appel à `add_action()` ou `add_filter()`.

De la même façon, si on veut trouver les fonctions hookées sur un certain hook, on peut chercher les appels à `add_action()` ou `add_filter()`, en ciblant `add_action( 'hook'` ou `add_filter( 'hook'` avec un espace après la parenthèse ouvrante, car c'est la convention d'écriture.

Faites le test en cherchant tout ce qui est hooké sur `the_content` dans le cœur de WordPress.

Vous pouvez aussi trouver le code d'une fonction en cherchant `function ma_fonction()`. Ce n'est pas forcément très utile pour WordPress dans le sens où la Code Reference peut nous trouver la définition de la fonction, mais pour certaines extensions non-documentées, c'est utile.

Enfin, pour découvrir ce qui est hooké sur un certain hook, on peut aussi utiliser l'astuce donnée au premier chapitre de ce guide :

```
add_filter( 'the_content', 'wpcookbook_content' );
/**
 * Lists every function hooked on the current hook (the_content)
 *
 * @param string $content Content of the post.
 */
function wpcookbook_content( $content ){
    global $wp_filter;
    $html = '<pre>' . print_r( $wp_filter['mon_hook'], true ) . '</pre>';
    return $content . $html;
}
```

Bim, on affiche le tableau des fonctions hookées. Un peu brutal, mais ça marche!

## Remonter la stack

Le principe de “Remonter la stack” est de partir de ce que l’on sait, et de remonter l’enchaînement des fonctions jusqu’à ce que l’on tombe sur le hook dont on a besoin.

Prenons un exemple basique. Imaginons que vous souhaitez afficher un bouton de partage Twitter en bas du contenu de vos articles. Créez une extension pour faire ce petit exercice. Si vous ne vous souvenez plus comment créer une extension, jetez un œil au chapitre sur la Plugin API.

Pour comprendre comment ajouter du contenu sur les pages, il faut d’abord comprendre comment le contenu est affiché. C’est le thème qui est responsable de l’affichage du contenu, donc on va ouvrir le fichier `single.php` du thème et y jeter un œil.

Nous verrons en détails comment fonctionnent les thèmes un peu plus tard. Pour le moment, faites-moi confiance et concentrons-nous sur le fichier `single.php` du thème `twentynineteen`, car c’est ce fichier qui est responsable de l’affichage des pages articles simples.

```
<?php
/**
 * The template for displaying all single posts
 *
 * @link https://developer.wordpress.org/themes/basics/template-hierarchy/#single-post
 *
 * @package WordPress
 * @subpackage Twenty_Nineteen
 * @since 1.0.0
 */

get_header();
?>

<section id="primary" class="content-area">
    <main id="main" class="site-main">

        <?php
            /* Start the Loop */
            while ( have_posts() ) :
                the_post();

                get_template_part( 'template-parts/content/content', 'single' );

                if ( is_singular( 'attachment' ) ) {
                    // Parent post navigation.
                    the_post_navigation(
                        array(
                            /* translators: %s: parent post link */
                            'prev_text' => sprintf( __( '<span class="meta-nav">Published
in</span><span class="post-title">%s</span>', 'twentynineteen' ), '%title' ),
                        )
                    );
                } elseif ( is_singular( 'post' ) ) {
                    // Previous/next post navigation.
                    the_post_navigation(
                        array(
                            'next_text' => '<span class="meta-nav" aria-hidden="true">' . __( 'Next Post', 'twentynineteen' ) . '</span> ' .
                        )
                    );
                }
            
```

```

        '<span class="screen-reader-text">' . __( 'Next post:',
'twentynineteen' ) . '</span> <br/>' .
        '<span class="post-title">%title</span>',
'prev_text' => '<span class="meta-nav" aria-hidden="true">' . __(
'Previous Post', 'twentynineteen' ) . '</span>' .
        '<span class="screen-reader-text">' . __( 'Previous post:',
'twentynineteen' ) . '</span> <br/>' .
        '<span class="post-title">%title</span>',
)
);
}

// If comments are open or we have at least one comment, load up the comment
template.
if ( comments_open() || get_comments_number() ) {
    comments_template();
}

endwhile; // End of the loop.
?>

</main><!-- #main -->
</section><!-- #primary -->

<?php
get_footer();

```

Après les balises `<section>` et `<main>`, la boucle de WordPress commence. L'article est préparé grâce à `the_post()`, un modèle partiel (ou template-part) est appelé avec `get_template_part()`, puis on a une structure conditionnelle qui affiche la navigation, et enfin les commentaires sont affichés.

Il n'y a pas de balisage concernant notre article. Donc, il faut aller jeter un oeil à ce modèle partiel.

En jetant un oeil à la fonction `get_template_part()` dans la Code Reference (<https://developer.wordpress.org/reference/>), on apprend qu'elle prend simplement le nom d'un fichier PHP et le charge. Le deuxième paramètre est optionnel et est concaténé au premier avec un “-”. Elle va donc chercher un fichier `content-single.php`, dans le dossier `template-parts/content`.

Le voici :

```

<?php
/**
 * Template part for displaying posts
 *
 * @link https://developer.wordpress.org/themes/basics/template-hierarchy/
 *
 * @package WordPress
 * @subpackage Twenty_Nineteen
 * @since 1.0.0
 */
?>

<article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
    <?php if ( ! twenty_nineteen_can_show_post_thumbnail() ) : ?>
        <header class="entry-header">
            <?php get_template_part( 'template-parts/header/entry', 'header' ); ?>
        </header>
        <?php endif; ?>

        <div class="entry-content">
            <?php
                the_content(
                    sprintf(
                        wp_kses(
                            /* translators: %s: Name of current post. Only visible to screen readers */
                            __( 'Continue reading' . "%s" . '' ),
                            'twenty_nineteen' ),
                        array(
                            'span' => array(
                                'class' => array(),
                            ),
                        )
                    ),
                    get_the_title()
                );
            wp_link_pages(
                array(
                    'before' => '<div class="page-links">' . __( 'Pages:', 'twenty_nineteen' ),
                    'after'  => '</div>',
                )
            );
        ?>
    </div><!-- .entry-content -->

    <footer class="entry-footer">
        <?php twenty_nineteen_entry_footer(); ?>
    </footer><!-- .entry-footer -->

    <?php if ( ! is_singular( 'attachment' ) ) : ?>
        <?php get_template_part( 'template-parts/post/author', 'bio' ); ?>
    <?php endif; ?>

</article><!-- #post-<?php the_ID(); ?> -->

```

Le commentaire en entête nous annonce qu'on est bien dans le bon fichier.

Vous pouvez voir qu'on a une balise `<div class="entry-content">`, et que la fonction

`the_content()` est appelée. C'est assez explicite, et il n'y a pas forcément besoin de connaître les fonctions en détails pour comprendre ce qu'elles font.

Donc on sait que c'est cette fonction qui affiche le contenu. Maintenant, allons voir dans la documentation de WordPress pour comprendre comment le contenu est affiché.

Cherchez la fonction `the_content` dans la Code reference (<https://developer.wordpress.org/reference/>).

```
function the_content( $more_link_text = null, $strip_teaser = false ) {
    $content = get_the_content( $more_link_text, $strip_teaser );

    /**
     * Filters the post content.
     *
     * @since 0.71
     *
     * @param string $content Content of the current post.
     */
    $content = apply_filters( 'the_content', $content );
    $content = str_replace( ']]>', ']]>', $content );
    echo $content;
}
```

La fonction récupère le contenu avec `get_the_content()` et l'affiche avec un `echo`, simplement. Mais on peut aussi voir qu'il y a un appel à `apply_filters()`. Ce qui signifie qu'il y a un `hook` à cet endroit, et que l'on peut filtrer le contenu juste avant de l'afficher ! En plus, le filtre est documenté dans le code source avec ses paramètres, ce qui est plutôt cool.

Dans notre extension, hookons une fonction sur `the_content`.

```
// in main plugin file
add_filter( 'the_content', 'wpcookbook_twitter_share_button', 10, 1 );
/**
 * Adds a Twitter share button after the content on single posts.
 *
 * @param string $content HTML content for the post
 * @return string $content
 */
function wpcookbook_twitter_share_button( $content ){
    if( is_single() ){
        $content .= '<div class="wpcookbook-share">Est-ce que ça marche ?</div>';
    }
    return $content;
}
```

On enregistre la fonction `wpcookbook_twitter_share_button()` pour s'exécuter à cet endroit. Elle reçoit le contenu `$content`, que l'on modifie uniquement si l'on est sur une page article simple (ce que l'on vérifie avec `is_single()`) en lui ajoutant une simple `<div>` pour le moment.

Mais surtout, on n'oublie pas de retourner le contenu !Et on le fait hors de la structure conditionnelle, sinon, on va avoir un souci sur tous les autres types de page car le contenu ne sera jamais retourné !

---

# Hello world!

• vincent • septembre 12, 2019 • Un commentaire • Modifier

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

Est-ce que ça marche ?



• vincent • septembre 12, 2019 • Uncategorized • Modifier

Notre contenu est bien concaténé au contenu de l'article.

Voilà ! On est au bon endroit. Je vous laisse le soin de baliser votre bouton et d'implémenter la fonctionnalité de partage. Ce n'est pas le but de l'exercice ici.

Pour résumer :

- On est parti de ce que l'on savait : c'est `single.php` qui est responsable de l'affichage des articles simples.
- En lisant le code, on a vu que c'est `get_template_part()` qui va chercher et afficher le balisage de l'article.
- Dans ce template partiel, le contenu est affiché avec `the_content()`.
- La fonction `the_content()` expose un filtre du même nom.

Vous voyez l'idée ? On creuse, on creuse, on creuse en remontant les appels de fonctions jusqu'à ce qu'on tombe sur le hook dont on a besoin.

L'exemple était très simple. Souvent, il faudra creuser bien plus.

Aussi, c'est moi qui vous ai dit que `single.php` était responsable de l'affichage des articles. Mais vous auriez aussi pu le découvrir vous-même. Avec l'expérience (et les raccourcis de ce guide), votre connaissance de WordPress grandira et vous pourrez choisir de mieux en mieux vos points de départ et

trouver de plus en plus vite !

## Rechercher des indices dans le code HTML

Imaginons que vous vouliez ajouter un champ supplémentaire pour les catégories d'articles, comme un simple sous-titre par exemple.

Ce champ texte devra apparaître sur la page des catégories, dans le formulaire de création de nouvelle catégorie.

The screenshot shows the WordPress admin dashboard with the 'Catégories' (Categories) screen selected. A yellow arrow points from the browser's address bar ('wpcookbook.local/wp-admin/edit-tags.php?taxonomy=category') to the top of the page. Another yellow arrow points from the browser's developer tools (Elements tab) to the 'addtag' input field in the HTML code, which is highlighted with a blue box.

Formulaire de la page des catégories.

Si on jette un œil au code HTML dans l'inspecteur, on voit que le formulaire a un identifiant addtag. Aussi, l'url dans la barre d'adresse pointe vers wp-admin/edit-tags.php.

On va donc ouvrir le fichier wp-admin/edit-tags.php dans notre éditeur de code pour voir comment est affiché ce formulaire. Il se peut qu'il soit affiché par une simple fonction ou alors qu'un hook y soit présent et qu'une autre fonction responsable de l'affichage y soit hookée.

On y trouve d'abord un gros switch qui va traiter les actions du formulaire. Puis on trouve effectivement du code pour afficher notre formulaire et le tableau des catégories.

Le code d'affichage de notre formulaire commence à la ligne 414. Si on lit attentivement, on découvre qu'il y a plusieurs hooks pour intervenir sur le formulaire. Il y a même un hook directement dans la balise <form> pour lui ajouter des attributs !

On y trouve :

- do\_action( "{\$taxonomy}\_term\_new\_form\_tag" ) pour ajouter des attributs au <form>
- \$dropdown\_args = apply\_filters( 'taxonomy\_parent\_dropdown\_args', \$dropdown\_args, \$taxonomy, 'new' ) pour filtrer les catégories parentes à afficher

- `do_action( 'add_tag_form_fields', $taxonomy )` pour ajouter des champs aux taxonomies non-hierarchiques (par défaut, ce sont les étiquettes)
- `do_action( "{$taxonomy}_add_form_fields", $taxonomy )` pour ajouter des champs pour tout type de taxonomie, le hook étant dynamique.
- Trois hooks dépréciés qui nous disent d'utiliser le hook `{$taxonomy}_add_form` à la place
- `do_action( "{$taxonomy}_add_form", $taxonomy )` pour ajouter des informations au formulaire après le bouton de soumission du formulaire

On a ici tout ce dont on a besoin. Plus précisément, c'est le hook `{$taxonomy}_add_form_fields` qui nous intéresse. Le seul souci est que vu que le formulaire est codé en dur, on ne peut pas modifier l'ordre des champs facilement, et notre champ supplémentaire sera placé après le champ `description`. Allez, on teste.

```
// in main plugin file
add_action( 'category_add_form_fields', 'wpcookbook_category_subtitle_field', 10, 1 );
function wpcookbook_category_subtitle_field( $taxonomy ){
    echo '<p>' . $taxonomy . '</p>';
}
```

On a besoin de notre champ uniquement pour les catégories, donc on remplace la portion dynamique de notre hook `{$taxonomy}_add_form_fields` par le slug de la taxonomie, donc `category`. Aussi, on affiche juste un texte, pour vérifier qu'on est bien hooké au bon endroit.

Ajouter une nouvelle catégorie

**Nom**

Ce nom est utilisé un peu partout sur votre site.

**Slug**

Le slug est la version normalisée du nom. Il ne contient généralement que des lettres minuscules non accentuées, des chiffres et des traits d'union.

**Catégorie parente**

Aucun

Les catégories, contrairement aux étiquettes, peuvent avoir une hiérarchie. Vous pouvez avoir une catégorie nommée Jazz, et à l'intérieur, plusieurs catégories comme Bebop et Big Band. Ceci est totalement facultatif.

**Description**

La description n'est pas très utilisée par défaut, cependant de plus en plus de thèmes l'affichent.

category

Ajouter une nouvelle catégorie

Actions groupées ▾ Appliquer

| Nom           | Description |
|---------------|-------------|
| Uncategorized | —           |

Supprimer une catégorie ne supprime pas les articles de cette catégorie uniquement à cette catégorie se voient assignés à la catégorie par défaut ne peut pas être supprimée.

Les catégories peuvent être converties de manière sélective en étiquettes.

Formulaire de la page des catégories avec notre texte.

Yay ! On est au bon endroit !

```
// in main plugin file
add_action( 'category_add_form_fields', 'wpcookbook_category_subtitle_field', 10, 1 );
/**
 * Adds a text input field for 'Subtitle' on new category page
 *
 * @param string $taxonomy Slug of the taxonomy ('category')
 */
function wpcookbook_category_subtitle_field( $taxonomy ){
    ?>
        <div class="form-field term-subtitle-wrap">
            <label for="category-subtitle"><?php esc_html_e( 'Subtitle', '05-finding-hooks' ); ?></label>
            <input type="text" name="category-subtitle" id="category-subtitle" class="category-subtitle" size=40 />
            <p><?php esc_html_e( 'Type in a subtitle to display in your theme.', '05-finding-hooks' ); ?></p>
        </div>
    <?php
}
```

The screenshot shows the WordPress admin dashboard under the 'WPCookBook' theme. The left sidebar contains links like 'Tableau de bord', 'Articles', 'Ajouter', 'Catégories', 'Étiquettes', 'Médias', 'Pages', 'Commentaires', 'Apparence', 'Extensions', 'Utilisateurs', 'Outils', 'Réglages', and 'Réduire le menu'. The main content area is titled 'Catégories' and has a sub-section 'Ajouter une nouvelle catégorie'. It includes fields for 'Nom' (Name), 'Slug' (Slug), 'Catégorie parente' (Parent Category) set to 'Aucun' (None), and 'Description'. Below these is a 'Subtitle' field, which is highlighted with a yellow border. A button 'Ajouter une nouvelle catégorie' (Add New Category) is at the bottom. To the right, there's a list of categories with 'Uncategorized' selected. A note says 'Supprimer une catégorie ne supprime pas les articles de cette catégorie uniquement à cette catégorie se voient assignés à la catégorie par défaut ne peut pas être supprimée.' Another note says 'Les catégories peuvent être converties de manière sélective en étiquettes.' At the top right, there are 'Actions groupées' and 'Appliquer' buttons.

Formulaire de la page des catégories avec notre champ.

Notre champ est en place. Pour le balisage, j'ai simplement utilisé l'inspecteur ou le code source de `edit-tags.php` pour voir comment les champs texte pour le slug et le nom étaient balisés. Easy.

Par contre, quand on remplit et soumet le formulaire, on a aucune trace de la valeur de notre nouveau champ. Elle n'est ni affichée dans le tableau, ni sur la page d'édition de la catégorie !

Donc concrètement, on a encore du travail pour terminer notre nouvelle fonctionnalité. Il faut chercher les hooks pour sauvegarder la valeur de notre champ (car ce n'est pas automatique), l'afficher dans le tableau, sur la page d'édition d'une catégorie, etc.

Quand on regarde le code HTML de plus près on peut voir que le formulaire soumet à la même page (`action="edit-tags.php"`) et qu'il contient un champ caché `action`. Plus haut dans le fichier, on avait repéré un `switch` sur la valeur du champ `action`, ligne 78.

Pour le cas `add-tag` on peut voir que WordPress utilise `wp_insert_term()` (ligne 91) pour insérer le nouveau terme de la taxonomie en base de données. Mais si on regarde le code de `wp_insert_term()` dans la documentation, on se rend compte qu'il ne traite que peu de champs. On a beau lui passer `$_POST`

tout entier (donc avec notre `category-subtitle`), il l'ignore !

Cela dit, on peut se greffer juste après que le terme soit créé en base de données (en se hookant sur `create_term` ou `create_{taxonomy}`) et sauvegarder notre champ dans ses métadonnées. Pas de panique si cela vous semble compliqué ici. On apprendra ça plus tard. L'objectif ici était d'utiliser les informations présentes dans le navigateur pour trouver les fichiers, fonctions et hooks à utiliser dans le code source.

Pour résumer :

- Dans la barre d'adresse et en inspectant le code HTML, on a trouvé l'identifiant du formulaire et le fichier contenant le code pour l'afficher.
- En lisant le code, on a trouvé le bon hook pour insérer notre champ supplémentaire.
- Avec l'inspecteur, on a pu repliquer le bon balisage pour notre champ
- On a repéré que le `<form>` envoyait ses données vers le même fichier, et passait des informations dans des champs cachés, notamment `action`.
- Dans le code du fichier correspondant, on a vu que `wp_insert_term()` était utilisé pour insérer le terme en base, mais la fonction ne traite pas notre champ directement. Donc il nous faut sauvegarder notre valeur dans des métadonnées. Par contre, `wp_insert_term()` contient un hook qui nous permet de le faire au bon moment.

## Google

Evidemment ! Si vous avez un problème, il y a de fortes chances que quelqu'un d'autre ait eu le même souci ! Et d'autres personnes ont peut-être proposé une solution.

Pensez toutefois à vérifier la pertinence et qualité de la solution proposée sur plusieurs sources et/ou en lisant vous-même les fonctions et hooks utilisés dans la documentation.

Aussi, vérifiez bien le contexte dans lequel la solution est fournie. On ne va pas copier-coller des pavés de 50 lignes sans rien y comprendre et si cela ne correspond pas à votre besoin. Par contre, j'ai découvert beaucoup de hooks de cette façon !

## Ce qu'il faut retenir

- Commencez simplement par analyser l'arborescence et le nom de fichiers. Parfois, il n'y a pas besoin de chercher loin.
- La documentation est une ressource précieuse. N'hésitez pas à utiliser son moteur de recherche.
- Utilisez les fonctionnalités de recherche de votre éditeur de code pour trouver les appels à `add_action` ou `add_filter` ou encore les déclarations de fonction.
- Parfois il faudra remonter l'enchaînement des fonctions utilisées pour trouver les hooks nécessaires.
- Inspectez le code HTML. Vous y trouverez peut-être une classe CSS ou un identifiant utile. Utilisez ensuite cet identifiant pour faire une recherche dans le code.
- Utilisez Google. Soyez prudent et vérifiez que les hooks et fonctions sont pertinents !

# Comment ajouter des hooks dans vos développements

Quand vous créez votre propre thème ou extension, il est essentiel d'ajouter des hooks dans votre code. Cela permet aux autres utilisateurs et développeurs de se greffer sur votre thème ou extension pour en étendre les fonctionnalités.

Vous êtes bien content de pouvoir vous greffer sur WordPress et le modifier à votre guise, non ? C'est ce que permet la plugin API. Donc offrez cela à vos utilisateurs aussi, simplement.

Cela va vous permettre aussi de créer des extensions pour étendre vos extensions. Un peu à la façon Woocommerce : vous créez une extension qui va se greffer sur WordPress, puis créez un addon qui vient étendre cette extension, et un autre utilisateur va en créer un pour personnaliser votre addon.

L'idée est de laisser la main à vos utilisateurs et à votre futur vous.

Il y a plusieurs façons de "laisser la main" à vos utilisateurs. Vous pouvez :

- **Créer une page de réglages** qui va permettre aux utilisateurs de choisir certaines fonctionnalités et certains comportements.
- **Leur laisser la possibilité de surcharger les modèles** : les utilisateurs pourront dupliquer vos templates dans leur thème enfant pour les personnaliser.
- **Les laisser intervenir quand ils le souhaitent** en ajoutant des hooks dans vos développements

On traitera le premier point plus tard dans ce guide quand on apprendra à créer une page de réglages dans l'administration et dans l'outil de personnalisation de WordPress.

Pour l'instant, on va discuter des deuxième et troisième points.

## Surcharger les modèles

On a déjà vu comment créer un thème enfant et personnaliser un modèle de page du parent. Vous vous souvenez ?

Il suffit de dupliquer le modèle de page entier ou le template-part dans votre thème enfant en respectant la structure des dossiers du thème parent, et WordPress et la fonction `get_template_part()` vont automatiquement chercher le fichier dans votre thème enfant d'abord.

Ce principe est utilisable aussi pour les extensions. Sauf que `get_template_part()` ne va chercher que dans le thème. Donc on aura un peu de travail.

Imaginons que nous voulions créer une extension de formulaire de contact (allez-y, créez une petite extension !). Pour le moment, on va utiliser un simple shortcode (code court) pour afficher un formulaire.

```

// In main plugin file
add_shortcode( 'wpcookbook_form', 'wpcookbook_form' );
/**
 * Displays our form
 */
function wpcookbook_form( $atts ){
    ob_start();
    ?>
    <form id="wpcookbook-form" action="" method="POST">
        <p>
            <label for="name" style="display: block;"><?php esc_html_e( 'Your name', '06-
adding-hooks' );?></label>
            <input type="text" id="name" name="name" required />
        </p>
        <p>
            <label for="email" style="display: block;"><?php esc_html_e( 'Your email', '06-
adding-hooks' );?></label>
            <input type="email" id="email" name="email" required />
        </p>
        <p>
            <label for="message" style="display: block;"><?php esc_html_e( 'Your message', '06-
adding-hooks' );?></label>
            <textarea id="message" placeholder=<?php esc_attr_e( 'Type something', '06-
adding-hooks' ); ?>></textarea>
        </p>
        <p>
            <input type="submit" value=<?php esc_attr_e( 'Send message', '06-
hooks' ); ?>>
        </p>
    </form>
    <?php
    return ob_get_clean();
}

```

On déclare un code court avec `add_shortcode()`, en lui passant un nom pour notre code court, puis une fonction de rappel à appeler pour afficher son contenu. On déclare donc une fonction de rappel qui affiche un formulaire tout simple.

Pas de panique, on verra plus en détail plus tard comment les codes courts fonctionnent, et dans le chapitre suivant ce à quoi servent les fonctions `esc_html_e()` et `esc_attr_e()`. Pardonnez aussi mon attribut `style`, s'il vous plaît.

Sur le devant du site, on obtient ceci.

---

## Contact

Your name

Your email

Your message

Type something

**Send message**

[Modifier](#)

### Notre formulaire de contact

On ne va pas s'inquiéter de la façon dont on va traiter les données du formulaire, mais juste de la façon dont on peut charger ce template.

Pour l'instant, le code HTML est en dur dans la fonction de rappel servant à afficher le code court. On peut déjà faire mieux en créant un dossier `templates` dans notre extension, en y créant un fichier `form.php`, en y copiant-collant notre `<form>` et en l'incluant dans notre fonction de rappel.

Notre fonction devient :

```
// in wpcookbook_form() function
/**
 * Displays our form
 */
function wpcookbook_form( $atts ){
    ob_start();
    include 'templates/form.php';
    return ob_get_clean();
}
```

Et notre fichier `templates/form.php`:

```
// in templates/form.php
<?php
/**
 * Our contact form template.
 */
defined( 'ABSPATH' ) || die();
?>

<form id="wpcookbook-form" action="" method="POST">
    ...
</form>
```

Ca marche toujours, car on inclut le template par rapport à notre fichier racine de l'extension. Mais dans une extension avec une architecture plus compliquée, naviguer dans nos dossiers va être plus compliqué.

On verra plus tard comment naviguer dans les dossiers des thèmes et plugins plus en détail. Pour le moment, on va définir deux constantes pour nous aider. Ajoutez ces lignes au début de votre fichier racine de l'extension.

```
// In main plugin file
define( 'ADDING_HOOKS_PATH', plugin_dir_path( __FILE__ ) );
define( 'ADDING_HOOKS_URL', plugin_dir_url( __FILE__ ) );
```

La fonction `plugin_dir_path()` retourne le chemin vers le dossier de notre fichier `__FILE__` passé en paramètre (donc notre fichier bootstrap), et `plugin_dir_url()` en retourne l'URL.

On peut modifier notre `include`:

```
// in wpcookbook_form() function
include ADDING_HOOKS_PATH . '/templates/form.php';
```

La constante va toujours retourner le chemin vers la racine de l'extension, et on peut naviguer à partir de cet endroit.

L'avantage d'un `include` est que les variables accessibles dans la fonction où se trouve le `include` sont aussi accessibles dans le fichier inclus. C'est comme si le code du fichier inclus était simplement copié-collé.

```
// in main plugin file
/**
 * Displays our form
 */
function wpcookbook_form( $atts ){
    $strings = array(
        'dummy_text' => __( 'TOTO', '06-adding-hooks' ),
    );
    ob_start();
    include ADDING_HOOKS_PATH . '/templates/form.php';
    return ob_get_clean();
}
```

```
<?php
/**
 * Our contact form template.
 */
defined( 'ABSPATH' ) || die();
?>

<p><?php echo $strings['dummy_text']; ?></p>

<form id="wpcookbook-form" action="" method="POST">
    ...
</form>
```

## Contact

TOTO



Your name

Your email

Your message

Type something

**Send message**

Modifier

Notre variable est disponible

Notre fichier inclus a bien accès à la variable \$strings, alors qu'elle a été définie dans un autre fichier.  
Pratique.

L'idée maintenant c'est de pouvoir automatiquement inclure les templates en regardant d'abord s'ils existent dans le thème enfant puis dans le parent, à la manière d'un get\_template\_part().

Si on ouvre le code source de get\_template\_part(), on voit qu'elle utilise locate\_template() et c'est cette fonction qui va vérifier si et où les fichiers demandés existent.

On peut donc s'en inspirer (largement) pour créer notre propre fonction.

```
// in main plugin file
/**
 * Returns the path to a template file.
 * Looks first if the file exists in the `wpcookbook/` folder in the child theme,
 * then in the parent's theme `wpcookbook/` folder,
 * finally in the default plugin's template directory
 *
 * @param string $template_name The template we're looking for
 * @return string $located      The path to the template file if found.
 */
function wpcookbook_locate_template( $template_name ) {
    $located = '';

    if ( file_exists( STYLESHEETPATH . '/wpcookbook/' . $template_name ) ) {
        $located = STYLESHEETPATH . '/wpcookbook/' . $template_name;
    } elseif ( file_exists( TEMPLATEPATH . '/wpcookbook/' . $template_name ) ) {
        $located = TEMPLATEPATH . '/wpcookbook/' . $template_name;
    } elseif ( file_exists( ADDING_HOOKS_PATH . '/templates/' . $template_name ) ) {
        $located = ADDING_HOOKS_PATH . '/templates/' . $template_name;
    }

    return str_replace( '...', '', $located );
}
```

La fonction va d'abord chercher dans les thèmes, puis dans notre extension. Pour plus de clarté, on cherche dans le dossier wpcookbook/ des thèmes. Cela évite la confusion si vous dupliquez des templates pour d'autres extensions (Woocommerce, Restrict Content Pro, etc...).

On peut donc modifier notre include :

```
include wpcookbook_locate_template( 'form.php' );
```

Dans notre thème enfant, créez un dossier wpcookbook/, dupliquez-y notre template form.php et ajoutez-y une ligne affichant un message bidon pour tester.

```
// In child theme wpcookbook/form.php
<?php
/**
 * Our child theme's contact form template.
 */
defined( 'ABSPATH' ) || die();
?><h2><?php esc_html_e( 'I am the child theme's template', 'twentynineteen-child' ); ?></h2>
<form id="wpcookbook-form" action="" method="POST">
    ...
</form>
```

Voilà. C'est bien le modèle dans le thème enfant qui est chargé.

## Contact

I am the child theme's template

Your name

Your email

Your message

Type something

**Send message**

Modifier

C'est le modèle du thème enfant qui est chargé

Vous pouvez faire le test en déplaçant le dossier dans le thème parent et en modifiant le message. Ca marche. Vous avez une implémentation très simple d'une fonction qui va s'occuper de charger vos templates en laissant aux utilisateurs la possibilité de les surcharger dans leur thème. C'est pas cool ? J'espère que si !

## Ajouter des hooks

La méthode précédente fonctionne pour les templates mais ne permet pas de modifier les comportements ou données de notre extension. Pour ça, on va devoir ajouter des hooks. C'est très simple, et en fait **vous savez déjà comment faire !**

Si vous ne vous en souvenez plus, rendez-vous au chapitre 1 pour réviser !

### Ajoutons des hooks de templating

Imaginons qu'un utilisateur ait besoin d'ajouter un champ au formulaire. Il peut le faire en dupliquant le template dans son thème enfant via la méthode précédente.

Mais si vous mettez à jour votre extension en ajoutant un champ par défaut dans le template, l'utilisateur **ne bénéficiera pas de votre nouveau champ**, car il aura dupliqué l'ancienne version du formulaire dans son thème enfant et ce sera toujours celle-ci qui sera chargée.

Vous voyez ? Ce n'est plus le fichier de l'extension (à jour) qui est utilisé, mais celui du thème enfant (l'ancien), donc il ne bénéficie pas de vos mises à jour, malheureusement.

Ce qu'on peut faire, c'est ajouter des hooks de type `action` qui vont permettre d'ajouter des champs et autres informations dans le formulaire.

```

<?php
/**
 * Our contact form template.
 */
defined( 'ABSPATH' ) || die();
?>

<?php do_action( 'wpcookbook_before_form', $strings ); ?>

<form id="wpcookbook-form" action="" method="POST">

    <?php do_action( 'wpcookbook_before_fields', $strings ); ?>

    <p>
        <label for="name" style="display: block;"><?php esc_html_e( 'Your name', '06-adding-
hooks' );?></label>
        <input type="text" id="name" name="name" required />
    </p>
    <p>
        <label for="email" style="display: block;"><?php esc_html_e( 'Your email', '06-adding-
hooks' );?></label>
        <input type="email" id="email" name="email" required />
    </p>
    <p>
        <label for="message" style="display: block;"><?php esc_html_e( 'Your message', '06-
adding-hooks' );?></label>
        <textarea id="message" placeholder=<?php esc_attr_e( 'Type something', '06-adding-
hooks' ); ?>></textarea>
    </p>

    <?php do_action( 'wpcookbook_after_fields', $strings ); ?>

    <p>
        <input type="submit" value="<?php esc_attr_e( 'Send message', '06-adding-hooks' ); ?>">
    </p>
</form>

<?php do_action( 'wpcookbook_after_form', $strings ); ?>

```

On a simplement ajouté des appels à `do_action()`, en passant la variable `$strings` à toutes les fonctions de rappel hookées à cet endroit. La variable est disponible et l'utilisateur pourrait en avoir besoin, donc on la lui passe.

Maintenant, un autre utilisateur peut se greffer sur `wpcookbook_after_fields` pour y ajouter son champ :

```
add_action( 'wpcookbook_after_fields', 'wpcookbook_newsletter_field', 10, 1 );
/**
 * Adds a simple checkbox field at the end of the form.
 *
 * @param array $strings Text strings used in the form.
 */
function wpcookbook_newsletter_field( $strings ) {
    ?>
    <p>
        <input id="newsletter" type="checkbox" name="newsletter" />
        <label for="newsletter"><?php echo esc_html( $strings['dummy_text'] );?></label>
    </p>
<?php
}
```

Ici, on se hooke sur `wpcookbook_after_fields` et on affiche un champ. La fonction prend le paramètre `$strings` passé par l'appel à `do_action()` correspondant.

---

## Contact

Your name

Your email

Your message

Type something

TOTO



**Send message**

[Modifier](#)

Notre nouveau champ s'affiche bien.

## Ajoutons un filtre

Donnons maintenant la possibilité de modifier les messages des labels et placeholders des champs. Pour ça, il faut abstraire un peu d'abord. On va extraire les chaînes pour les mettre dans le tableau `$strings` que l'on a défini dans la fonction de rappel (callback function) de notre shortcode.

```
/**  
 * Displays our form  
 */  
function wpcookbook_form( $atts ) {  
    $strings = array(  
        'name' => array(  
            'label'      => __( 'Your name', '06-adding-hooks' ),  
            'placeholder' => __( 'John Doe', '06-adding-hooks' ),  
        ),  
        'email' => array(  
            'label'      => __( 'Your email', '06-adding-hooks' ),  
            'placeholder' => __( 'jdoe@example.com', '06-adding-hooks' ),  
        ),  
        'message' => array(  
            'label'      => __( 'Your message', '06-adding-hooks' ),  
            'placeholder' => __( 'Type something', '06-adding-hooks' ),  
        ),  
    );  
  
    ob_start();  
    include wpcookbook_locate_template( 'form.php' );  
    return ob_get_clean();  
}
```

On va en profiter pour enlever le dummy\_text. Notre template devient :

```

<?php
/**
 * Our contact form template.
 */
defined( 'ABSPATH' ) || die();
?>

<?php do_action( 'wpcookbook_before_form', $strings ); ?>

<form id="wpcookbook-form" action="" method="POST">

    <?php do_action( 'wpcookbook_before_fields', $strings ); ?>

    <p>
        <label for="name" style="display: block;"><?php echo esc_html( $strings['name'] )
['label'] ;?></label>
        <input type="text" id="name" name="name" placeholder="<?php echo esc_attr(
$strings['name']['placeholder'] ); ?>" required />
    </p>
    <p>
        <label for="email" style="display: block;"><?php echo esc_html( $strings['email']
['label'] ); ?></label>
        <input type="email" id="email" name="email" placeholder="<?php echo esc_attr(
$strings['email']['placeholder'] ); ?>" required />
    </p>
    <p>
        <label for="message" style="display: block;"><?php echo esc_html( $strings['message']
['label'] ); ?></label>
        <textarea id="message" placeholder="<?php echo esc_attr( $strings['message']
['placeholder'] ); ?>"></textarea>
    </p>

    <?php do_action( 'wpcookbook_after_fields', $strings ); ?>

    <p>
        <input type="submit" value="<?php esc_attr_e( 'Send message', '06-adding-hooks' );
?>">
    </p>
</form>

<?php do_action( 'wpcookbook_after_form', $strings ); ?>

```

Sur le devant du site, rien n'a changé. Notre formulaire s'affiche normalement. Les chaînes de caractères ont simplement été déplacées dans le tableau \$strings.

Pour permettre aux utilisateurs de personnaliser ces chaînes, on va simplement ajouter un filtre :

```

$strings = apply_filters( 'wpcookbook_form_strings' , array(
    'name' => array(
        'label'      => __( 'Your name', '06-adding-hooks' ),
        'placeholder' => __( 'John Doe', '06-adding-hooks' ),
    ),
    'email' => array(
        'label'      => __( 'Your email', '06-adding-hooks' ),
        'placeholder' => __( 'jdoe@example.com', '06-adding-hooks' ),
    ),
    'message' => array(
        'label'      => __( 'Your message', '06-adding-hooks' ),
        'placeholder' => __( 'Type something', '06-adding-hooks' ),
    ),
) );

```

Le deuxième paramètre est le tableau à filtrer, que l'on passe directement dans `apply_filters()`:

Notre filtre est en place, un utilisateur peut dans sa propre extension se hooker dessus pour modifier le tableau des chaînes de caractères du formulaire.

Rien ne nous empêche d'utiliser notre propre filtre dans notre extension et même de lui ajouter des éléments :

---

```

add_filter( 'wpcookbook_form_strings', 'wpcookbook_form_strings' );
/**
 * Filters text strings used in the form
 *
 * @param array $strings Array of labels and placeholders
 * @return array $strings
 */
function wpcookbook_form_strings( $strings ){
    $strings['message']['label'] = __( 'Type in your message. Don\'t hesitate to include lots
of details.', '06-adding-hooks' );
    $strings['newsletter']['label'] = __( 'Subscribe to the newsletter', '06-adding-hooks' );
    return $strings;
}

```

---

On a ajouté un élément 'newsletter' au tableau, que l'on peut utiliser dans notre nouveau champ :

```

add_action( 'wpcookbook_after_fields', 'wpcookbook_newsletter_field', 10, 1 );
/**
 * Adds a simple checkbox field at the end of the form.
 *
 * @param array $strings Text strings used in the form.
 */
function wpcookbook_newsletter_field( $strings ){
    ?>
    <p>
        <input id="newsletter" type="checkbox" name="newsletter" />
        <label for="newsletter"><?php echo esc_html( $strings['newsletter']['label'] );?>
    </label>
    </p>
    <?php
}

```

## Contact

Your name

John Doe

Your email

jdoe@example.com

Type in your message. Don't hesitate to include lots of details.

Type something

Subscribe to the newsletter

**Send message**

Notre formulaire a des chaines personnalisées.

Et voilà !

### On peut aller encore bien plus loin !

Ici, on ne peut pas modifier l'ordre des champs. Comment faire si un utilisateur a besoin d'ajouter un champ Sujet juste avant le message ? Ce n'est pas possible ici, il n'y a pas de hooks avant le champ message.

On pourrait abstraire les champs dans un tableau déclaratif filtrable du type :

```

$default_fields = apply_filters( 'wpcookbook_default_form_fields', array(
    'name' => array(
        'type'      => 'text',
        'label'     => __( 'Your name', '06-adding-hooks' ),
        'placeholder' => __( 'John Doe', '06-adding-hooks' ),
        ...
    ),
    'email'   => array( ... ),
    'message' => array( ... ),
) );

```

et créer une fonction qui bouclerait sur ces champs pour construire l'HTML, et utiliser cette fonction dans la fonction de rappel de notre code court. Ainsi, on pourrait plus facilement filtrer les champs, en ajouter, en modifier l'ordre, et ce sans avoir besoin des hooks de templating.

Vous voyez l'idée ?

## Ce qu'il faut retenir

- Dans la mesure du possible, **isolez vos templates** dans leurs propres dossiers et fichiers, et laissez à l'utilisateur la possibilité de les dupliquer dans son thème enfant.
- **Ajoutez des hooks** dans vos développements pour laisser à d'autres extensions la possibilité de s'y greffer.
- Les hooks de type **action** sont faciles à ajouter dans des templates ou des processus (traitement des données d'un formulaire, par exemple)
- Les hooks de type **filter** permettent de manipuler des données. C'est donc une bonne idée de **séparer vos données de la logique pour les utiliser** (voir l'exemple juste au-dessus)
- **Mettez des do\_action() et apply\_filters() partout !**

**Ajouter des hooks est essentiel dans vos développements !** Non seulement cela vous force à abstraire votre code et à l'améliorer mais en plus, cela permet à d'autres extensions de venir s'y greffer.

Vendre des addons pour votre extension gratuite, vous trouvez ça intéressant ? Autant vous laisser la porte ouverte pour le faire, non ?

À chaque fois que vous créez une fonction ou déclarez une variable, posez-vous simplement la question : "Est-ce que quelqu'un pourrait avoir besoin d'intervenir à cet endroit/ce moment ?" ou "Est-ce que cela pourrait être utile de rendre cette variable filtrable ?". Souvent la réponse est oui !

**Implémenter ce qui est nécessaire pour répondre à ces questions fera de vous un bien meilleur développeur.** Je vous le garantis.

# Internationaliser ses développements

WordPress est un logiciel qui se veut le plus possible inclusif. Un soin est apporté à l'accessibilité de son interface et les auteurs de thèmes et extensions sont fortement encouragés à prendre en compte au maximum la diversité des utilisateurs de leurs produits.

Certains utilisateurs ont des troubles de la vision à différents degrés. Par exemple, certains peuvent avoir des difficultés à discerner certaines couleurs, d'autres confondent les formes des lettres ou encore ont du mal à suivre un texte si l'espacement des lignes est trop faible. Ces troubles nécessitent des aménagements et une conception de l'interface spécifique.

Mais le trait qui différencie le plus la base immense des utilisateurs de WordPress, c'est simplement la langue. WordPress est traduit dans des dizaines de langues différentes et c'est ce multilinguisme qui contribue aussi à sa popularité.

« Internationaliser » son thème (ou extension) signifie simplement « le rendre traduisible ». C'est maintenant un pré-requis. Pour moi, il est impensable de publier un morceau de code, une extension ou un thème sans le rendre accessible à l'ensemble des utilisateurs, quelle que soit leur langue.

Heureusement, WordPress, tout magique comme peut l'être ce CMS, nous offre tous les outils nécessaires pour rendre nos développements traduisibles.

## `__() et _e()` : les fonctions de base

Dans votre code, promettez-moi de ne jamais écrire ça:

```
$ma_chaine = 'Ma chaine';
```

Mais préférez simplement :

```
$ma_chaine = __( 'Ma chaine' , 'mon-domaine' );
```

Ok, j'avoue, je l'ai fait au début de l'ebook pour un exemple. Ou deux... Bouh ! Mais c'était juste pour vous montrer comment se hooker.

Dans le premier snippet, la chaîne de caractère est codée en dur et non-traduisible. Dans le deuxième, elle l'est. C'est tout simple.

Note : la langue par défaut de WordPress est l'anglais. Donc toutes les chaînes non traduites par défaut dans votre code seront en anglais. Il faudra donc d'abord écrire en anglais, puis traduire en français.

La fonction `__()` prend deux paramètres: une chaîne `$string` et une autre chaîne `$textdomain`, et retourne la traduction de la chaîne de caractères donnée `$string`. S'il n'y a pas de traduction disponible, la chaîne originale est retournée.

La fonction `_e()` prend les deux mêmes paramètres, mais affiche le résultat au lieu de simplement le retourner. Faire `_e( 'Bonjour', 'textdomain' )` ou `echo __( 'Bonjour', 'textdomain' )` est parfaitement équivalent.

Le paramètre `$textdomain` est une clé indiquant dans quel groupe de traductions chercher celle correspondant à la chaîne en question. Simplement parce que la même chaîne peut être utilisée par plusieurs extensions, mais peut se traduire différemment dans chaque extension.

Par exemple, le mot « item » peut se traduire par « produit » (« cart item ») ou « élément » (« menu item »), ou même « item ». Le `text domain` permet de lever cette ambiguïté en catégorisant en quelque sorte vos traductions par extension ou thème.

Dans 99,9% des cas, un seul `text domain` est utilisé par extension ou thème. En général, le `text domain` est exactement la même chaîne que le slug de votre extension ou thème. Si vous voulez utiliser un `text domain` différent, il faut le déclarer dans l'entête de votre extension ou thème. Pour plus de clarté, même si ce n'est pas nécessaire, je le déclare toujours. Pour mon thème Kawi, le `text domain` est `kawi`, simplement.

```
/**  
 * Theme Name: Kawi  
 * Text Domain: kawi  
 * Domain Path: /languages  
 */
```

Le `Domain Path` est le chemin vers les traductions, si vous les incluez dans votre thème ou extension. Comme le `text domain`, `languages/` est la valeur par défaut, que vous pouvez donc omettre.

Note : Vous remarquerez que depuis le début de ce guide, j'utilise un `text domain` correspondant au chapitre, comme '`06-adding-hooks`', alors que j'utilise `wpcookbook_` comme préfixe de fonction. Par convention, je devrais utiliser la même chaîne pour le préfixe et le `text domain`. J'ai utilisé un `text domain` différent par chapitre car chaque chapitre a sa petite extension. Le préfixe unique est utilisé uniquement par `flemme` commodité, et aussi parce qu'un nom de fonction commençant par un chiffre est invalide.

## Traduire des chaînes plus complexes

Dans un bon 50% des cas, les fonctions `__()` et `_e()` sont suffisantes. Maintenant il faut traiter les autres cas, qui sont au final tout aussi courants !

Par exemple, comment faire si vous avez besoin de traduire « Il y a 35 commentaires sur cet article », sachant que 35 est une variable ?

```
$string = __( 'There is ', 'textdomain' ) . $commentsnumber . __( ' comments on this post.',  
'textdomain' );  
echo $string;
```

C'est moche. Et imaginez le traducteur, qui, dans son outil va trouver un « There is » isolé, sans contexte, et un « comments on this article. » dans le même cas (avec un espace devant) ! En plus, si c'est un pluriel cela devrait être « There are », même si j'entends pas mal de « There is » + pluriel de la bouche d'anglophones. Mais bon, c'est une autre histoire.

Non, on ne concatène pas de chaînes comme ça. On utilise `printf()` et `sprintf()`. Ce ne sont pas des fonctions de traduction à proprement parler, mais elles sont très utiles, puisqu'elles permettent d'insérer des placeholders (valeurs fictives) dans une chaîne de caractères pour pouvoir les remplacer par des variables.

Cela donne donc :

```
printf(__( 'There is %d comments on this post.', 'textdomain' ), $commentsnumber );
```

On avance. On a effectivement une seule phrase à traduire et pas deux morceaux sans aucun sens. Dans la chaîne à traduire, `%d` sera remplacé par la valeur numérique de `$commentsnumber`.

Avec `sprintf()` et `printf()` vous pouvez aussi utiliser plusieurs placeholders. Dans ce cas, il est vivement recommandé de les numéroter, car on ne sait jamais : différentes langues utilisent des ordres des mots différents, donc certains placeholders pourraient être inversés. Par exemple:

```
printf(__( 'You have %d subscribers and %d lists', 'textdomain' ), $subscribers, $lists );
```

`printf()` remplace les placeholders dans l'ordre dans lequel il les trouve dans ses paramètres. Donc si par hasard, un traducteur inversait les éléments dans la chaîne (« Vous avez %d listes et %d abonnés ») les données seraient inversées !

```
printf(__( 'You have %1$d subscribers and %2$d lists', 'textdomain' ), $subscribers, $lists );
```

Voilà qui est mieux. On peut maintenant traduire en faisant une référence plus explicite à chaque placeholder. C'est bien beau tout ça, mais on n'a toujours pas résolu le souci du singulier/pluriel.

## `_n()` à la rescousse !

`_n()` est (un peu) plus complexe. Elle prend quatre paramètres : `_n( string $single, string $plural, int $number, string $domain )`, et permet d'aller chercher le singulier ou le pluriel d'une traduction, en fonction de `$number`.

Ce qui veut dire qu'on peut traduire notre phrase deux fois (une forme singulier et une forme pluriel) et aller chercher la bonne en fonction du nombre de commentaires comme ceci:

```
printf(
    _n(
        'There is %d comment on this post.',
        'There are %d comments on this post.',
        $commentsnumber,
        'textdomain'
    ),
    $commentsnumber
);
```

## HTML et traductions

Pour les liens ou tout autre élément HTML, on a un souci. Si votre phrase contient un lien (une balise `<a>`) dont l'ancre va changer en fonction de la langue, alors il vaut mieux laisser la balise dans la chaîne à traduire.

```
printf(
    __( 'For more help, visit <a href="%s">our website</a> and submit a support request.',
    'textdomain' ),
    'https://example.com'
);
```

En utilisant `printf()` et un placeholder pour l'URL, on s'assure que le traducteur ne peut pas modifier l'adresse du lien. Il pourrait cependant enlever la balise `<a>` complète. Mais, on est un peu coincé, là. Donc on n'a pas vraiment le choix et on laisse le lien dans la chaîne à traduire. On peut aussi laisser les balises de formatage comme `<em>`, `<strong>`, ou `<span>`, utilisé pour donner une classe CSS.

Par contre, pour tout ce qui est balise de structure, comme `<div>`, `<section>` ainsi que les titres, on va autant que possible garder les balises hors de la chaîne à traduire.

On va éviter :

```
<?php _e( '<h1>Here is my title !</h1>', 'textdomain' ); ?>
```

Et préférer :

```
<h1><?php _e( 'Here is my title !', 'textdomain' ); ?></h1>
```

Ici pas de risque que le traducteur modifie accidentellement la balise. Il ne s'occupe que du texte.

L'idée générale est qu'il faut éviter au maximum de découper les phrases / unités sémantiques complètes, et la chaîne à traduire doit contenir le moins de code HTML possible. Juste du texte.

## Ajoutons un peu de contexte

`_x()` fonctionne comme `__()`, mais prend un troisième paramètre : `_x( string $text, string $context, string $domain )`. Le paramètre `$context` est simplement une indication pour les traducteurs concernant le contexte dans lequel apparaît la chaîne à traduire pour éviter toute ambiguïté, car même au sein d'une même extension, certaines chaînes isolées peuvent apparaître plusieurs fois et se traduire différemment.

Aussi, il peut être difficile de traduire un mot isolé, genre « Add », ou « Item » ou « Complete ». Un peu de contexte aide beaucoup.

```
$label = _x( 'Item', 'Cart Item', 'textdomain' );
```

## Traductions et sécurité

Une règle de sécurité assez basique est d'échapper les chaînes affichées sur le devant du site. Que veut dire échapper ? Prenons un exemple simple.

Vous avez un formulaire de contact, qui après soumission, affiche un résumé de votre demande. Du type « Merci ! Voici un résumé de votre message : » .

Imaginez qu'un petit malin y écrive « `<script>alert('TOTO');</script>` » dans le champ « nom ». Si le contenu du champ n'est pas échappé lors de l'affichage du résumé sur l'écran suivant, vous aurez une alerte JavaScript !

Ici l'exemple est complètement bidon mais le but est de vous montrer que le code HTML et le JS sont effectivement interprétés et exécutés lors de l'affichage de la valeur brute du champ. Une alerte, c'est bénin, mais ce petit malin a effectivement exécuté du JS sur votre site et aurait pu faire bien pire !

On ne va pas parler de ce qu'il se passe en backend mais le souci est le même. Il faut traiter la donnée et s'assurer qu'elle soit propre et bien ce qui est attendu. Considérez toutes les données provenant d'un utilisateur / navigateur comme malicieuse, ainsi vous saurez quand échapper ou nettoyer.

Sur le devant du site, il faut échapper la chaîne, c'est-à-dire faire en sorte que le texte s'affiche sans que le code HTML de votre site ne soit modifié. Et encore une fois, WordPress a plein d'outil pour nous aider.

La fonction `esc_html()` va échapper une chaîne de caractères en s'assurant que le code HTML ne soit pas interprété. On l'utilise donc pour échapper et afficher les chaînes qui vont se trouver dans du code HTML comme les titres, les paragraphes, etc. En échappant le champ « nom » du formulaire de l'exemple précédent, on va avoir :

Nom: `<script>alert('TOTO');</script>`

Email : ...

Mais le JavaScript ne sera pas exécuté !

Pour les chaînes censées apparaître dans des attributs HTML (comme les `value` des `<input>` par exemple), vous devez utiliser `esc_attr()`. Pour les URL, vous devez utiliser en plus `esc_url()`. Il y en a d'autres, mais ce sont les plus courantes et les plus utiles. On échappe selon un contexte et non selon la valeur.

Par contre, attention quand il y a du code HTML dans la chaîne à traduire, comme :

```
printf(
    __( 'For more help, visit <a href="%s">our website</a> and submit a support request.',
'textdomain' ),
'https://example.com'
);
```

Ici, si on échappe la chaîne, comme ceci:

```
printf(
    esc_html( __( 'For more help, visit <a href="%s">our website</a> and submit a support
request.', 'textdomain' ) ),
'https://example.com'
);
```

On n'aura pas le lien, car la balise `<a>` va être encodée de façon à s'afficher normalement :

For more help, visit <a href="https://example.com">our website</a> and submit a support request.

Donc dans ce cas, on peut laisser passer sans échapper ou utiliser une fonction outil de WordPress comme `wp_kses_post()`. Par contre, si l'URL est une variable, il faut l'échapper :

```
printf(
    __( 'For more help, visit <a href="%s">our website</a> and submit a support request.',
'textdomain' ),
    esc_attr( esc_url( $url ) )
);
```

`wp_kses_post()` est une fonction bien pratique qui va nettoyer notre chaîne en ne laissant que des balises et attributs autorisées par défaut dans l'éditeur de WordPress : `<a>`, `<div>`, `<strong>`, etc.

Maintenant, vous vous demandez sûrement « Pourquoi j'échapperais des chaînes de textes traduisibles ? Pourquoi un traducteur s'amuserait-il à ajouter du code HTML dans ses traductions pour casser les sites sur lesquels sa traduction est installée ? »

C'est vrai. Ils n'ont pas vraiment de raison de faire ça. Mais si je présente ces fonctions, c'est pour que vous soyez conscients qu'elles existent et sont indispensables, mais aussi parce qu'il vaut mieux se méfier des données dans les placeholders. Surtout si ce sont des données entrées par l'utilisateur ! Mieux vaut prévenir que guérir !

## Combos de fonctions

Ce qui est sympa avec ces fonctions de traductions et d'échappement, c'est qu'on a souvent besoin d'en utiliser plusieurs imbriquées les unes dans les autres. Et pour ce faire, WordPress met à notre disposition des combos de fonctions, deux en un, ou plus. Par exemple, on a déjà vu `_e()`, qui est un combo echo + `__()`.

Mais on a aussi :

```
_ex() = _e() + _x()  
_nx() = _n() + _x()  
esc_html__( ) = esc_html() + __()  
esc_html_e() = esc_html() + _e()  
esc_html_x() = esc_html() + _x()  
esc_attr__( ) = esc_attr() + __()  
esc_attr_e() = esc_attr() + _e()  
esc_attr_x() = esc_attr() + _x()
```

Cool, hein ?

Vous voyez ? Internationaliser son thème ou son extension n'est pas si compliqué ! WordPress met plein d'outils à notre disposition pour rendre nos développements traduisibles, et nous devons le faire.

## Ce qu'il faut retenir

- JAMAIS de chaînes de caractères hardcodées. On utilise TOUJOURS une fonction de traduction comme `__()` ou `_e()` au minimum.
- Autant que possible, on ne coupe pas d'unités sémantiques (phrases ou paragraphes). On utilise `printf()` et `sprintf()` avec des placeholders. Attention aux formes singulier/pluriel.
- Aussi, on évite au maximum le code HTML dans les chaînes à traduire. Le seul cas où on a pas vraiment le choix, c'est pour les liens.
- On échappe toute chaîne/donnée pouvant être considérée comme non-sécurisée quand on les affiche, en utilisant les fonctions `esc_html()`, `esc_attr()` et ses copines.
- On écrit d'abord en anglais, car c'est la langue par défaut.
- On déclare toujours un `text-domain`. On en parlera plus en détails dans le chapitre suivant.

Ce qui est sécurisé ou pas pourrait faire l'objet d'un ebook entier. Une donnée venant de la base de données pourrait potentiellement ne pas être sécurisée. La règle est simple : ne jamais faire confiance à vos utilisateurs et toujours être prudent avec leurs entrées !

# Traduire vos extensions et thèmes

Rendre son extension ou thème traduisible est la première étape. Maintenant, on va voir comment fournir et charger les fichiers de traductions. Ceux-ci sont de trois types : les .pot, .po, et les .mo.

Le fichier .pot (Portable Object Template) est le modèle des traductions. Il contient les identifiants des chaînes à traduire, mais aucune traduction.

Les fichiers .po (Portable Object) vont contenir vos traductions. Chaque fichier va contenir un identifiant par chaîne et sa traduction dans la langue concernée. Il faut donc un fichier .po par locale.

Les fichiers .mo (Machine Object) sont les fichiers qui sont lus par WordPress pour charger vos traductions. Un .mo est un .po compilé et compressé en quelque sorte.

Tout d'abord, voyons comment WordPress charge les traductions des thèmes et extensions.

Pour ça, rien de bien compliqué. Il suffit de connaître quelques conventions, deux fonctions et de les hooker au bon endroit.

## Deux options possibles

Pour vos traductions vous avez deux possibilités :

- Soit votre extension ou thème est disponible sur le répertoire officiel de WordPress, auquel cas vous avez la possibilité de le traduire via <https://translate.wordpress.org>. Vos fichiers de traductions sont générés automatiquement et sont téléchargés quand WordPress vérifie les mises à jour disponibles. Les fichiers sont alors téléchargés dans le dossier wp-content/languages/themes ou wp-content/languages/plugins.
- Soit vous fournissez vos propres fichiers de traduction dans votre thème ou extension. Si vous distribuez votre extension vous-même, vous n'avez pas trop le choix. Les fichiers de traductions sont à placer dans le dossier languages/ de votre thème ou extension.

## Text Domain et Domain Path

Par convention, le text domain doit être identique au slug de votre thème ou extension (qui est le nom de son dossier).

Donc si votre thème s'appelle "Mon Super Thème", alors son dossier et slug seront mon-super-theme et son text domain aussi.

Pour rappel, le domain path est le chemin vers les traductions, si elles sont fournies avec votre thème ou extension. Il vaut languages/ par défaut, et il n'y a pas vraiment de raison de le changer. Vous pouvez l'inclure dans l'entête de votre thème ou extension ou non.

Par exemple, dans le fichier `style.css` du thème:

```
Theme Name : Mon Super Thème  
Text Domain: mon-super-theme  
Domain Path: /languages  
...
```

Attention ! Si votre `text_domain` est différent de votre slug, alors vos traductions sur <https://translate.wordpress.org> ne fonctionneront pas. Il faut simplement respecter la convention. C'est facile.

## Charger les traductions

### Dans un thème

Pour charger les traductions **dans un thème**, il suffit d'utiliser la fonction `load_theme_textdomain` sur le hook `after_setup_theme`.

```
// in functions.php  
add_action( 'after_setup_theme', 'mon_super_theme_load_translations' );  
/**  
 * Load translations for the theme  
 */  
function mon_super_theme_load_translations(){  
    load_theme_textdomain( 'mon-super-theme', get_template_directory() . '/languages' );  
}
```

Imaginons que le site est en français FR. La locale est donc `fr_FR`.

La fonction `load_theme_textdomain()` va d'abord chercher dans le dossier des traductions (`wp-content/languages/themes/`) si le fichier `mon-super-theme-fr_FR.mo` existe. Sinon, elle va chercher le fichier `fr_FR.mo` (sans le slug) dans le dossier donné en second paramètre. Si vous omettez le deuxième paramètre, elle va chercher à la racine du thème.

### Dans une extension

Dans le fichier `bootstrap` de votre extension, on va juste inclure :

```
add_action( 'init', 'mon_extension_load_textdomain' );  
/**  
 * Load translations  
 */  
function mon_extension_load_textdomain() {  
    load_plugin_textdomain( 'mon-extension', FALSE, basename( dirname( __FILE__ ) ) .  
    '/languages/' );  
}
```

Comme `load_theme_textdomain()`, `load_plugin_textdomain()` va d'abord chercher dans le dossier `wp-content/languages/plugins/` si un fichier `mon-extension-{locale}.mo` existe, sinon elle va chercher dans le dossier passé en troisième paramètre.

Par contre, le fichier que vous fournissez doit s'appeler `mon-extension-{locale}.mo` et pas simplement `{locale}.mo`.

## Ce qu'il faut retenir

- Le `text_domain` et le `slug` de votre thème ou extension doivent être identiques.
- On hooke la fonction `load_plugin_textdomain()` sur `init` pour charger les traductions d'une extension.
- On hooke la fonction `load_theme_textdomain()` sur `after_setup_theme` pour charger les traductions d'un thème.
- Si vous avez traduit via <https://translate.wordpress.org>, vous n'avez rien d'autre à faire.
- Sinon, vous devez embarquer vos fichiers de traduction dans le dossier `languages/` de votre thème ou extension. Le fichier `.mo` doit s'appeler `{locale}.mo` dans le cas d'un thème, et `{slug}-{locale}.mo` pour une extension.

## Comment générer un fichier de traductions

Si vous devez fournir vos traductions avec votre thème ou extension, il faut générer un fichier `.pot` et si besoin les `.po` et `.mo` pour les langues que vous voulez supporter.

Pour ce faire, on va créer une petite extension bidon pour tester, et on va fournir une traduction.

Créez une extension avec ce code dans le fichier `bootstrap` :

```

<?php
/**
 * Plugin Name: 08 - Localizing
 * Plugin URI: https://example.com/plugins/08-localizing/
 * Description: Examples of how to load translations.
 * Version: 1.0
 * Requires at least: 5.2
 * Requires PHP: 7.2
 * Author: Vincent Dubroeucq
 * Author URI: https://vincentdubroeucq.com/
 * License: GPL v2 or later
 * License URI: https://www.gnu.org/licenses/gpl-2.0.html
 * Text Domain: 08-localizing
 * Domain Path: languages/
 */
defined( 'ABSPATH' ) || die();

add_action( 'wp_footer', 'wpcookbook_footer_message' );
/**
 * Displays a simple message in the footer of the site.
 */
function wpcookbook_footer_message(){
    ?>
    <p><?php esc_html_e( 'Made with love with WordPress', '08-localizing' ); ?></p>
    <?php
}

```

On se hooke sur wp\_footer et on affiche un message, tout simplement. Pas trop compliqué.



## Catégories

[Uncategorized](#)

## Méta

[Admin. du site](#)

[Déconnexion](#)

[Flux RSS des articles](#)

[RSS des commentaires](#)

[Site de WordPress-FR](#)

WPCookBook, Fièrement propulsé par WordPress.

Made with love with WordPress



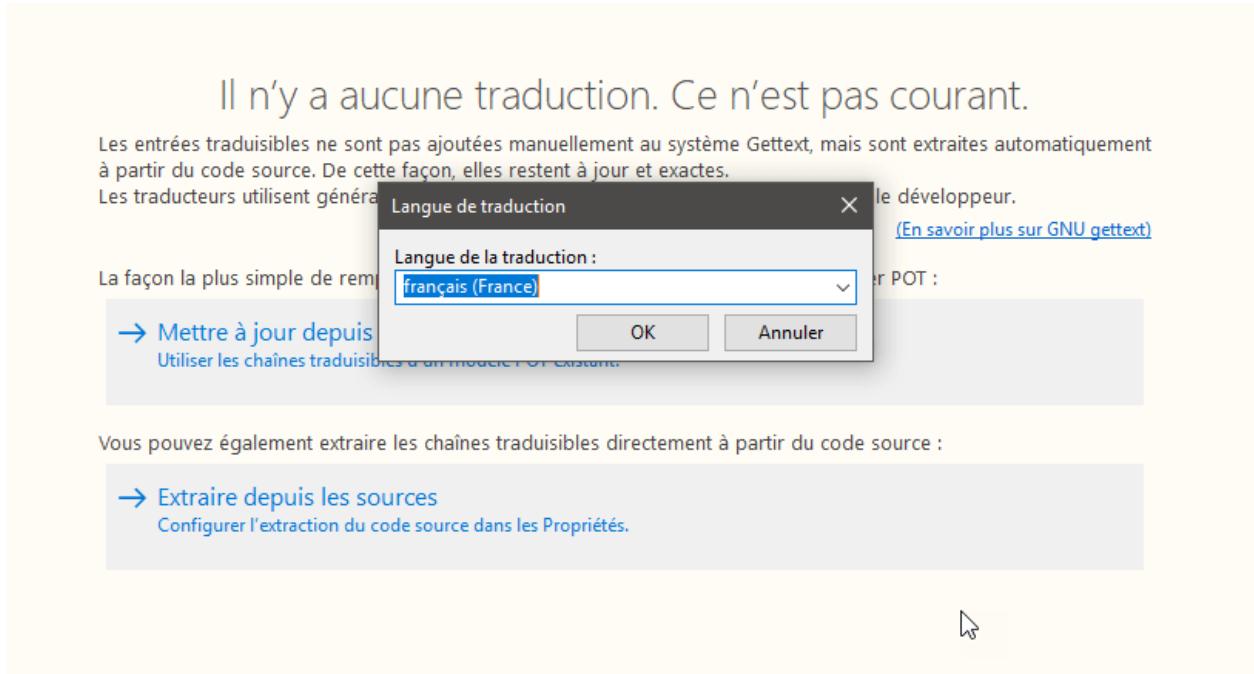
On affiche un message tout simple

Maintenant, on va utiliser le logiciel PoEdit pour créer un modèle de traduction et une traduction française. Téléchargez le logiciel [PoEdit](#), et installez-le.

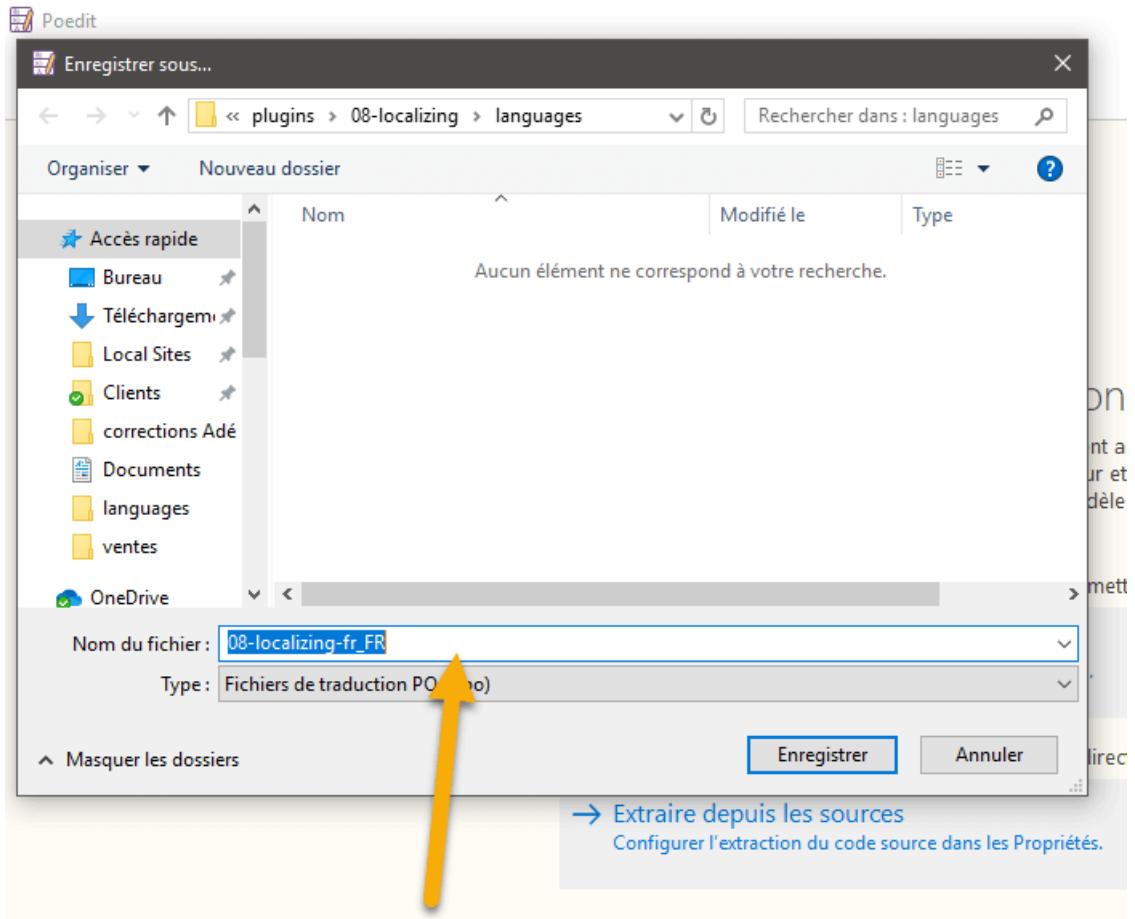
Note : Il y a d'autres façons de générer les fichiers de traductions, comme des outils CLI, par exemple. Mais le plus simple est d'avoir une joli GUI, non ? Ok, elle n'est pas spécialement jolie...

Créez un dossier `languages/` dans votre extension.

Ouvrez PoEdit, puis cliquez sur Fichiers > Nouveau, sélectionnez français pour votre nouvelle traduction.



Sauvegardez tout de suite votre projet sous le format `{slug}-fr_FR.po`. Puis allez dans Catalogue > Propriétés pour préparer votre catalogue de chaînes.



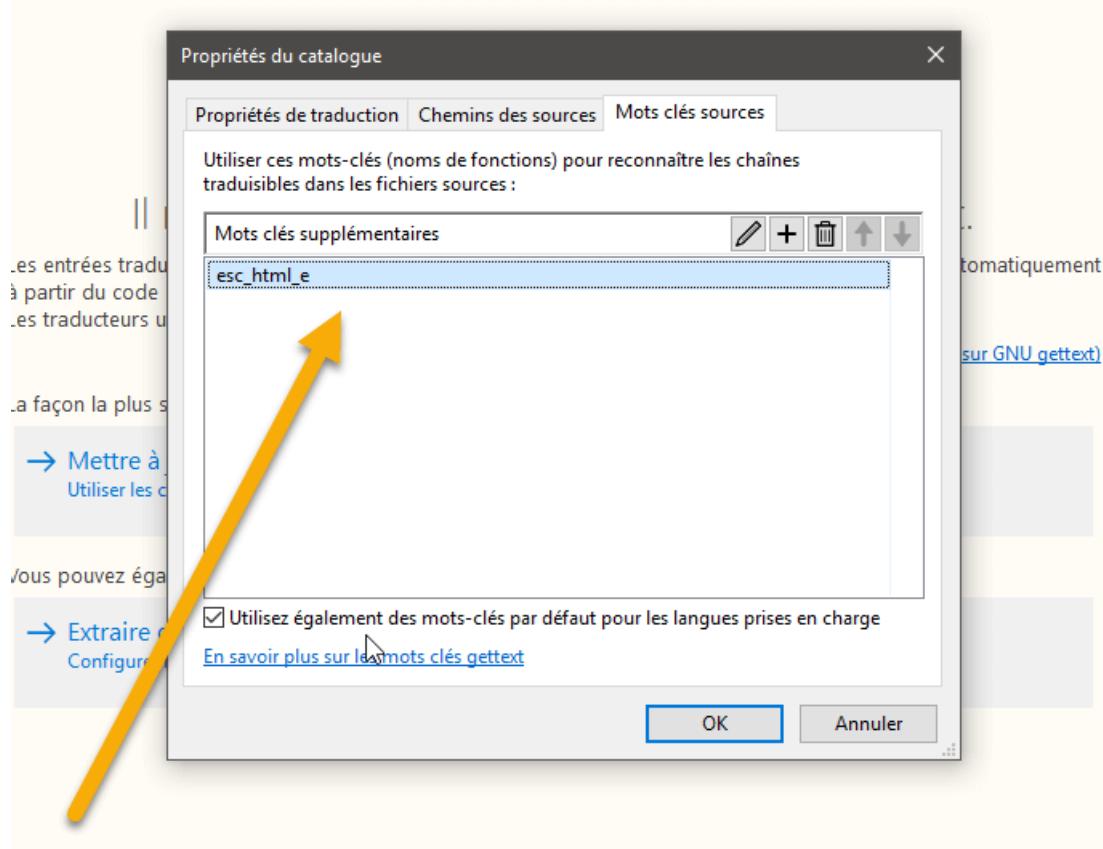
#### Sauvegarde du projet

Entrez un nom de projet et auteur dans le premier onglet (Propriétés de traduction), puis dans l'onglet Chemin des sources, ajoutez le dossier racine de votre extension.

Puis dans le troisième onglet (Mots clés sources), il va falloir ajouter tous les mots clés gettext utilisés dans votre code pour afficher vos chaînes.

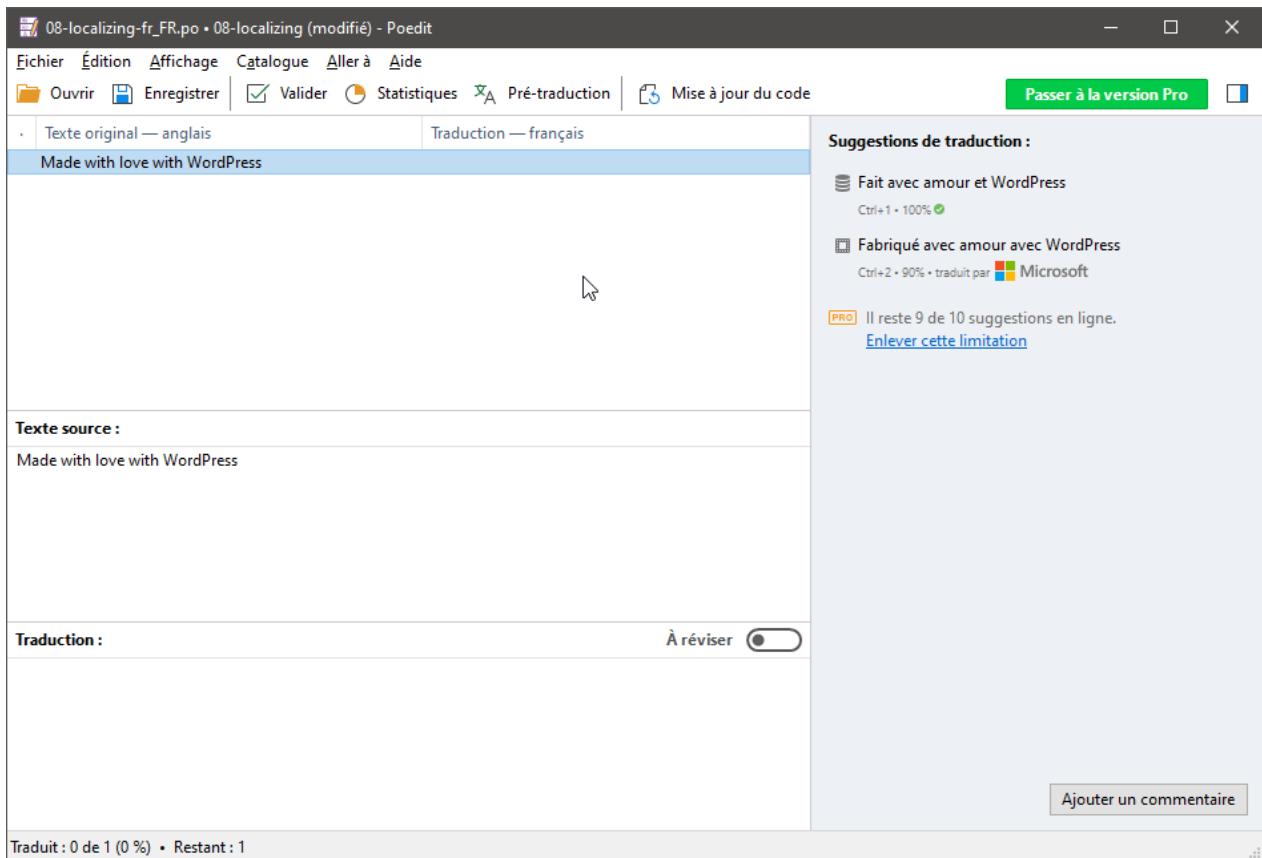
C'est-à-dire, on va donner à PoEdit les fonctions utilisées pour afficher nos chaînes, puis il va scanner nos fichiers et trouver les chaînes correspondantes.

Donc pour le moment, on va juste lui donner esc\_html\_e.



Mots clés Gettext

Cliquez sur "Ok", sauvegardez votre projet, et cliquer sur "Mise à jour du code". Tadaa !



Nos chaînes à traduire sont détectées.

Traduisez simplement la chaîne comme vous le souhaitez, et sauvegardez. Cela génère automatiquement les fichiers .po et .mo dont on a besoin. Vous pouvez aller jeter un œil à votre dossier languages/.

Maintenant, il faut revenir dans le code de notre extension et charger la traduction. Ajoutez simplement ces quelques lignes juste avant notre fonction hookée sur wp\_footer.

```
// in main plugin file
add_action( 'init', 'wpcookbook_load_textdomain' );
/**
 * Load translations
 */
function wpcookbook_load_textdomain() {
    load_plugin_textdomain( '08-localizing', FALSE, basename( dirname( __FILE__ ) ) .
    '/languages/' );
}
```

Sur le devant du site, on a bien nos traductions chargées.

# Catégories

## Uncategorized

WPCookBook, Fièrement propulsé par WordPress.

Fait avec amour et WordPress



Nos chaines sont traduites.

Notre traduction fonctionne, mais nous n'avons utilisé qu'une seule fonction de traduction : `esc_html_e()`. Il manque donc encore pas mal de configuration pour que notre catalogue soit bulletproof.

Honnêtement, entrer tous les mots clés gettext avec leurs paramètres dans PoEdit est très pénible. C'est le genre de chose qu'on n'a envie de faire qu'une seule fois, et de copier-coller ensuite.

Heureusement, notre ami FX Bénard l'a fait pour nous ! Merci ! Vous pouvez télécharger un fichier `.pot` vierge sur <https://github.com/fxbenard/Blank-WordPress-Pot>

Ce fichier ne contient aucune traduction, évidemment, mais tous les mots-clés sources sont configurés.

Il ne vous reste qu'à placer ce fichier `.pot` dans votre dossier `languages/`, de le nommer correctement, de changer les données de base (nom du projet, auteur, etc...) et de vous en servir comme modèle pour vos traductions.

Merci FX !

## Ce qu'il faut retenir

- Un .pot vide est disponible sur <https://github.com/fxbenard/Blank-WordPress-Pot>. On ne va pas s'en priver.
- Renommez ce .pot selon le slug de votre thème ou extension, et placez-le dans votre dossier languages/.
- Ouvrez-le dans PoEdit et mettez à jour les propriétés du catalogue.
- Sauvegardez le tout, et mettez à jour les chaînes en les extrayant à partir du code.
- Créez une nouvelle traduction, et nommez-la correctement :{slug}-{locale}.po pour les extensions et {locale}.po pour les thèmes.
- Utilisez `load_plugin_textdomain()` ou `load_theme_textdomain()` pour charger vos fichiers.

Voilà, votre thème ou extension est translation-ready !

# Comment naviguer dans vos thèmes et extensions

Pendant vos développements, vous aurez forcément besoin de référencer des fichiers, que ce soit dans votre extension ou thème ou alors dans l'installation de WordPress complète.

Encore une fois, WordPress met à notre disposition des fonctions utiles qui vont couvrir tous nos besoins. C'est magique non ?

Ce qu'il est important de comprendre, c'est qu'il ne faut jamais coder en dur un chemin ou une URL vers un fichier !

WordPress permet de TOUT personnaliser, y compris les dossiers dans lesquels on va ranger notre contenu, nos uploads, etc.

Donc si vous voulez que votre développement soit utilisable sur un maximum de sites, même si les utilisateurs ont personnalisé leurs dossiers avec des noms imprévisibles, il faut absolument utiliser toutes ces fonctions et outils que je vais détailler qui vont effectuer toutes ces vérifications de chemin pour vous.

Je m'excuse en avance, ce chapitre est un peu un catalogue des fonctions utiles pour obtenir les bons chemins et URL dans vos thèmes et extensions. Donc un chapitre un peu "liste" avec des exemples, mais pas forcément concrets. Pas de panique, c'est le dernier avant d'attaquer les choses vraiment plus concrètes. Courage !

## Pour les extensions

### `plugins_url()`

`plugins_url( string $path, string $plugin )` est une fonction ultra utile.

Elle retourne une URL du chemin de l'extension passée en second paramètre, et lui ajoute le chemin passé en premier paramètre.

Si les paramètres sont omis, elle retourne l'url vers le dossier des extensions, simplement.

Ce qui veut dire que dans votre fichier bootstrap de l'extension, vous pouvez utiliser la constante `__FILE__`, qui réfère au chemin du fichier actuel, pour récupérer l'URL de votre extension :

```
echo plugins_url( '', __FILE__ );
// Dans mon cas, cela affiche https://wpcookbook.local/wp-content/plugins/09-moving-around
```

Vous pouvez ensuite lui ajouter le chemin souhaité en premier paramètre.

```
echo plugins_url( 'assets/js', __FILE__ );
// Dans mon cas, cela affiche https://wpcookbook.local/wp-content/plugins/09-moving-around/assets/js
```

## plugin\_dir\_url et plugin\_dir\_path()

plugin\_dir\_url( string \$file ) renvoie l'URL vers le dossier dans lequel est situé le fichier passé en paramètre (obligatoire).

Comme plugins\_url(), on lui passe \_\_FILE\_\_ pour avoir l'URL vers le fichier courant.

plugin\_dir\_path( string \$file ) renvoie le chemin vers le dossier contenant le fichier, simplement.

```
echo plugin_dir_url( __FILE__ );
// Dans mon cas, cela affiche https://wpcookbook.local/wp-content/plugins/09-moving-around
echo plugin_dir_path( __FILE__ );
// Dans mon cas, cela affiche /app/public/wp-content/plugins/09-moving-around/
```

## plugin\_basename()

plugin\_basename( string \$file ) retourne le nom de votre extension.

```
echo plugin_basename( __FILE__ );
// Dans mon cas, cela affiche 09-moving-around/09-moving-around.php
```

Dans une extension, vous pouvez vous en sortir très facilement avec uniquement ces fonctions.

Au fur et à mesure que votre extension deviendra complexe, avec pleins de dossiers et fichiers, cela peut devenir compliqué d'utiliser ces fonctions. Donc, au lieu de vous casser la tête avec les paramètres de plugins\_url(), définissez simplement quelques constantes dans votre fichier bootstrap qui vont vous aider tout au long de votre développement.

```
define( 'WPCOOKBOOK_PLUGIN_URL', plugin_dir_url( __FILE__ ) );
define( 'WPCOOKBOOK_PLUGIN_PATH', plugin_dir_path( __FILE__ ) );
```

Ensuite, utilisez ces constantes pour naviguer dans vos dossiers. Par exemple, si vous avez besoin de charger un fichier CSS,

```
wp_enqueue_style( 'wpcookbook-styles', WPCOOKBOOK_PLUGIN_URL . 'css/style.css' );
```

C'est beaucoup plus simple que de recalculer l'URL vers le fichier CSS à partir du fichier dans lequel se situe l'appel à wp\_enqueue\_style() avec plugins\_url().

Vous pouvez aussi aller plus loin en définissant des constantes plus précises vers votre dossier templates/ ou assets/ par exemple.

```
define( 'WPCOOKBOOK_TEMPLATE_URL', plugins_url( 'templates/' , __FILE__ ) );
define( 'WPCOOKBOOK_TEMPLATE_PATH', plugin_dir_path( __FILE__ ) . 'templates/' );
```

## Pour les thèmes

### get\_template\_directory() et get\_template\_directory\_uri()

Ces deux fonctions ne prennent aucun paramètre et retournent le chemin (get\_template\_directory()) ou l'URL (get\_template\_directory\_uri()) vers le thème parent utilisé.

```
// Mon thème actif est twentynineteen-child
echo get_template_directory();
// Affiche /app/public/wp-content/themes/twentynineteen
echo get_template_directory_uri();
// Affiche https://wpcookbook.local/wp-content/themes/twentynineteen
```

Attention, c'est très important ! On parle du thème parent ! Ces fonctions ne peuvent pas être utilisées pour naviguer dans le thème enfant !

### get\_stylesheet\_directory() et get\_stylesheet\_directory\_uri()

Ces fonctions fonctionnent (haha...) exactement comme les deux précédentes, sauf qu'elles renvoient le chemin ou l'URL du thème **actif**. Donc si vous les utilisez dans un thème enfant, elles vous donneront le chemin et l'URL du thème **enfant**, mais pas du parent. Si vous n'utilisez pas de thème enfant, alors utiliser ces fonctions ou celles décrites juste avant est équivalent.

```
echo get_stylesheet_directory();
// Affiche /app/public/wp-content/themes/twentynineteen-child
echo get_stylesheet_directory_uri();
// Affiche https://wpcookbook.local/wp-content/themes/twentynineteen-child
```

### get\_theme\_file\_uri() et get\_theme\_file\_path()

Ces fonctions prennent un nom de fichier en paramètre et vont retourner le chemin ou l'URL vers ce fichier en cherchant d'abord s'il existe dans le thème enfant, sinon dans le parent.

```
echo echo get_theme_file_uri( '404.php' );
// Affiche https://wpcookbook.local/wp-content/themes/twentynineteen/404.php (donc l'url dans
le thème parent)
echo echo get_theme_file_uri( 'style.css' );
// Affiche https://wpcookbook.local/wp-content/themes/twentynineteen-child/style.css (thème
enfant)
```

Dans l'exemple, un thème enfant de twentynineteen est installé et le fichier 404.php n'est pas

surcharge dans ce dernier. Donc c'est l'URL du fichier dans le thème parent qui est retournée. Par contre, le fichier `style.css` existe bien dans le thème enfant.

Pour obtenir l'URL de la feuille de style directement, on peut utiliser `get_theme_file_uri()` comme je l'ai fait dans l'exemple, ou alors on peut aussi utiliser :

```
echo get_stylesheet_uri();  
// Affiche https://wpcookbook.local/wp-content/themes/twentynineteen-child/style.css
```

Bim. C'est encore plus simple. On obtient directement l'URL du fichier `style.css` du thème actif (enfant ou parent).

Ces fonctions sont donc extrêmement utiles, car si elles sont utilisées dans le thème parent, elles permettent au thème enfant de surcharger les fichiers appelés de cette manière.

Souvenez-vous quand on a créé un thème enfant de `twentynineteen` dans le chapitre 2, nos styles du thème parent n'étaient pas chargés. Simplement parce que le thème parent utilisait `get_stylesheet_uri()` pour récupérer l'URL de la feuille de style du thème actif (donc l'enfant). On a donc dû aller charger explicitement les styles du parent avec `get_template_directory_uri()`. L'inverse (styles du parent chargés) est aussi possible, selon le thème.

## `get_parent_theme_file_uri()` et `get_parent_theme_file_path()`

Je pense que les noms des fonctions sont assez explicites. Elles prennent un nom de fichier en paramètre, et vont le chercher dans le parent uniquement.

```
echo get_parent_theme_file_uri( 'style.css' );  
// Affiche https://wpcookbook.local/wp-content/themes/twentynineteen/style.css (thème parent)  
echo get_parent_theme_file_path( 'style.css' );  
// Affiche /app/public/wp-content/themes/twentynineteen/style.css
```

## `get_theme_root()` et `get_theme_root_uri()`

Personnellement, je n'en ai encore jamais eu besoin, mais si vous avez besoin d'accéder au dossier des thèmes, vous pouvez utiliser `get_theme_root()` et `get_theme_root_uri()`.

```
echo get_theme_root();  
// Affiche /app/public/wp-content/themes  
echo get_theme_root_uri();  
// Affiche https://wpcookbook.local/wp-content/themes
```

## Pour le reste

### home\_url() et site\_url()

Attention à ne pas confondre ces deux fonctions. La plupart du temps, elles renvoient la même chose, mais il est important de comprendre pourquoi.

home\_url() renvoie l'URL de la page d'accueil du site, alors que site\_url() renvoie l'URL à laquelle les fichiers de WordPress sont disponibles.

Typiquement, chez votre hébergeur vous allez placer vos fichiers dans un dossier `public_html` et votre domaine va pointer à cet endroit.

Mais si vous avez installé votre site dans un sous-dossier `wordpress` (par exemple) du dossier `public_html` et pas à la racine, alors la fonction site\_url() va vous renvoyer `https://example.com/wordpress`.

```
// Cas 1 : Mes fichiers de WordPress sont à la racine du dossier vers lequel pointe mon nom de domaine.  
echo home_url();  
// Affiche https://wpcookbook.local  
echo site_url();  
// Affiche aussi https://wpcookbook.local, car mes fichiers sont à la racine du dossier où pointe mon domaine.
```

```
// Cas 2 : Mes fichiers de WordPress sont dans un sous-dossier wordpress/ du dossier vers lequel pointe mon nom de domaine.  
echo home_url();  
// Affiche https://wpcookbook.local  
echo site_url();  
// Affiche https://wpcookbook.local/wordpress
```

Donc si vous avez besoin d'accéder à un fichier de WordPress, on va plutôt utiliser site\_url(). Si vous avez besoin d'une URL censée être disponible sur le devant du site, on va utiliser home\_url().

Aussi, ces deux fonctions acceptent toutes les deux les mêmes paramètres : `home_url( string $path = '', string|null $scheme = null )`. \$path est un chemin qui va être ajouté à l'URL du devant du site, et \$scheme permet de forcer le protocole http ou https.

```
echo home_url( 'example-page' );  
// Affiche https://wpcookbook.local/example-page
```

## admin\_url()

Cette fonction renvoie une URL vers le dossier de l'administration de WordPress. Comme `home_url()`, vous pouvez lui passer un chemin en premier paramètre qui va être concaténé à l'URL renvoyée, et forcer le protocole avec le second paramètre.

```
echo admin_url();
// Affiche https://wpcookbook.local/wp-admin
echo admin_url( 'admin-ajax.php' );
// Affiche https://wpcookbook.local/wp-admin/admin-ajax.php
```

## includes\_url() et content\_url()

Ces fonctions renvoient les URL vers les dossiers `wp-includes/` et `wp-content/` de WordPress. Comme `home_url()`, vous pouvez passer un chemin pour le concaténer. Par contre, avec `content_url()`, on ne peut pas forcer le protocole.

```
echo includes_url();
// Affiche https://wpcookbook.local/wp-includes
echo content_url();
// Affiche https://wpcookbook.local/wp-content
```

## wp\_upload\_dir()

Celle-ci est plus complexe, car elle renvoie un tableau de données :

```
var_dump( wp_upload_dir() );
/** Afficher
 *array(6) {
 *  ["path"]=>
 *  string(38) "/app/public/wp-content/uploads/2019/10"
 *  ["url"]=>
 *  string(51) "https://wpcookbook.local/wp-content/uploads/2019/10"
 *  ["subdir"]=>
 *  string(8) "/2019/10"
 *  ["basedir"]=>
 *  string(30) "/app/public/wp-content/uploads"
 *  ["baseurl"]=>
 *  string(43) "https://wpcookbook.local/wp-content/uploads"
 *  ["error"]=>
 *  bool(false)
 *} */
*/
```

La fonction prend trois paramètres : `wp_upload_dir( string $time = null, bool $create_dir = true, bool $refresh_cache = false )`. Elle renvoie toutes les informations dont nous avons besoin dans le tableau, et par défaut, elle crée un dossier pour y placer les téléchargements automatiquement si le dossier pour le mois en cours n'existe pas encore et que vous avez choisi de ranger vos fichiers de cette

façon dans vos réglages des médias.

On peut lui passer un format de date si l'on veut créer un autre dossier ou changer la structure actuelle, et forcer ou non la création automatique du dossier ou purger le cache. Dans 99% des cas, si vous avez besoin de cette fonction, vous l'utiliserez telle quelle, sans paramètres.

L'avantage c'est que l'on a accès aux chemins et URL vers le dossier en cours, ou le dossier `uploads/` racine. C'est très pratique.

## Les constantes magiques

Pour calculer tous ces chemins et URL, WordPress se base sur des constantes qui sont définies dans le fichier `wp_config.php` ou à différents moments de son chargement.

- `ABSPATH` : contient le chemin vers le dossier racine de WordPress
- `WPINC` : contient le chemin vers le dossier `wp-includes/`
- `WP_CONTENT_DIR` : contient le chemin vers le dossier `wp-content/`
- `WP_CONTENT_URL` : contient l'URL vers le dossier `wp-includes/`
- `WP_PLUGIN_DIR` : contient le chemin vers le dossier des extensions
- `WP_PLUGIN_URL` : contient l'URL vers le dossier des extensions
- `TEMPLATEPATH` : contient le chemin vers le dossier du thème parent
- `STYLESHEETPATH` : contient le chemin vers le dossier du thème enfant

Il existe bien d'autres constantes, que ce soit pour des chemins, URL ou autres données mais le but du guide n'est pas de les lister.

Par contre, ce qu'il faut bien comprendre, c'est que ces constantes sont utilisées dans les fonctions précédemment décrites, et sont surchargeables dans votre fichier `wp-config.php`. C'est pourquoi il faut absolument utiliser les fonctions outils précédemment décrites pour calculer les chemins et URL dans vos développements, pour éviter les bugs si un utilisateur a personnalisé ses dossiers via ces constantes.

## Utilisez les paramètres des fonctions

Certaines fonctions présentées ici, comme `admin_url()` prennent un ou plusieurs paramètres pour concaténer un chemin à l'URL. Il est important d'utiliser le paramètre `$path` de la fonction au lieu de concaténer "en dur" vos chemins.

C'est-à-dire qu'au lieu de :

```
echo $url = admin_url() . 'admin-ajax.php';
```

Préférez :

```
$url = admin_url( 'admin-ajax.php' );
```

Mais pourquoi ?

Le résultat dans votre code isolé est le même, mais en coulisses, `admin_url()` appelle `get_admin_url()` et cette dernière expose le filtre `admin_url`.

Ce qui veut dire qu'on peut filtrer le résultat de la fonction, en se hookant sur `admin_url`. Or, les fonctions de rappel hookées ici prennent l'URL en premier paramètre, mais reçoivent aussi le chemin `$path` passé en paramètre (ainsi que l'identifiant du site sur un multisite).

Si une extension a hooké un filtre sur `admin_url`, et que l'extension compte vérifier le `$path` passé en paramètre, vous risquez de créer des bugs si vous concaténez au lieu d'utiliser les paramètres de `admin_url()`.

Voici un exemple simple pour que vous compreniez bien :

```
add_filter( 'admin_url', 'wpcookbook_admin_url', 10, 3 );
/**
 * Adds query arg only to urls pointing to admin-ajax.php
 */
function wpcookbook_admin_url( $url, $path, $blog_id ){
    if( 'admin-ajax.php' === $path ){
        $url = add_query_arg( 'test', 'true', $url );
    }
    return $url;
}

// Dans une fonction affichant du contenu :
echo $url = admin_url() . 'admin-ajax.php'; // https://wpcookbook.local/wp-admin/admin-
ajax.php
echo '<br>';
echo $url = admin_url( 'admin-ajax.php' ); // https://wpcookbook.local/wp-admin/admin-
ajax.php?test=true
```

Notre fonction hookée sur `admin_url` reçoit l'URL, mais aussi le `$path` passé en paramètre dans les appels à la fonction `admin_url()`. Elle vérifie simplement sa valeur et si on cherche à obtenir l'adresse du fichier `admin-ajax.php`, elle lui ajoute la query string `?test=true`.

Plus tard, on cherche à utiliser l'URL vers le fichier `admin-ajax.php`. Si on utilise la fonction convenablement (c'est-à-dire en utilisant son paramètre) alors le filtre peut faire son travail. Sinon, le filtre ne fait rien. Et ça pourrait créer des bugs !

Ne pas utiliser les paramètres, c'est simplement omettre de passer une information importante à tous les filtres qui vont suivre !

## À retenir

Désolé si ce chapitre était un peu répétitif. Il fallait absolument que vous ayiez conscience des outils que WordPress offre pour naviguer dans ses fichiers. Cette liste n'est pas exhaustive, mais elle comprend les fonctions les plus utiles.

Pour me faire pardonner, voici la version courte :

- Dans une extension, utilisez `plugin_dir_url()` et `plugin_dir_path()` dans votre fichier bootstrap pour définir des constantes que vous allez utiliser dans le reste de vos développements. Easy.
- Dans un thème, `get_theme_file_uri()` et `get_theme_file_path()` sont magiques. Si vous devez absolument charger un fichier dans le parent, utilisez `get_parent_theme_file_uri()` et `get_parent_theme_file_path()`.
- Ne confondez pas `home_url()` et `site_url()`. `home_url()` renvoie l'URL du site, alors que `site_url()` renvoie l'adresse des fichiers du site.
- Utilisez `admin_url()` quand vous devez lier vers un fichier de l'admin, comme `admin-ajax.php` ou `admin-post.php`.
- Il existe des constantes pour ces chemins et URL.
- Utilisez les paramètres des fonctions sans concaténer quand c'est possible.

# Comprendre son thème

Avec WordPress, même s'il existe des milliers de thèmes gratuits et premiums, on n'est que rarement 100% satisfait du thème. On fait toujours un compromis. Pour avoir à 100% ce qu'on veut, on n'a pas le choix, on doit aller dans le thème pour le bidouiller et ajouter ou modifier ce dont on a besoin.

Par exemple pour ajouter un modèle de page pour un type de contenu personnalisé, pour modifier vos styles ou une police de caractères ou pour ajouter des informations à un template existant, on est obligé d'ouvrir un éditeur de code.

Ce chapitre va être un peu théorique, mais il est **absolument nécessaire** de comprendre quels sont les fichiers qui constituent votre thème, comment ils sont organisés et comment ils s'appellent les uns les autres pour construire les pages de votre site. Aussi, on a déjà parlé de certains aspects des thèmes dans les chapitres précédents, donc quelques éléments seront sûrement des rappels pour vous, si vous avez lu de façon linéaire.

## Où se cachent les fichiers de vos thèmes WordPress ?

On a déjà vu comment créer un thème enfant, donc en théorie, j'enfonce une porte grande ouverte ici. Mais si vous avez zappé le chapitre sur la création de thème enfant, voici un rappel : dans une installation classique de WordPress, les thèmes se trouvent dans le dossier `wp-content/themes/`.

Tous les thèmes sont tous bien rangés dans leur dossier et à la racine, on retrouve un certain nombre de fichiers. Ce qui est important de remarquer et de bien comprendre, c'est que tous les thèmes n'ont pas le même nombre de fichiers mais on retrouve les mêmes fichiers dans beaucoup de thèmes.

Par exemple, **tous** les thèmes ont un fichier appelé `index.php`, (quasiment) tous les thèmes ont un fichier appelé `single.php`. Certains thèmes ont un fichier appelé `home.php` ou `front-page.php`.

Tous ces noms de fichiers sont des conventions de WordPress. Votre feuille de style principale doit s'appeler `style.css`, sinon votre thème ne sera pas reconnu. Les fonctionnalités du thème sont à ranger obligatoirement dans `functions.php`, et tous les modèles de page doivent avoir des noms bien précis pour être reconnus par WordPress.

## Les deux fichiers essentiels

Pour que WordPress reconnaise votre thème, il doit contenir au moins deux fichiers : `index.php` et `style.css`. Sans ces deux fichiers, WordPress ne considérera pas votre thème comme un thème valide. Sauf si c'est un thème enfant, auquel cas seul `style.css` est nécessaire.

De plus, `style.css` doit contenir une en-tête particulière. C'est un bloc de commentaires contenant des informations sur le thème que WordPress peut afficher dans l'administration. On en a déjà parlé dans le chapitre sur la création de thème enfant.

Juste pour le fun, dans votre installation locale de WordPress, créez un dossier dans `wp-content/themes/` et appelez-le `mon-super-theme`, puis dans ce dossier, créez deux fichiers `index.php` et `style.css`.

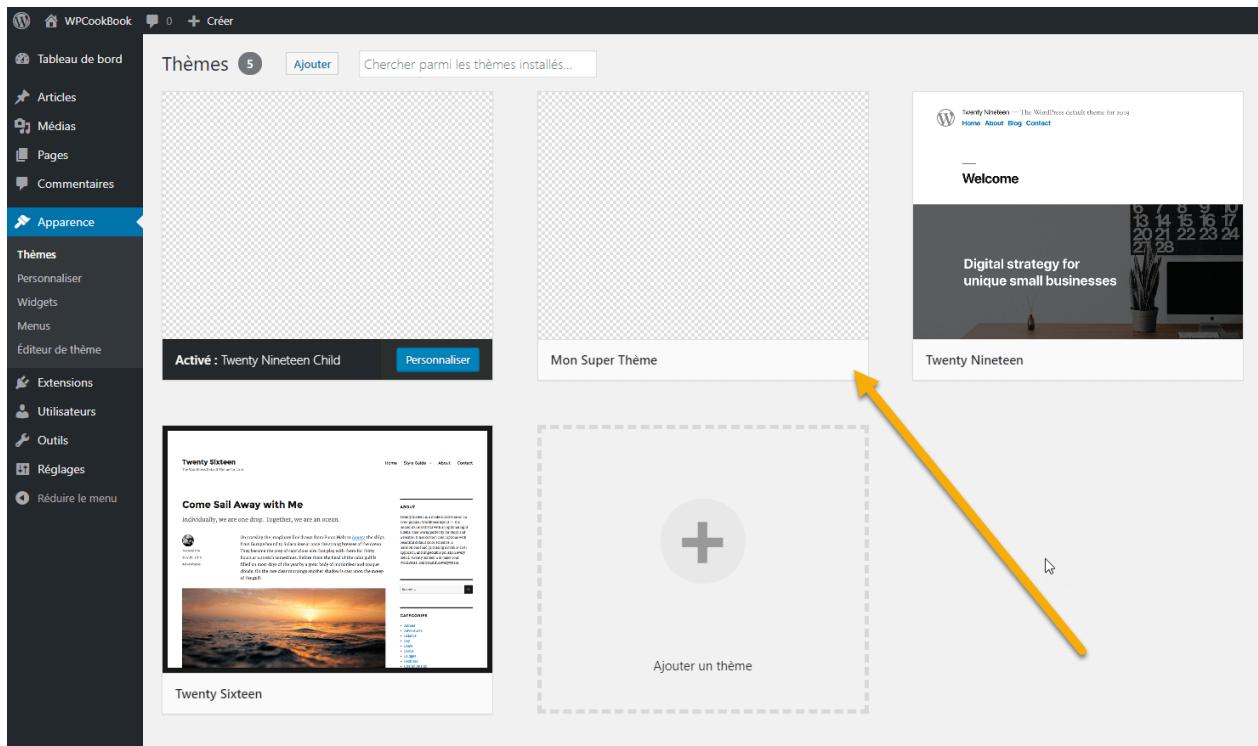
Dans `index.php`, ajoutez :

```
<h1>Hello World !</h1>
```

Dans style.css, ajoutez:

```
/*
 * Theme name: Mon Super Thème
 */
```

Tadaa !! Vous avez créé un thème. Si vous allez dans l'admin, WordPress reconnaît votre thème et vous pouvez l'activer !



Notre thème est disponible dans l'administration

## Les templates

Les thèmes WordPress fonctionnent par templates, ou modèles de page. C'est-à-dire que pour chaque page du même type, le même modèle est utilisé. Donc, un seul fichier est utilisé par type de contenu. C'est WordPress qui va automatiquement vérifier si le fichier correspondant au type de vue existe dans votre thème et le charger, ou se rabattre sur un autre modèle plus générique.

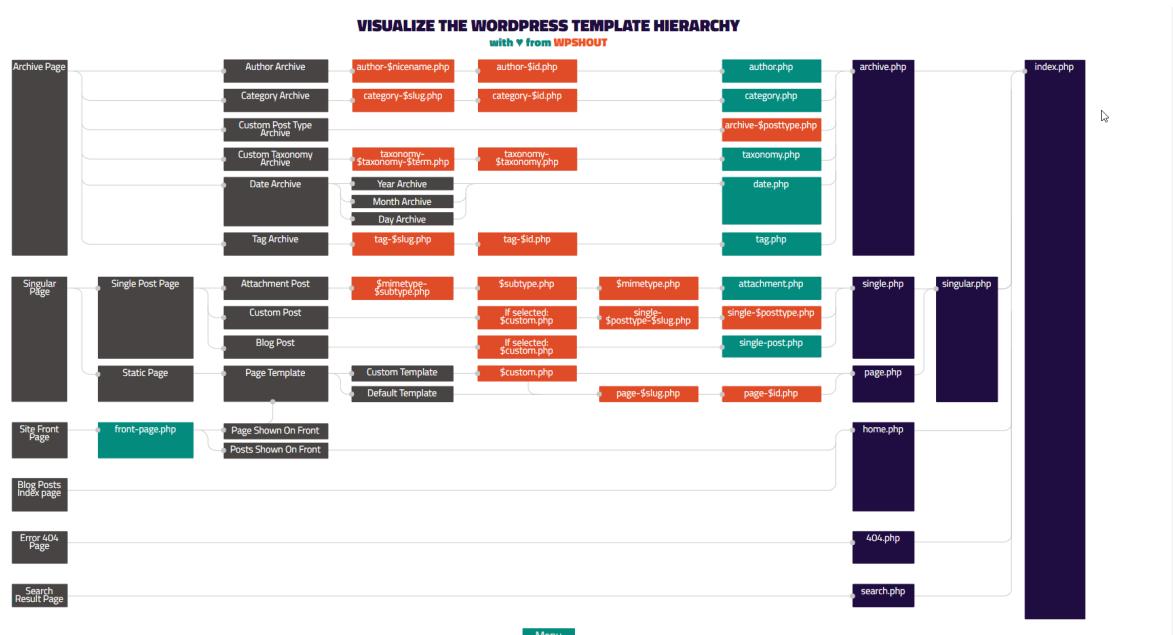
Par exemple, si vous cliquez sur un article, il s'affichera d'une certaine manière. Vous aurez des informations sur l'auteur de l'article, la date de publication et peut-être une colonne latérale. Si vous cliquez sur un autre article, il s'affichera normalement de la même façon. Seul le contenu sera différent. Une page standard n'affichera pas d'information sur la date de publication et n'aura peut être pas de colonne latérale ou pas de commentaire.

C'est simplement parce que les articles et les pages n'utilisent pas le même fichier de template. Quand vous cliquez sur une page, WordPress va chercher un fichier nommé `page.php` pour afficher votre page et quand vous cliquez sur un article, c'est `single.php` que WordPress va chercher.

Là où WordPress est magique, c'est que si vous n'avez pas ces fichiers dans votre thème, ça marche quand même ! C'est juste que WordPress utilisera `index.php` à la place. Ce système de plan B s'appelle la hiérarchie des templates.

## La hiérarchie des templates

Quand vous demandez une page : WordPress va chercher le template le plus spécifique pouvant afficher le contenu désiré. S'il ne le trouve pas, il va chercher si le thème comporte un fichier un peu plus générique. Puis il va encore descendre d'un niveau de spécificité jusqu'à ce qu'il tombe sur un modèle existant qui fera l'affaire. En dernier recours, il utilise `index.php`.



La hiérarchie des templates WordPress

Ce schéma représente la hiérarchie des templates complète. Vous pouvez le trouver sur <https://wphierarchy.com/>. Les types de pages sont listés à gauche sur fond noir et les templates pouvant être utilisés sont listés de gauche à droite, en allant du plus spécifique au moins spécifique. Les templates très spécifiques sont en rouge, les templates secondaires en vert et les templates principaux sont en bleu foncé.

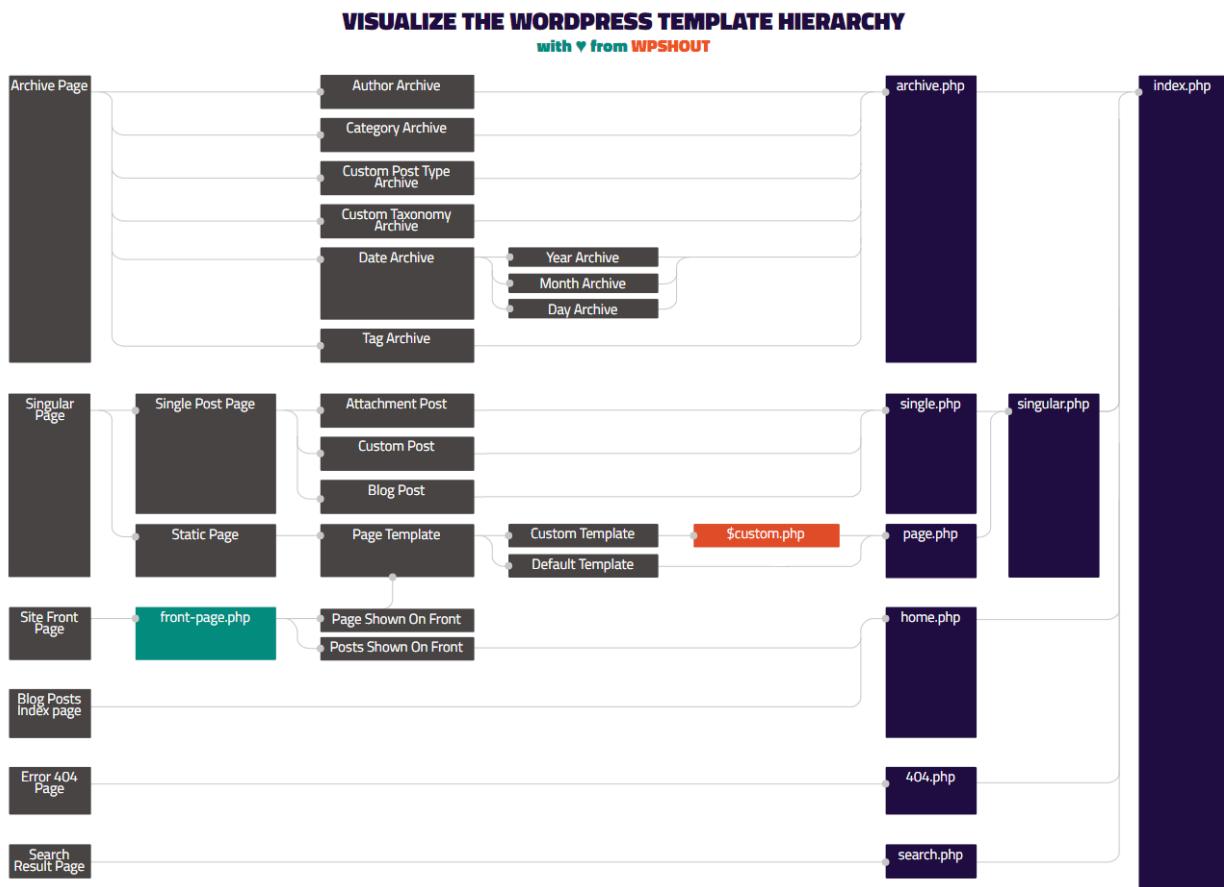
Par exemple, si un utilisateur clique sur une étiquette nommée CSS pour obtenir la liste des articles concernant des astuces CSS, il demande une page d'archives (tout en haut à gauche du schéma). Plus précisément une archive d'étiquettes. Donc, WordPress va chercher dans le thème s'il existe un template nommé `tag-CSS.php`. S'il ne le trouve pas, il va chercher l'identifiant de l'étiquette et il va chercher un template avec cet identifiant, par exemple `tag-3.php` si l'identifiant de l'étiquette CSS est 3.

S'il ne trouve toujours pas ces templates très spécifiques, il va chercher un template plus générique appelé

tag.php, puis archive.php pour finalement se rabattre sur index.php si aucun ne fait l'affaire.

## Les templates génériques

On n'a pas si souvent besoin de templates ultra spécifiques comme ceux notés sur fond rouge, et vous verrez qu'enormément de thèmes n'utilisent que des templates plutôt génériques notés sur fond bleu foncé (ou plus rarement vert). Donc pour mieux comprendre et résumer un peu, on pourrait simplifier notre hiérarchie comme ça :



## La hiérarchie des templates WordPress, version light

Avec ce schéma simplifié, on comprend bien que pour une page d'archives, le template générique par défaut est archive.php, pour un article, c'est single.php, et pour une page c'est page.php, etc.

Ces modèles sont les modèles génériques les plus utilisés, `index.php` étant le template par défaut le plus générique.

## Les templates spécifiques

Si la plupart des thèmes ne proposent que les templates génériques, c'est simplement parce qu'il est impossible de prévoir votre besoin spécifique. Votre site est unique, votre besoin est unique. Le thème va vous fournir la majorité de ce dont vous allez avoir besoin, mais certaines pages ou sections de votre site vont nécessiter un travail de personnalisation du thème.

Donc pour personnaliser votre site, vous allez **obligatoirement** avoir besoin de comprendre le fonctionnement de WordPress pour comprendre quels modèles sont chargés, quand, et connaitre l'ordre de spécificité dans lequel WordPress les cherche.

On a vu au chapitre sur la création de thème enfant que WordPress était malin et cherchait d'abord les modèles de pages dans le thème enfant. Donc dans ce thème enfant, vous allez, selon votre besoin, probablement créer les modèles spécifiques (rouges et verts) de vos pages particulières, et vous allez vous reposer sur les modèles plus génériques bleus foncés de votre thème parent, ou les surcharger si besoin est.

## La page d'accueil

La page d'accueil est un cas un peu spécial dans la hiérarchie. Si vous avez réglé vos préférences de lecture pour afficher une page statique en accueil, alors c'est un modèle normal de page qui sera utilisé (par exemple `page.php`). Si vous choisissez d'afficher vos derniers articles en page d'accueil, alors c'est `home.php` ou `index.php` qui sera utilisé. Jusqu'ici, pas trop de soucis.

Par contre si le thème comporte un fichier `front-page.php`, alors c'est celui-ci qui sera utilisé, que vous ayez choisi d'afficher une page statique ou vos articles en page d'accueil !

C'est une des raisons pour laquelle j'évite d'utiliser ce template dans les thèmes que je publie sur le répertoire officiel. Si mon thème a besoin d'une page d'accueil bien particulière, mais que je souhaite laisser le choix aux utilisateurs d'utiliser une page standard ou une page d'articles, je préfère créer un modèle de page personnalisé pour la page d'accueil qui sera sélectionnable dans l'écran d'édition de la page.

Pour votre site personnel ou le site d'un client, si vous avez conçu une page d'accueil bien particulière et unique, alors utiliser `front-page.php` ne pose pas de souci. Il faut juste avoir conscience que si ce modèle est présent, WordPress l'utilise en ignorant vos réglages de page d'accueil.

## Explorons `twentynineteen`

Pour bien comprendre comment WordPress et votre thème construisent vos pages, on va explorer le thème par défaut de WordPress: `twentynineteen`. Au moment où j'écris cette section du guide, Twenty Twenty n'est pas encore disponible sur le répertoire de WordPress, mais ça ne saurait tarder.

Quand on ouvre `twentynineteen` dans un éditeur de texte, on trouve les fichiers suivants :

- `404.php` va être utilisé par WordPress quand l'utilisateur va sur une page inexistante, en cliquant sur un lien cassé par exemple.
- `archive.php` est un template générique utilisé pour les pages d'archives, par exemple les pages catégories. On a vu qu'on peut aussi créer des templates plus spécifiques pour les pages d'archives, comme `category.php` qui va être utilisé pour afficher les archives d'une catégorie, ou `date.php` pour les archives du mois.
- `functions.php` est un des fichiers les plus importants. Ce n'est pas un fichier de template, mais il contient toutes les fonctionnalités de votre thème, et est chargé automatiquement. Toutes les fonctions personnalisées de votre thème sont dans ce fichier. Même s'il n'est pas obligatoire pour que WordPress reconnaissse votre thème, il est essentiel !
- `search.php` pour l'affichage des résultats de recherche.

- `page.php` pour les pages simples.
- `single.php` pour les articles.
- `index.php` pour tout le reste.
- `style.css` : la feuille de style principale.

Mais on trouve aussi tout un tas de fichiers dont on n'a pas encore parlé.

- `header.php`, `footer.php` et `sidebar.php` : ces fichiers affichent respectivement l'en-tête du site, le pied de page et la barre latérale (zone de widgets). Ils sont appelés par les autres modèles et permettent de ne pas répéter le même code sur tous les fichiers de template. Vous codez une seule fois l'en-tête de votre thème et lappelez sur les autres templates. Ce sont des templates partiels.
- `readme.txt` : c'est la documentation. C'est donc un fichier très important. Vous y trouverez des informations sur l'installation, les changements dans les différentes versions et tout un tas d'autres informations utiles.
- `screenshot.png` : c'est le screenshot qui apparaît dans l'admin de WordPress.
- `style-rtl.css` : C'est la feuille de styles pour les langues se lisant de droite à gauche.
- `print.css` : C'est la feuille de styles pour le print, c'est-à-dire quand vous voulez imprimer une page.
- `style-editor.css` et `style-editor-customizer.css` : Ce sont les feuilles de styles pour le nouvel éditeur de WordPress et l'outil de personnalisation.
- `image.php` : C'est un template spécifique utilisé pour afficher les images. Eh oui, même les images, les vidéos, et les fichiers sons peuvent avoir leur propre modèle de page !

Le thème contient la plupart des templates principaux (bleus foncés sur le schéma de la hiérarchie), sauf `home.php`. Il se repose donc sur `index.php` pour afficher la liste des articles de blog. Aussi, il contient un template spécifique pour les images. Il y a aussi d'autres fichiers utiles pour le développement (`package.json`, les fichiers `.scss`, etc...), mais on n'en parlera pas ici.

## Comment sont affichés nos articles ?

Si on observe le schéma de la hiérarchie des modèles encore une fois, on peut voir que normalement, la page listant les articles (le blog) est affichée en utilisant `home.php` ou `index.php`. Dans `twentynineteen`, ce sera `index.php`.

Vous pouvez tester en ouvrant le fichier dans un éditeur de code et en ajoutant une ligne test affichant un titre.

```
<section id="primary" class="content-area">
    <main id="main" class="site-main">
        <h1>I am index.php !</h1>
        ...
    </main>
</section>
```

# I am index.php !

## Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

 vincent  septembre 12, 2019  Uncategorized  Un commentaire  Modifier

Recherche...

Rechercher

## Articles récents

[Hello world!](#)

C'est bon, on est dans le bon template.

Pour rappel, `index.php` est le template par défaut. C'est le template le plus générique que WordPress va utiliser quand il ne trouve aucun template plus spécifique pour la page requise.

Quand on observe le code du modèle, c'est plutôt léger. Un peu le désert, même. Cela dit, le modèle montre bien comment s'articulent tous les éléments de la page.

### get\_header() et get\_footer()

Le modèle commence par appeler la fonction `get_header()`, ouvre la zone de contenu avec des balises `<section>` et `<main>`, puis appelle `get_footer()` après cette section. Les deux fonctions sont utilisées pour charger les fichiers `header.php` et `footer.php` respectivement.

`header.php` est chargé d'afficher l'en-tête du site, du `<!doctype>` jusqu'à l'ouverture de la zone de contenu, en passant par la balise `<head>`.

`footer.php` est chargé de fermer la zone de contenu, d'afficher le pied de page du site et de fermer les balises `<body>` et `</html>`.

Ces deux fonctions sont utilisées sur tous les modèles du site. Cela permet d'éviter de dupliquer le même code sur tous les modèles de page.

On peut aussi leur passer un paramètre `$name` qui va permettre d'aller chercher un modèle spécial pour votre en-tête ou pied de page nommé `header-$name.php` ou `footer-$name.php`. Ce qui veut dire que vous n'êtes absolument pas limités à un seul et unique modèle d'en-tête et de pied de page ! Vous pouvez parfaitement créer de multiples en-têtes, voire une en-tête différente pour chaque modèle !

## La boucle de WordPress

A l'intérieur de la balise <main>, on trouve un des concepts les plus importants de WordPress : la boucle.

La boucle de WordPress est responsable de l'affichage de la liste des articles ou du contenu demandé sur une page simple. C'est un concept très puissant qu'il faut bien comprendre pour appréhender le développement de thème.

On commence par vérifier s'il y a bien du contenu à afficher.

```
<?php  
if ( have_posts() ) {  
    ...
```

Ce qu'il faut avoir à l'esprit, c'est qu'au moment où WordPress affiche le contenu, la requête en base de données pour ce contenu est déjà faite. Le modèle ne fait qu'afficher ce qu'on lui donne. On verra plus tard comment personnaliser les résultats à afficher, créer de nouvelles boucles, et tout un tas d'autres choses utiles, ne vous inquiétez pas !

S'il y a bien du contenu à afficher, alors on continue.

```
// Load posts loop.  
while ( have_posts() ) {  
    the_post();  
    get_template_part( 'template-parts/content/content' );  
}  
  
// Previous/next page navigation.  
twentynineteen_the_posts_navigation();
```

On a une boucle while qui va boucler sur nos articles. the\_post() va préparer les données de chaque article, avant de l'afficher en utilisant un modèle partiel (template part) content.php qu'il va chercher dans le dossier template-parts/content.

get\_template\_part( string \$slug, string \$name = null ) est une fonction très pratique, qui va permettre d'aller chercher un modèle partiel \${slug}.php ou \${slug}-\${name}.php, et ce d'abord dans le thème enfant puis dans le parent s'il ne l'a pas trouvé.

Ce qui est pratique, c'est qu'on peut jouer avec des variables pour inclure différents modèles. Par exemple, les thèmes supportant les post-formats ont souvent différents fichiers content-\$format.php (content-audio.php, content-video.php, etc...) qu'ils vont inclure en utilisant get\_template\_part('content', get\_post\_format()); la fonction get\_post\_format() renvoyant le format de la publication en question.

C'est donc le modèle partiel appelé par get\_template\_part() qui va contenir le balisage de chacun de nos articles. On va voir tout ça un peu plus loin.

Ensuite, après avoir affiché tous les articles pour cette page, on a une fonction `twentynineteen_the_posts_navigation()` chargée d'afficher la pagination.

```
if ( have_posts() ) {  
    ...  
}  
else {  
    // If no content, include the "No posts found" template.  
    get_template_part( 'template-parts/content/content', 'none' );  
}
```

Enfin, on a un `else` qui vient afficher un modèle partiel différent s'il n'y avait pas de contenu à afficher.

## Les template tags

On a vu que l'article en lui-même était affiché grâce à un modèle partiel `content.php`, que l'on trouve dans le dossier `template-parts/content/`.

Si on ouvre ce fichier dans notre éditeur de code, on peut voir qu'il affiche le contenu de la balise `<article>` contenant chaque article et contient un peu de logique pour gérer les titres.

On peut repérer que chaque article a une balise `<header>`, `<div class="entry-content">`, et `<footer>`, et affiche les éléments de contenu avec des fonctions comme `the_title()` (qui affiche le titre), `the_content()` (qui va afficher le contenu de l'article). Ces fonctions sont appelées des `template tags` et sont chacune responsables de l'affichage d'un élément de contenu.

Il existe des dizaines de `template tags` pour à peu près toutes les informations de base à afficher. Par exemple, `b bloginfo( 'name' )` va afficher le titre de votre site, alors que `the_post_thumbnail()` va afficher l'image mise en avant de votre article. Ce sont ces fonctions qui vont vous permettre de personnaliser les informations que vous voulez présenter dans chaque template.

Aussi, il est possible de créer des `template tags` personnalisés pour vos besoins spécifiques, comme le fait le `twentynineteen` avec `twentynineteen_post_thumbnail()`. Ces `template tags` sont à ranger dans votre fichier `functions.php` ou un autre fichier inclus dans celui-ci.

## Ce qu'il faut retenir

- Un thème est reconnu par WordPress s'il contient un fichier `index.php` et un fichier `style.css` avec un bloc de commentaire spécifique en en-tête.
- Un thème est constitué de différents modèles de pages, et ces modèles doivent avoir des noms spécifiques pour être reconnus par WordPress.
- Quand il cherche quel modèle utiliser, WordPress cherche du plus spécifique au moins spécifique, en se rabattant sur `index.php` en dernier lieu.
- Aussi, WordPress cherche si les modèles existent dans le thème enfant, puis dans le parent.
- Les modèles `header.php`, `footer.php` et `sidebar.php` sont utilisés pour afficher l'en-tête du site,

le pied de page et la colonne latérale, respectivement. Ils sont appelés à l'aide des fonctions `get_header()`, `get_footer()`, et `get_sidebar()`.

- La boucle de WordPress est une simple bouclewhile. On utilise les fonctions `have_posts()` pour vérifier s'il y a encore des éléments à afficher, et `the_post()` pour les préparer.
- On utilise la fonction `get_template_part()` pour charger un modèle partiel. Comme pour les modèles standards, WordPress cherche d'abord s'ils existent dans le thème enfant, puis dans le parent.
- Les template tags sont des fonctions servant à afficher les différents éléments de contenu. Cela peut être le titre, la date de publication, le contenu principal, les catégories, etc. Il y a des dizaines de template tags déjà existantes, et on peut facilement créer les nôtres.

Si vous avez bien compris comment fonctionnent les thèmes, alors vous êtes en mesure de trouver quels modèles gouvernent l'affichage de tel ou tel type de contenu, les dupliquer dans votre thème enfant et effectuer vos personnalisations.

Les autres recettes de ce guide vont vous aider à personnaliser les contenus à afficher, personnaliser la boucle principale, créer d'autres boucles, déclarer et afficher des zones de widgets, déclarer et afficher des menus, etc. Bref, tout ce dont vous avez besoin pour prendre le contrôle de votre thème, ou créer le vôtre de toute pièce !

Ready ?

# Charger les ressources JS et CSS correctement

Imaginons que nous ayons besoin d'ajouter une feuille de styles supplémentaire pour notre thème.

Il y a plusieurs façons de charger le CSS et le JS qui "marchent" : il y a la façon brutale, la façon WordPress presque correcte et la "vraie" façon WordPress.

## La balise hardcodée

Une façon un peu brutale de charger vos ressources sur le devant du site est de hardcoder la balise `<link>` ou `<script>` dans le template `header.php` ou `footer.php` de votre thème.

Par exemple dans `twentynineteen` :

```
...
<!doctype html>
<html <?php language_attributes(); ?>>
<head>
    <meta charset=<?php bloginfo( 'charset' ); ?>" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="profile" href="https://gmpg.org/xfn/11" />
    <link rel="stylesheet" href="my-custom-styles.css">
    <?php wp_head(); ?>
</head>

<body <?php body_class(); ?>>
...
```

Ça va marcher, pas de souci. Mais quand vous allez avoir besoin d'un peu plus de contrôle, par exemple charger les styles de façon conditionnelle sur une page précise, cela va complexifier grandement votre fichier `header.php`, car vous aurez besoin d'ouvrir des balises PHP pour y insérer votre logique.

Et ne parlons pas de la mise à jour du thème si vous faites ça directement dans le parent. Certes, vous pouvez dupliquer entièrement le fichier `header.php` dans votre thème enfant, mais bon. Dupliquer le fichier pour y insérer des balises hardcodées, c'est pas très propre tout ça.

L'intérêt d'un template, c'est de séparer la logique de l'affichage. Du coup, dans notre exemple précédent, on va à l'encontre de ce principe. Donc on va éviter de faire comme ça.

## Hook sur `wp_head`

Dans la balise `<head>` du thème, on trouve la fonction `wp_head()`, qui a pour seul rôle d'exposer le hook du même nom.

Une autre façon de faire qui va nous permettre d'utiliser de la logique sans polluer le template, c'est de se greffer sur le hook `wp_head`.

On va supprimer notre balise hardcodée, et dans le fichier `functions.php` de notre thème, on va donc se hooker sur `wp_head`.

```
// in plugin file or functions.php
add_action( 'wp_head', 'wpcookbook_head' );
/**
 * Adds <link> tag and other scripts to <head> tag.
 */
function wpcookbook_head() {
    if ( is_page( 'sample-page' ) ) {
        $stylesheet_url = get_theme_file_uri( 'assets/css/my-custom-styles.css' );
        printf( '<link rel="stylesheet" href="%s">', esc_url( $stylesheet_url ) );
    }
}
```

Dans cette fonction, on va simplement vérifier que l'on est sur une page spécifique ayant pour slug '`sample-page`' grâce à la fonction `is_page()`, et si on est sur la bonne page, on va chercher l'URL de notre feuille de styles avec `get_theme_file_uri()` (cf. chapitre 9) et on affiche la balise avec `printf()` (cf chapitre 7).

La fonction `esc_url()` va échapper l'URL passée en paramètre pour s'assurer qu'elle soit affichable sans risque (en remplaçant les &, les espaces, et d'autres caractères). On peut aussi lui passer un tableau de protocoles en deuxième paramètre et un contexte sous forme de chaîne de caractères en troisième paramètre, mais on en a rarement besoin.

On a parlé brièvement des fonctions d'échappement dans le chapitre 7, combinées aux fonctions de traductions. Pas de panique, on en rencontrera encore plusieurs au fur et à mesure des recettes de ce guide, mais `esc_url` est une des fonctions que vous utiliserez le plus, avec `esc_html()` et `esc_attr()`.

Si l'on navigue sur la page d'exemples de WordPress qui a pour slug `sample-page` en anglais, alors notre feuille de styles est chargée, mais uniquement sur cette page.

WPCookBook — Apprenez à développer pour WordPress

[Page](#) [Article](#) [Catégorie](#) [Contact](#)

## Sample Page

This is an example page. It's different from a blog post because it will stay in one place and will show up in your site navigation (in most themes). Most people start with an About page that introduces them to potential site visitors. It might say something like this:

Hi there! I'm a bike messenger by day, aspiring actor by night, and this is my website. I live in Los Angeles, have a great dog named Jack, and I like piña coladas. (And gettin' caught in the rain.)

...or something like this:

The XYZ Doohickey Company was founded in 1971, and has been providing quality doohickeys to the public ever since. Located in Gotham City, XYZ employs over 2,000 people and does all kinds of awesome things for the Gotham community.

As a new WordPress user, you should go to [your dashboard](#) to delete this page and create new pages for

Un beau background jaune

Cette méthode fonctionne et est un peu plus propre que la balise hardcodée, mais ce n'est pas encore parfait.

## wp\_enqueue\_style() et wp\_enqueue\_script()

En réalité, voici la meilleure (la seule !) façon de charger des styles :

```
// in functions.php
add_action( 'wp_enqueue_scripts', 'wpcookbook_styles_and_scripts' );
/**
 * Enqueues stylesheet and scripts.
 */
function wpcookbook_styles_and_scripts() {
    if ( is_page( 'sample-page' ) ) {
        $stylesheet_url = get_theme_file_uri( 'assets/css/my-custom-styles.css' );
        wp_enqueue_style( 'my-custom-styles', esc_url( $stylesheet_url ) );
    }
}
```

On se hooke sur `wp_enqueue_scripts`, pour utiliser ensuite la fonction `wp_enqueue_style()` pour charger nos styles.

`wp_enqueue_scripts` est le hook sur lequel WordPress s'attend à ce que les extensions et thèmes se greffent pour déclarer les styles et scripts à charger sur le devant du site, à l'aide des fonctions

`wp_enqueue_style()` et `wp_enqueue_script()`.

Pour être plus précis, les fonctions `wp_enqueue_style()` et `wp_enqueue_script()` ne chargent pas vraiment les ressources directement, mais les ajoutent à une file d'attente de ressources à charger, et WordPress va simplement écrire automatiquement les balises `<link>` ou `<script>` correspondantes sur `wp_head` ou `wp_footer`.

## Charger les styles avec `wp_enqueue_style()`

```
wp_enqueue_style( string $handle, string $src = '', array $deps = array(),
string|bool|null $ver = false, string $media = 'all' )
```

`$handle` est l'identifiant de la feuille de styles, `$src` est l'URL du fichier que l'on va obtenir avec les fonctions décrites au chapitre 9.

`$deps` est un tableau de dépendances. C'est-à-dire que vous pouvez passer un tableau d'identifiants de feuilles de styles (comme le `$handle` que vous venez de lui passer) et WordPress va faire en sorte que ces feuilles de styles soient chargées avant la vôtre. Magique, je vous dis.

On peut aussi passer un numéro de version `$ver`, qui sera ajouté comme chaîne de requêtes à la fin de l'URL de la ressource chargée. L'intérêt est que si le numéro de version change, le navigateur détecte que la feuille de styles a été mise à jour, et ne va donc pas vous servir celle qu'il a en cache. Mais en développement, cela peut embêter le monde et vous empêcher de voir vos changements instantanément. Donc quand vous développez, vous pouvez passer `time()` pour éviter les soucis de cache. En effet, le numéro de version changeant à chaque rafraîchissement de page, vous aurez toujours une version fraîche de votre feuille de styles.

`$media` permet de passer un type de média (`screen`, `print`, ou `all`) ou carrément une media query (`max-width: 1280px`) pour laquelle il faut charger la feuille de styles. C'est surtout utile si vous avez une feuille de styles spécifique pour le `print`. Pour le reste, en général les media queries sont dans votre feuille de styles.

Seuls les deux premiers paramètres sont obligatoires dans ce cas, mais souvent je vais jusqu'au 4e pour pouvoir explicitement dire à WordPress d'ajouter un numéro de version différent de la version de WordPress, car par défaut la version de WordPress est utilisée comme numéro de version. C'est rarement une bonne idée de dévoiler la version de WordPress utilisée !

Du coup, on va ajuster notre morceau de code et ajouter les 3e et 4e paramètres :

```
// in functions.php
add_action( 'wp_enqueue_scripts', 'wpcookbook_styles_and_scripts' );
/**
 * Enqueues stylesheet and scripts.
 */
function wpcookbook_styles_and_scripts() {
    if ( is_page( 'sample-page' ) ) {
        $stylesheet_url = get_theme_file_uri( 'assets/css/my-custom-styles.css' );
        wp_enqueue_style( 'my-custom-styles', esc_url( $stylesheet_url ), array(), time() );
    }
}
```

Voilà qui est mieux.

## Charger les scripts avec `wp_enqueue_script()`

```
wp_enqueue_script( string $handle, string $src = '', array $deps = array(),
string|bool|null $ver = false, bool $in_footer = false )
```

Pour charger vos fichiers JavaScript, on va utiliser la fonction `wp_enqueue_script()` qui va prendre presque les mêmes paramètres.

Comme `wp_enqueue_style()`, on doit lui passer un identifiant (slug) et une URL vers le fichier à charger, et on peut lui passer un tableau de dépendances, ainsi qu'un numéro de version.

Par exemple, on peut lui passer `array( 'jquery' )` si notre JavaScript dépend de jQuery. La bibliothèque jQuery est incluse dans WordPress, et est utilisée partout dans l'administration. Elle est enregistrée comme script disponible et il suffit donc de passer son identifiant pour la charger correctement en tant que dépendance.

Notez qu'il vaut mieux utiliser la version du cœur de WordPress même si elle est un peu vieillote. En effet, si vous chargez une version fraîche de jQuery dans votre thème et qu'une extension utilise aussi jQuery mais en chargeant celle de WordPress, vous risquez d'avoir les deux bibliothèques chargées et des conflits peuvent apparaître. Elle est déjà dans le cœur, donc autant utiliser celle-ci.

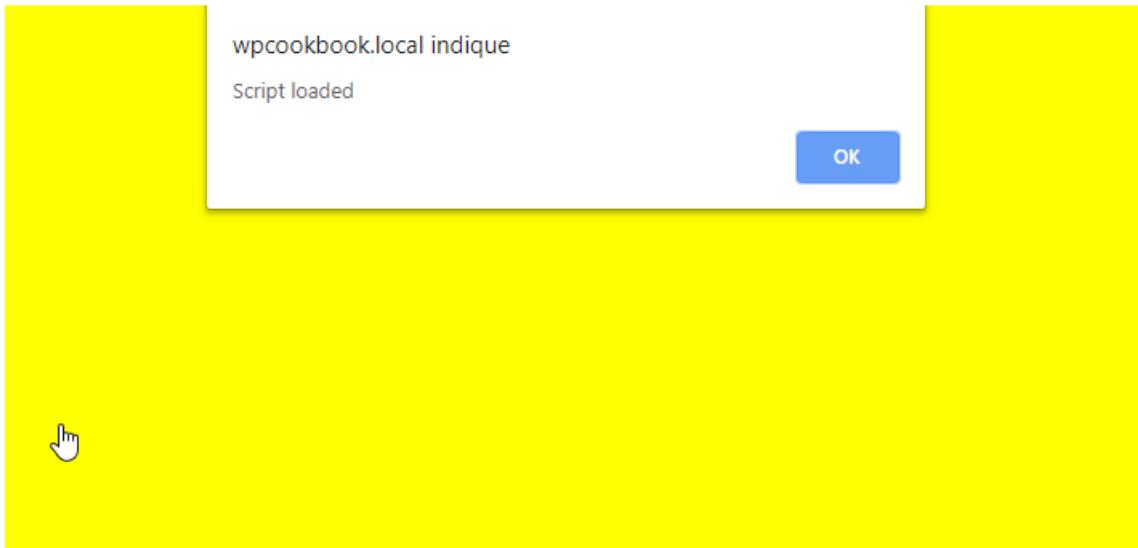
Les différents scripts déjà enregistrés par WordPress et donc disponibles sont listés sur la page de documentation de la fonction : [https://developer.wordpress.org/reference/functions/wp\\_enqueue\\_script/](https://developer.wordpress.org/reference/functions/wp_enqueue_script/).

Le dernier paramètre `$in_footer` est très intéressant. Il permet de dire à WordPress si vous voulez que vos scripts soient chargés au plus tôt (dans la balise `<head>`) ou alors en bas de page (plus exactement sur `wp_footer`). Les ressources chargées dans la balise `<head>` vont bloquer le chargement et le rendu du site, donc souvent, on recommande de charger les scripts non-essentiels en bas de page.

Pour l'exemple, on va créer un petit script `alert.js` dans notre dossier `assets/js` qui fait juste une alerte avec un message.

Pour charger ce script on va utiliser `wp_enqueue_script()`, mais on va charger le script dans la balise `<head>`.

```
// in functions.php
add_action( 'wp_enqueue_scripts', 'wpcookbook_styles_and_scripts' );
/**
 * Enqueues stylesheet and scripts.
 */
function wpcookbook_styles_and_scripts() {
    if ( is_page( 'sample-page' ) ) {
        ...
    }
    wp_enqueue_script( 'alert', get_theme_file_uri( 'assets/js/alert.js' ) );
}
```



En se chargeant et s'exécutant de suite, le script bloque le reste de la page

Pour corriger cela, on va ajuster notre appel à la fonction :

```
wp_enqueue_script( 'alert', get_theme_file_uri( 'assets/js/alert.js' ), array(), time(), true );
```

WPCookBook — Apprenez à développer pour WordPress

[Page](#) [Article](#) [Catégorie](#) [Contact](#)

wpcookbook.local indique  
Script loaded

OK

# Hello world!



Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

vincent · septembre 12, 2019 · Uncategorized · Un commentaire · Modifier

En chargeant le script en bas de page, celle-ci s'affiche.

En passant simplement `true` comme valeur pour le paramètre `$in_footer`, notre page va s'afficher, mais s'il y a d'autres ressources chargées après la nôtre, celles-ci seront bloquées à cause du `alert()`, jusqu'à ce qu'on clique sur OK. Par exemple, malgré le fait que je suis connecté, la barre d'admin ne

s'affiche pas.

## wp\_register\_script() et wp\_register\_style()

Ces fonctions permettent de déclarer des styles et scripts, mais ne les ajoutent pas à la file d'attente des ressources à charger. Elles ajoutent simplement les styles et scripts à la liste des ressources disponibles connues par WordPress. C'est tout.

Une fois déclarées, ces ressources peuvent être chargées très simplement avec wp\_enqueue\_style() ou wp\_enqueue\_script().

```
// in functions.php
add_action( 'wp_enqueue_scripts', 'wpcookbook_styles_and_scripts' );
/**
 * Enqueues stylesheet and scripts.
 */
function wpcookbook_styles_and_scripts() {
    $stylesheet_url = get_theme_file_uri( 'assets/css/my-custom-styles.css' );
    wp_register_style( 'my-custom-styles', esc_url( $stylesheet_url ), array(), time() );
    if ( is_page( 'sample-page' ) ) {
        wp_enqueue_style( 'my-custom-styles' );
    }

    wp_register_script( 'alert', get_theme_file_uri( 'assets/js/alert.js' ), array(), time(),
true );
    wp_enqueue_script( 'alert' );
}
```

Ici, on travaille en deux temps. On déclare d'abord nos styles et scripts avec wp\_register\_style() et wp\_register\_script(), puis on les charge quand on en a besoin avec wp\_enqueue\_style() et wp\_enqueue\_script(). Vu que nos ressources sont déjà connues de WordPress, il suffit de passer leur identifiant pour les charger.

Dans notre exemple initial, on a uniquement utilisé wp\_enqueue\_style() et wp\_enqueue\_script() car elles permettent aussi de déclarer une ressource si elle n'existe pas déjà, puis de la charger. La fonction est un peu "deux en un".

Les fonctions wp\_register\_script() et wp\_register\_style() sont surtout utiles pour déclarer des scripts et styles utilisés comme dépendances, ou alors si vous voulez grouper toutes vos déclarations au même endroit mais que vous avez besoin de charger à des endroits différents.

Mais ce qu'il faut bien comprendre c'est qu'une ressource doit être déclarée et connue de WordPress pour pouvoir la charger correctement et au bon moment, et que wp\_enqueue\_style() et wp\_enqueue\_script() permettent à la fois de déclarer le cas échéant et de charger vos ressources.

## `wp_dequeue_style()` et `wp_dequeue_script()`

Ces deux fonctions prennent simplement un identifiant \$handle comme paramètre, et enlèvent les ressources correspondantes de la file d'attente des ressources à charger.

Elles permettent simplement de ne pas charger une ressource.

## Chargez vos ressources au bon moment

### Charger les ressources sur le bon hook

Le moment attendu pour charger vos ressources sur le devant du site, c'est le hook `wp_enqueue_scripts`. Mais il existe d'autres hooks sur lesquels vous pouvez vous greffer dans d'autres situations.

Par exemple, vous pouvez vous greffer sur `admin_enqueue_scripts` pour charger des ressources uniquement disponibles dans l'administration, et sur `login_enqueue_scripts` pour la page de connexion.

```
// in plugin file or functions.php
add_action( 'admin_enqueue_scripts', 'wpcookbook_admin_scripts', 10, 1 );
/**
 * Loads styles only on the general options page
 */
function wpcookbook_admin_scripts( $hook_suffix ) {
    if ( 'options-general.php' === $hook_suffix ) {
        wp_enqueue_style( 'admin-styles', get_theme_file_uri( 'assets/css/admin.css' ),
array(), time() );
    }
}
```

Le hook `admin_enqueue_scripts` fournit le slug de la page `$hook_suffix` qui permet de charger les ressources uniquement sur certaines pages. C'est très pratique.

Dans votre fonction de rappel, vous pouvez utiliser les fonctions de WordPress comme `is_page()`, `is_home()`, `is_front_page()`, ou vérifier le type de contenu affiché avec `get_post_type()` pour charger vos ressources sur le devant du site uniquement quand c'est nécessaire. Vous pouvez aussi lire la globale `$pagenow` dans l'administration.

### Charger les ressources dans le modèle

Ces hooks couvrent les cas courants, mais il existe certains cas où il n'est pas possible de prévoir quand la ressource est requise.

Imaginez que, dans une extension, l'on ait besoin de créer un code court [wpcookbook\_table] pour afficher un tableau. Mais ce tableau a besoin d'un peu de CSS et de JS personnalisé. Or, on ne sait pas sur quelle page va se trouver le code court [wpcookbook\_table], donc on ne peut pas utiliser la fonction `is_page()` ou autre.

Mais on sait que les fonctions `wp_register_style()` et `wp_register_script()` (tout comme `wp_enqueue_style()` et `wp_enqueue_script()`) ne font qu'enregistrer des ressources à charger plus tard.

Du coup, on peut utiliser ces fonctions directement dans la fonction de rappel de notre code court

```
// in main plugin file
add_shortcode( 'wpcookbook_table', 'wpcookbook_table' );
/**
 * Adds a simple shortcode displaying a table
 *
 * @param string $atts shortcode attributes
 */
function wpcookbook_table( $atts ) {
    // Enqueue our scripts.
    $stylesheet_url = plugins_url( 'assets/css/custom-table.css', __FILE__ ); // Assuming this
    // function is in bootstrap file
    $script_url     = plugins_url( 'assets/js/custom-table.js', __FILE__ );
    wp_enqueue_style( 'custom-table-styles', esc_url( $stylesheet_url ), array(), time() );
    wp_enqueue_script( 'custom-table-script', esc_url( $script_url ), array(), time(), true );

    // Get the HTML content.
    ob_start();
    printf( '<h2>%s</h2>', __( 'My custom table', 'wpcookbook' ) );
    /* include table template here with include() for example */
    return ob_get_clean();
}
```

Ici, on enregistre la feuille de styles et le script, qui vont se charger quand WordPress tombera sur `wp_footer`, mais ces ressources seront chargées uniquement sur les pages contenant le code court `[wpcookbook_table]`. En observant le code source de la page, on voit que nos ressources sont chargées en bas de page, avec tous les autres scripts.

```
132
133 </div><!-- #page -->
134
135
136 <link rel='stylesheet' id='custom-table-styles-css' href='https://wpcookbook.local/wp-
137 content/plugins/11-loading-css-js/assets/css/custom-table.css' type='text/css' media='all' />
138 <script type='text/javascript' src='https://wpcookbook.local/wp-includes/js/admin-bar.min.js?
139 ver=5.3'></script>
140 <script type='text/javascript' src='https://wpcookbook.local/wp-
141 content/themes/twentynineteen/js/priority-menu.js?ver=1.1'></script>
142 <script type='text/javascript' src='https://wpcookbook.local/wp-
143 content/themes/twentynineteen/js/touch-keyboard-navigation.js?ver=1.1'></script>
144 <script type='text/javascript' src='https://wpcookbook.local/wp-includes/js/wp-embed.min.js?
145 ver=5.3'></script>
146 <script type='text/javascript' src='https://wpcookbook.local/wp-content/plugins/11-loading-css-
147 js/assets/js/custom-table.js?ver=1575922227'></script>
```

Nos ressources sont bien chargées en bas de page.

## Pourquoi ?

On a vu les fonctions et hooks à utiliser pour charger les ressources CSS et JS. Mais pourquoi faut-il absolument utiliser ces méthodes ?

Simplement parce qu'en faisant ainsi vous passez par l'API mise en place par WordPress et de ce fait il a connaissance de vos ressources. Elles sont inscrites au registre si vous voulez.

Du coup, WordPress peut manipuler vos ressources et elles sont aussi disponibles pour les autres extensions qui auraient besoin d'y accéder.

Par exemple, si vous voulez ne pas inclure les styles par défaut d'une certaine extension, vous pouvez. Si l'extension en question fait les choses proprement et utilise `wp_enqueue_script()` et `wp_enqueue_style()`, alors vous pouvez intervenir et utiliser `wp_dequeue_style()` et `wp_dequeue_script()` au bon moment pour éviter de charger ses ressources CSS et JS et utiliser les vôtres.

Quel est le bon moment ? Cherchez simplement quand l'extension charge ses scripts, et greffez-vous au même endroit. Il y a de grandes chances que ce soit sur `wp_enqueue_scripts`. Vous aurez probablement à jouer un peu avec les priorités, mais ça, je vous laisse chercher !

## Ce qu'il faut retenir

- On ne hardcode pas de balises `<script>`, `<link>` ou `<style>`.
- On utilise `wp_register_style()`, `wp_register_script()` pour déclarer des ressources.
- On utilise `wp_enqueue_style()` et `wp_enqueue_script()` pour déclarer nos ressources si besoin et les mettre dans la file d'attente des ressources à charger.
- On se hooke généralement sur `wp_enqueue_scripts` pour les ressources à utiliser sur le devant du site, `admin_enqueue_scripts` pour l'administration et `login_enqueue_scripts` pour la page de connexion.
- On peut utiliser `wp_enqueue_style()` et `wp_enqueue_script()` directement dans des templates. Les ressources correspondantes seront chargées sur `wp_footer`, donc dans le bas de page.

# Les fonctions indispensables de tout thème WordPress

Pour développer un thème WordPress correctement et garantir le maximum de compatibilité avec le maximum d'extensions, il y a un certain nombre de principes à respecter et de passages obligés.

Parmi ces contraintes, il y a l'**obligation** d'utiliser certaines fonctions indispensables. En effet, celle-ci sont critiques, car la plupart d'entre elles exposent des hooks de WordPress essentiels. Si on n'utilise pas ces fonctions dans nos thèmes, une grande partie des extensions et même du cœur de WordPress risque de ne pas fonctionner correctement, voire pas du tout.

## Obligatoire ? Pas obligatoire ?

C'est une question de contexte. Il faut penser en priorité à l'utilisateur final.

Certaines fonctions sont indispensables car WordPress lui-même les utilise et en a besoin dans votre thème. Si vous ne les incluez pas, ça ne marche pas, tout simplement. D'autres sont indispensables, car beaucoup d'extensions du répertoire officiel comptent sur leur utilisation pour y greffer leur fonctionnalité. Donc si vous ne les incluez pas, ces extensions du répertoire ne vont pas fonctionner.

Est-ce un problème ? Sur votre site personnel ou le site d'un client cela pourrait ne pas en être un !

Mais tout au long de ce guide, on est parti du principe que l'on va respecter au maximum les standards de WordPress, pour une qualité maximum, une compatibilité maximum avec tout ce qui existe déjà, mais surtout pour exclure le minimum d'utilisateurs existants de WordPress

Votre thème n'a pas à garantir la compatibilité avec toutes les extensions existantes (car c'est impossible), mais utiliser `the_content()` parce que les extensions de partage se greffent dessus, c'est compliqué ? Respecter tous les standards d'accessibilité, c'est difficile (il faut essayer de tendre vers ça cependant), mais inclure un `skip link`, vous croyez que c'est si compliqué ? Franchement ?

La communauté WordPress est très riche et variée, il faut donc veiller au maximum à inclure les fonctionnalités auxquelles s'attendent ses utilisateurs.

Donc oui et non, certaines fonctions ne sont pas forcément obligatoires, mais en partant du principe que vous allez partager votre thème ou extension sur le répertoire avec potentiellement des milliers d'utilisateurs, vous vous forcerez à prendre en compte un maximum de besoins, vous augmenterez la qualité de votre code et votre compréhension de WordPress et de ses utilisateurs. C'est plutôt cool non ?

Voici donc quelques fonctions indispensables pour tout thème WordPress. Indispensables car sinon rien ne fonctionne ou indispensables car elles permettent d'implémenter une fonctionnalité attendue.

## `language_attributes()`

Cette fonction va simplement ajouter l'attribut `lang` et la bonne valeur à votre balise `<html>`. Elle va même ajouter la direction du texte pour les langues se lisant de droite à gauche. C'est tout simple, mais c'est essentiel pour l'accessibilité de votre thème.

## wp\_head()

La fonction `wp_head()` ne fait pas grand chose seule. Elle expose simplement le hook du même nom. Mais ce hook est **absolument indispensable**.

Cette fonction est à placer dans la balise `<head>` du fichier `header.php` de votre thème, juste avant la balise fermante, et elle permet à WordPress, toutes vos extensions et aussi à votre thème lui-même d'écrire dans cette balise `<head>`.

Par exemple, c'est sur ce hook que certaines extensions écrivent d'importantes métadonnées pour le référencement. Les balises `<link>` et `<style>` de vos styles sont aussi à écrire dans la balise `<head>`.

Comme on a vu au chapitre sur le chargement des ressources, harcoder une référence absolue à une feuille de styles dans une balise `<link>` est un gros NO-NO. Carton rouge ! Simplement parce que WordPress n'en a pas connaissance. Donc pas possible de la déhooker si besoin ni de s'y référer de quelque façon que ce soit.

Donc, pour faire ça correctement, on appelle `wp_head()` dans la balise `<head>` du thème et on utilise les fonctions de WordPress `wp_enqueue_script()` et `wp_enqueue_style()` pour inscrire les balises `<script>` et `<link>` du thème. Pas de hardcoding. Comme dans le fichier `header.php` de `twentynineteen` :

```
<head>
    <meta charset="<?php bloginfo( 'charset' ); ?>" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="profile" href="https://gmpg.org/xfn/11" />
    <?php wp_head(); ?>
</head>
```

C'est tout. Il n'y a rien d'autre à faire.

## wp\_footer()

Celle-ci fonctionne de la même façon que `wp_head()`, sauf qu'elle se place dans le footer, juste avant la balise `</body>` fermante.

Indispensable elle aussi, elle expose le hook du même nom qui va permettre à WordPress et vos extensions d'écrire les balises `<script>` nécessaires pour que leur magie opère. Sans ce hook, de nombreuses extensions risquent de ne pas fonctionner correctement.

Idem que pour `wp_head()`, écrire des balises `<script>` en dur dans vos templates mérite un carton rouge ! Plus généralement, on n'écrit pas de références absolues ou relatives directement dans nos templates, mais on passe toujours par une fonction de WordPress. Mais ça, si vous avez tout lu jusqu'ici, vous le savez déjà !

Dans le fichier footer.php de twentynineteen, cela donne:

```
</div><!-- .site-info -->
</footer><!-- #colophon -->
</div><!-- #page -->

<?php wp_footer(); ?>

</body>
</html>
```

## wp\_body\_open()

Cette nouvelle fonction date de la version 5.2 de WordPress. Comme ses grandes soeurs, elle expose le hook du même nom, et c'est tout.

Cette fonction est à placer juste après la balise <body> ouvrante. Elle permet aux auteurs d'extensions d'ajouter des métadonnées et autres scripts juste avant le contenu de la page.

Si vous voulez ajouter cette fonction et que votre thème reste rétro-compatible avec les anciennes versions de WordPress, il faudra ajouter un snippet dans votre fichier functions.php.

```
if ( ! function_exists( 'wp_body_open' ) ) {
    function wp_body_open() {
        do_action( 'wp_body_open' );
    }
}
```

Une autre solution consisterait à ajouter simplement la ligne do\_action( 'wp\_body\_open' ); juste après la balise <body>, mais autant respecter le format attendu par WordPress et utiliser la fonction.

## post\_class() & body\_class()

Ces fonctions permettent à WordPress d'ajouter des classes très utiles à votre balise <body> et vos balises <article> de votre contenu. Par exemple, avec post\_class() vos articles auront une classe selon leur format, leur catégorie, leur statut, leur identifiant, etc...

Encore une fois, la sortie de ces fonctions est filtrable et ces filtres sont utilisés par de nombreuses extensions. Il est donc fortement recommandé de les utiliser.

## get\_template\_part()

Si vous voulez découper votre thème en plusieurs petits templates réutilisables et en faciliter la maintenance, alors get\_template\_part() est votre alliée.

On a déjà rencontré cette fonction à plusieurs reprises. Elle permet d'aller chercher un modèle partiel et de l'inclure directement. Ce qui est pratique, c'est qu'elle permet de chercher un modèle d'abord dans le

thème enfant, puis dans le parent. C'est pourquoi il est indispensable de l'utiliser !

Ne pas l'utiliser et faire un bête `include` pourrait fonctionner, mais ça enlèverait à vos utilisateurs la possibilité de surcharger ces modèles dans leur thème enfant.

## the\_title() et the\_content()

Ici on entre dans les fonctions qui affichent du contenu.

La première fonction affiche simplement le titre de votre publication, et la deuxième le contenu. Comme la plupart des fonctions affichant du contenu, la sortie de ces fonctions est filtrable et de nombreuses extensions utilisent ces filtres pour ajouter du contenu ou le traiter d'une façon ou d'une autre.

Par exemple, les extensions ajoutant des boutons de partage après le contenu utilisent souvent le filtre `the_content`, donc si vous accédez au contenu et l'affichez d'une autre façon, vous ne passerez pas par ce filtre, donc l'extension ne pourra afficher ses boutons.

Ici, je ne donne qu'un exemple assez simple, mais le nombre d'extensions se hookant sur `the_title` et `the_content` est juste énorme. Si vous n'utilisez pas ces fonctions, vous allez vous faire des ennemis.

## wp\_link\_pages()

Cette fonction permet d'afficher les liens de pagination pour un contenu séparé en plusieurs pages.

WordPress permet de diviser le contenu d'une publication pour l'afficher sur plusieurs pages, en utilisant des quicktags `<!--nextpage-->` lors de l'édition du contenu. Si vous omettez cette fonction, alors le contenu sous la première balise ne sera pas affiché, et les utilisateurs n'auront aucun moyen de lire la suite de la publication.

Ne pas mettre cette fonction revient à priver les utilisateurs de WordPress de cette fonctionnalité. Encore une fois, pour votre site personnel ou le site d'un client, vous pouvez l'omettre si vous êtes sûr que l'utilisateur final n'utilisera pas cette fonctionnalité, mais pour ce guide, on va essayer de respecter le plus possible les standards de WordPress. Donc, on ne réfléchit pas trop, et on l'ajoute !

La fonction prend un tableau d'arguments en paramètres. Voici les paramètres par défaut :

```
<?php
wp_link_pages( array(
    'before'      => '<p class="post-nav-links">' . __( 'Pages:' ),
    'after'       => '</p>',
    'link_before' => '',
    'link_after'  => '',
    'aria_current' => 'page',
    'next_or_number' => 'number',
    'separator'   => ' ',
    'nextpagelink' => __( 'Next page' ),
    'previouspagelink' => __( 'Previous page' ),
    'pagelink'     => '%',
    'echo'         => 1,
) );
?>
```

Vous pouvez personnaliser le code HTML autour des liens (`before` et `after`), le code HTML autour de chaque lien (`before_link` et `after_link`), la valeur de l'attribut `aria-current`, si vous voulez utiliser des liens de pagination standard ou des boutons “Page suivante” et “Page précédente” avec `next_or_number`, le séparateur des liens, le texte des boutons avec `nextpagelink` et `previouspagelink`, et aussi le texte du lien de page avec `pagelink`. Par exemple, utiliser ‘Page: %’ générera des liens ‘Page 1’, ‘Page 2’, etc. au lieu de juste ‘1’, ‘2’...

Il suffit de placer la fonction dans votre modèle, généralement juste après l'appel à `the_content()`. WordPress gèrera tout seul les liens et leurs URL pour que tout fonctionne. Comme ici dans le fichier `content-page.php` du thème `twentynineteen`:

```
...
<div class="entry-content">
<?php
the_content();

wp_link_pages(
    array(
        'before' => '<div class="page-links">' . __( 'Pages:', 'twentynineteen' ),
        'after'  => '</div>',
    )
);
?>
</div><!-- .entry-content -->
...
```

## `the_posts_navigation()` et `the_posts_pagination()`

Un peu comme `wp_link_pages()`, ces fonctions servent à gérer la pagination, mais elles s'occupent de celle des publications. Plus précisément, elles affichent les liens entre vos pages de publications sur votre blog (ou autres archives). Vous pouvez gérer la navigation entre vos pages en affichant une pagination standard (`the_posts_pagination()`) ou des boutons vers les pages suivantes et précédentes (`the_posts_navigation()`).

Elles prennent un tableau d'arguments et permettent de personnaliser l'affichage des liens, d'afficher ou non les liens vers les pages suivantes et précédentes, etc. On verra tout ça en détails plus loin, ne vous inquiétez pas.

Par exemple, le thème `twentynineteen` choisit d'afficher des boutons avec des icônes sur la page de listing des articles (`index.php`):

```
if ( ! function_exists( 'twentynineteen_the_posts_navigation' ) ) :
/**
 * Documentation for function.
 */
function twentynineteen_the_posts_navigation() {
    the_posts_pagination(
        array(
            'mid_size' => 2,
            'prev_text' => sprintf(
                '%s <span class="nav-prev-text">%s</span>',
                twentynineteen_get_icon_svg( 'chevron_left', 22 ),
                __( 'Newer posts', 'twentynineteen' )
            ),
            'next_text' => sprintf(
                '<span class="nav-next-text">%s</span> %s',
                __( 'Older posts', 'twentynineteen' ),
                twentynineteen_get_icon_svg( 'chevron_right', 22 )
            ),
        )
    );
}
endif;
```

## `the_comments_navigation()`, `the_comments_pagination()`

Ces fonctions s'utilisent exactement comme `the_posts_navigation()` et `the_posts_pagination()`, mais s'occupent de la pagination des commentaires.

À partir du moment où votre thème va devoir afficher un blog, ou des commentaires, donc dans 100% des cas, vous allez avoir besoin de ces fonctions.

Certes vous pouvez réinventer la roue et coder votre propre système de pagination, mais WordPress vous donne tous les outils pour afficher vos liens, gérer les URL et les requêtes pour les pages suivantes, etc. Magique.

## Ce qu'il faut retenir

- Certaines fonctions comme `wp_head()` ou `wp_footer()` sont techniquement indispensables dans votre thème.
- Certaines autres fonctions ne sont pas techniquement indispensables, car on pourrait s'en sortir sans. Mais leur utilisation est attendue par WordPress ou ses extensions. Ne pas les utiliser revient à se fermer des portes.
- Utilisez au maximum les APIs et fonctions de WordPress. De manière générale, il y a (presque) toujours une fonction WordPress ou une API complète pour votre besoin. La règle est donc simple : s'il y a une fonction ou API WordPress qui fait le job, utilisez-la !

# Créer une zone de widgets

Créer une zone de widgets est très simple. Il suffit de déclarer la zone de widgets dans le fichier `functions.php` de votre thème, et ensuite d'appeler votre zone de widgets dans vos modèles de page. Easy.

## Déclarer une zone de widgets

Pour déclarer une zone de widgets dans l'administration WordPress, on utilise la fonction `register_sidebar()`, hookée sur `widgets_init`.

La fonction prend un tableau d'arguments. On va pouvoir lui passer un nom, un identifiant, une description qui apparaîtra dans l'admin, et le balisage que l'on veut voir avant et après chaque widget, ainsi qu'avant et après chaque titre. On va placer notre appel à `register_sidebar()` dans une fonction que l'on va hooker sur `widgets_init` dans notre fichier `functions.php`.

Note : Tout comme on organise une extension en plusieurs fichiers et dossiers pour plus de clarté, on va organiser les fonctionnalités de notre thème de la même façon. Donc même si j'annonce qu'une fonction sera placée dans `functions.php`, on va éviter de mettre tout et n'importe quoi dans ce même fichier et en faire un monstre illisible, ok ? Ranger ses fonctionnalités dans différents fichiers, inclus dans `functions.php`, c'est bien plus propre.

Voici la fonction avec ses paramètres par défaut :

```
add_action( 'widgets_init', 'wpcookbook_sidebars' );
/**
 * Declares a widget area.
 */
function wpcookbook_sidebars(){
    register_sidebar( array(
        'name'          => 'Sidebar $i',
        'id'            => 'sidebar-$i',
        'description'   => '',
        'class'          => '',
        'before_widget' => '<li id="%1$s" class="widget %2$s">',
        'after_widget'  => '</li>\n',
        'before_title'  => '<h2 class="widgettitle">',
        'after_title'   => '</h2>\n'
    ) );
}
```

Par défaut, les zones de widgets sont numérotées automatiquement (d'où le `$i`). Ce qui veut dire que potentiellement, ce numéro peut bouger si vous installez une extension et que celle-ci vient déclarer une zone de widgets avant votre thème. Donc il est fortement recommandé de passer au moins un `name` explicite qui apparaîtra dans l'administration et un `id` unique.

La `description` apparaîtra dans l'administration pour aider les utilisateurs. Vous pouvez y inclure l'endroit où les widgets seront affichés par exemple. Aussi, si vous avez besoin de cibler la zone de

widgets dans l'administration de WordPress, vous pouvez lui passer une class CSS. Attention, cette classe n'apparaît que dans l'administration. Pour cibler la zone de widgets sur le devant du site, utilisez les classes CSS que vous allez introduire dans votre modèle.

`before_widget` et `after_widget` permet de personnaliser le code HTML entourant chaque widget. Par défaut les widgets sont des éléments de liste, avec un identifiant unique (`%1$s`) et deux classes CSS: l'une est `widget` et l'autre (`%2$s`) sera sous la forme `widget_{type}` ou `#{type}` est l'identifiant du widget utilisé (`widget_categories` par exemple).

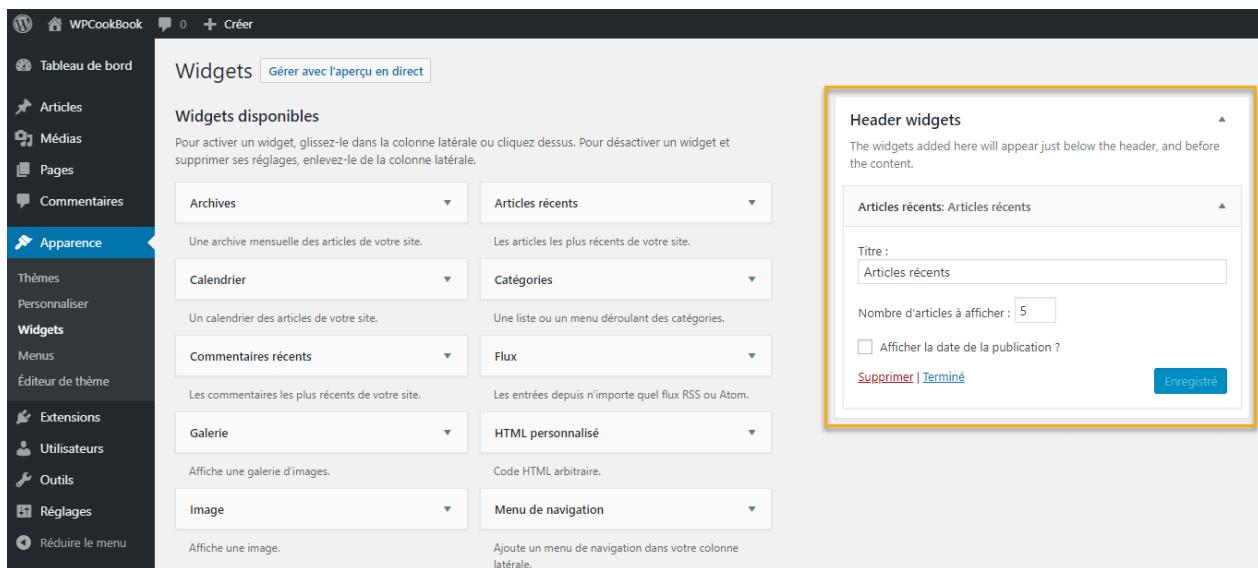
`before_title` et `after_title` permettent de personnaliser le balisage des titres des widgets.

Pour l'exemple, déclarons donc une zone de widgets qui sera placée juste sous l'entête du site, et avant le contenu.

```
function wpcookbook_sidebars(){
    register_sidebar( array(
        'name'          => __( 'Header widgets', 'twentynineteen-child' ),
        'id'            => 'header-widgets',
        'description'   => __( 'The widgets added here will appear just below the header, and before the content.', 'twentynineteen-child' ),
        'before_widget' => '<div id="%1$s" class="widget header-widget %2$s">',
        'after_widget'  => '</div>',
        'before_title'  => '<h2 class="widget-title">',
        'after_title'   => '</h2>'
    ) );
}
```

Maintenant, en allant dans notre administration, on a une zone de widgets qui est déclarée, et on peut y ajouter des widgets ! Notez qu'elle n'apparaît pas encore dans l'outil de personnalisation de WordPress, simplement parce qu'aucun modèle de page ne l'affiche, donc l'outil ne s'embête pas à afficher une option invisible dans la prévisualisation.

J'ai simplement changé les `<li>` par défaut par des `<div>`, ajouté une classe CSS `header-widget` aux widgets et changé la classe `widgettitle` illisible en `widget-title`.



Notre zone de widgets apparaît dans l'admin.

Notre zone de widgets existe. Maintenant, il faut l'afficher dans notre thème. Dupliquez le fichier `header.php` du thème `twentynineteen` dans notre thème enfant. Puis, juste sous la balise fermante `</header>`, ajoutez ceci :

```
<?php endif; ?>
</header><!-- #masthead -->

<?php if( is_active_sidebar( 'header-widgets' ) ) : ?>
    <aside class="header-widgets" role="complementary" aria-label="<?php esc_attr_e('Header widgets', 'twentynineteen-child');?>">
        <?php dynamic_sidebar( 'header-widgets' ); ?>
    </aside>
<?php endif; ?>

<div id="content" class="site-content">
```

La fonction `is_active_sidebar( $id )` prend un identifiant de zone de widgets (numérique ou une chaîne de caractères) et retourne un booléen selon s'il y a bien des widgets enregistrés dans cette zone ou non.

Si oui, alors on peut afficher nos widgets avec `dynamic_sidebar( $id )`. Comme `is_active_sidebar()`, la fonction prend un identifiant et affiche les widgets enregistrés dans cette zone. Je vous ai déjà dit que WordPress était magique ?

Les widgets sont dans un conteneur `<aside>`. Pour améliorer l'accessibilité du thème, on va inclure un attribut `role` et un `aria-label`. On a déjà parlé de la fonction `esc_attr_e()`. Pour rappel, c'est une combinaison des fonctions `_e()` qui affiche une chaîne traduisible, et `esc_attr()` qui va échapper une chaîne pour l'afficher sans risque dans un attribut HTML.

The widgets added here will appear just below the header, and before the content.

**Articles récents:** Articles récents

**Réorganiser** **Ajouter un Widget**

**Archives**  
Une archive mensuelle des articles de votre site.

**Articles récents**  
Les articles les plus récents de votre site.

**Calendrier**  
Un calendrier des articles de votre site.

**Catégories**  
Une liste ou un menu déroulant des catégories.

**Commentaires récents**  
Les commentaires les plus récents de votre site.

**Flux**  
Les entrées depuis n'importe quel flux RSS ou Atom.

**Galerie**  
Affiche une galerie d'images.

**WPCookBook** **Apprenez à développer pour Wo**  
**Page Article Catégorie Contact**

# Articles récents

Hello world!

# Hello world!

Welcome to WordPress. This is your first post. Edit or del

Notre zone de widgets apparaît dans l'outil de personnalisation et sur le devant du site.

Par contre, on aura un peu de style à faire. Un peu de magie CSS, et bim, le tour est joué. En ajoutant ces quelques lignes dans `style.css` de votre thème enfant, ça va déjà mieux.

```
*****
* Header widget area
*****
.header-widgets {
    display: flex;
    flex-wrap: wrap;
    margin: 0 calc(10% + 60px);
}

.header-widget {
    margin-right: 3em;
}
```

## Articles récents

[Hello world!](#)

## Catégories

[Uncategorized](#)

## Commentaires récents

[A WordPress Commenter dans Hello world!](#)



# Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

vincent septembre 12, 2019 Uncategorized Un commentaire Modifier

Notre zone de widgets avec quelques styles.

## sidebar.php

Notre zone de widgets est fonctionnelle, mais elle mérite mieux. Elle mérite son propre fichier de template.

Créez un fichier nommé `sidebar-header.php` à la racine du thème enfant, et coupez-collez y le code utilisé dans `header.php` pour afficher notre zone de widgets. Puis, on va simplement remplacer notre code dans `header.php` par :

```
<?php endif; ?>
</header><!-- #masthead -->

<?php get_sidebar( 'header' ); ?>

<div id="content" class="site-content">
```

La fonction `get_sidebar( $name )` prend un identifiant de zone de widgets en paramètre, et tente d'aller chercher un fichier `sidebar-$name.php` à la racine de votre thème. S'il ne le trouve pas, il va chercher `sidebar.php`.

On a vu au chapitre Comprendre son thème que l'on pouvait appeler différents entêtes ou pieds de site à l'aide des fonctions `get_header()` et `get_footer()`. Ici, `get_sidebar()` fonctionne exactement de la même façon. On peut donc inclure autant de fichiers `sidebar-$name.php` que l'on a besoin dans notre thème.

## Ce qu'il faut retenir

- On déclare une zone de widgets avec `register_sidebar()` hookée sur `widgets_init`.
- On vérifie qu'une zone de widgets contient des widgets avec `is_active_sidebar()`.
- On affiche les widgets d'une zone de widgets avec `dynamic_sidebar()`.
- On appelle un template de zone de widgets avec `get_sidebar()`.
- On peut créer autant de zones de widgets que nécessaires et autant de fichiers templates que nécessaires.

Et c'est tout ! Par contre, n'oubliez pas de travailler vos modèles dans votre thème enfant !

# Comment déclarer une zone de menu dans votre thème

Parmi les besoins simples et indispensables, il y a les menus. C'est assez rare qu'on ait d'emblée tous les menus dont on a besoin dans notre thème. Donc, pas de secret, il faut ajouter un petit bout de code. Voilà comment faire.

Comme pour les autres chapitres, on va travailler dans notre thème enfant de `twentynineteen` que l'on a créé au début du guide. Si vous vous êtes précipités jusqu'à cette section sans lire le début, je vous renvoie au chapitre [Comment créer un thème enfant](#).

## Comment enregistrer notre menu dans l'administration

Imaginons que nous voulions ajouter une Hello Bar tout en haut de notre thème `twentynineteen`, avec des liens vers les pages "Mon compte", "Panier", ou autre.

Avant d'afficher un menu, il faut déclarer un emplacement de menu dans l'administration. Comme beaucoup de fonctionnalités des thèmes, il faut le faire en se hookant sur `after_setup_theme`. On peut utiliser les fonctions `register_nav_menu()` ou `register_nav_menus()` (au pluriel).

Dans `functions.php`, essayez :

```
add_action( 'after_setup_theme', 'twentynineteen_child_setup' );
/**
 * Setup all of our theme's functionnalities
 */
function twentynineteen_child_setup(){
    register_nav_menu( 'top-bar-menu', __( 'Top bar menu', 'twentynineteen-child' ) );
}
```

`register_nav_menu( string $location, string $description )` prend deux paramètres. `$location` est simplement le slug (l'identifiant) de votre menu, et `$description` est une chaîne traduisible qui va être utilisée dans l'administration. On lui passe donc un identifiant clair et lisible et une description simple.

La fonction `register_nav_menus()` fonctionne exactement de la même façon, sauf qu'elle prend un tableau associatif de menus. Elle permet simplement d'enregistrer plusieurs menus avec une seule fonction. On aurait donc pu écrire notre fonction ainsi :

```
function twentynineteen_child_setup(){
    register_nav_menus( array(
        'top-bar-menu' => __( 'Top bar menu', 'twentynineteen-child' ),
    ) );
}
```

C'est comme vous voulez, selon si vous avez besoin d'ajouter un ou plusieurs menus.

The screenshot shows the WordPress 'Menus' screen. On the left, there's a sidebar with various menu items like Tableau de bord, Articles, Médias, Pages, Commentaires, Apparence, Thèmes, Personnaliser, Widgets, Menus, Éditeur de thème, Extensions, Utilisateurs, Outils, Réglages, and Réduire le menu. The 'Menus' item is selected.

The main area has two tabs: 'Modifier les menus' (Edit menus) and 'Gérer les emplacements' (Manage locations). Below that, it says 'Sélectionnez le menu à modifier : Menu (Principal) ▾ Sélectionner ou créez un nouveau menu. N'oubliez pas d'enregistrer vos modifications !' (Select the menu to modify: Main menu ▾ Select or create a new menu. Don't forget to save your changes!).

The 'Ajouter des éléments de menu' (Add menu items) panel on the left lists 'Pages' (Pages), 'Articles', 'Liens personnalisés', and 'Catégories'. Under 'Pages', there's a list of items like Testimonial form, Shortcode, Roles, Custom Table, Navigating, Contact, Sample Page, and Page B. There's also a 'Tout sélectionner' (Select all) checkbox and an 'Ajouter au menu' (Add to menu) button.

The 'Structure du menu' (Menu structure) panel on the right shows the menu hierarchy. It includes fields for 'Nom du menu' (Name of the menu) and 'Enregistrer le menu' (Save menu). Below that, it says 'Glissez chaque élément pour les placer dans l'ordre que vous préférez. Cliquez sur la flèche à droite de l'élément pour afficher d'autres options de configuration.' (Drag each element to place it in the order you prefer. Click on the arrow to the right of the element to display other configuration options.). It lists 'Page', 'Article', 'Catégorie', and 'Contact' with dropdown menus for 'Page', 'Article', 'Catégorie', and 'Page' respectively.

The 'Réglages du menu' (Menu settings) panel at the bottom has sections for 'Ajoutez automatiquement les pages de premier niveau à ce menu' (Automatically add first-level pages to this menu) and 'Afficher l'emplacement' (Display location). Under 'Afficher l'emplacement', there are checkboxes for 'Top bar menu', 'Principal' (which is checked), 'Menu du pied de page', and 'Menu des liens de réseaux sociaux'. A yellow arrow points to the 'Principal' checkbox.

At the bottom of the screen, there's a message 'Merci de faire de WordPress votre outil de création.' (Thank you for making WordPress your creation tool.) and 'Version 5.3'.

Notre emplacement de menu est bien présent.

Note : On aurait aussi pu se hooker sur `init`. Mais en général, toutes les fonctionnalités du thème sont hookées un poil plus tôt, sur `after_setup_theme`, c'est-à-dire juste après que le thème soit chargé. La fonction `twentynineteen_child_setup()` hookée sur `after_setup_theme` pourra donc servir pour déclarer plusieurs fonctionnalités, comme le support pour le logo, l'entête personnalisée, etc.

## Afficher notre menu

Afficher notre menu est presque aussi simple, il suffit de dupliquer le fichier `header.php` du thème parent dans le thème enfant si ce n'est pas déjà fait, et d'appeler la fonction `wp_nav_menu()` dans ce fichier, avec un peu de balisage si besoin.

Dans header.php, ajoutez un élément <nav> juste après le skip-link.

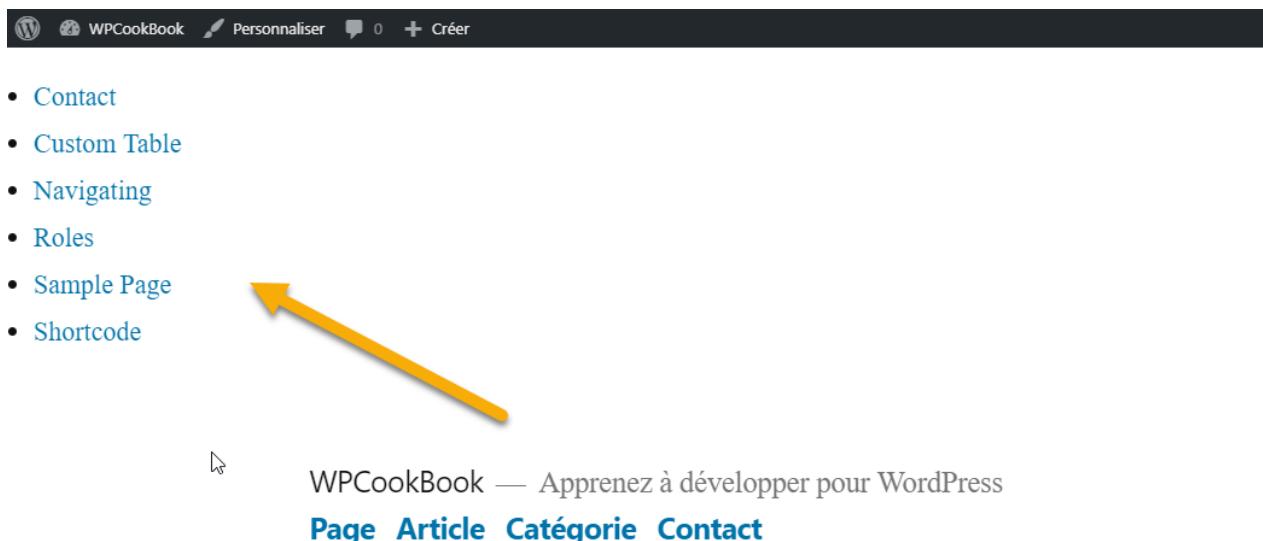
```
...
<div id="page" class="site">
    <a class="skip-link screen-reader-text" href="#content"><?php _e( 'Skip to content',
'twentynineteen' ); ?></a>

    <nav class="top-bar-navigation" aria-label="<?php esc_attr_e( 'Top Bar Menu',
'twentynineteen-child' ); ?>">
        <?php wp_nav_menu( array( 'theme_location' => 'top-bar-menu' ) ); ?>
    </nav><!-- #site-navigation -->

    <header id="masthead" class="<?php echo is_singular() &&
twentynineteen_can_show_post_thumbnail() ? 'site-header featured-image' : 'site-header'; ?>">
    ...

```

En allant sur le devant du site, vous aurez une surprise !



Notre menu s'affiche. Non, pardon. UN menu s'affiche.

Par défaut, WordPress génère le menu des pages du site ou le premier menu non-vide si aucun menu personnalisé n'est affecté à l'emplacement de menu en question !

C'est important et c'est quelque chose qu'il faut prendre en compte si vous publiez un thème sur le répertoire officiel, parce que par défaut, `wp_nav_menu()` utilisera `wp_page_menu()` en lui passant les mêmes arguments s'il ne trouve pas de menu à afficher. Vos styles devront gérer les deux cas, et selon vos paramètres, le balisage produit pourra être différent.

Vous allez me dire, "Mais qui ne personnalise pas ses menus ?". Je suis assez d'accord. Mais bon. C'est le cas par défaut, et il faut le gérer.

En attendant, si aucun menu n'est affecté à cet emplacement, autant ne rien afficher du tout. Pour ce faire, il suffit d'utiliser `has_nav_menu()` pour vérifier qu'un menu est affecté à l'emplacement passé en paramètre, simplement.

```
<?php if ( has_nav_menu( 'top-bar-menu' ) ) : ?>
    <nav class="top-bar-navigation" aria-label="<?php esc_attr_e( 'Top bar Menu',
'twenty nineteen-child' ); ?>">
        <?php wp_nav_menu( array( 'theme_location' => 'top-bar-menu' ) ); ?>
    </nav><!-- #site-navigation -->
<?php endif; ?>
```

Maintenant, rien ne s'affiche si aucun menu n'est affecté à notre top bar.

## wp\_nav\_menu()

La fonction permettant d'afficher les menus est très souple et prend un tableau d'arguments en paramètres. Voici les paramètres par défaut :

```
wp_nav_menu( array(
    'menu'          => '',
    'container'     => 'div',
    'container_class' => '',
    'container_id'   => '',
    'menu_class'     => 'menu',
    'menu_id'        => '',
    'echo'           => true,
    'fallback_cb'    => 'wp_page_menu',
    'before'         => '',
    'after'          => '',
    'link_before'    => '',
    'link_after'     => '',
    'items_wrap'      => '<ul id="%1$s" class="%2$s">%3$s</ul>',
    'item_spacing'    => 'preserve',
    'depth'          => 0,
    'walker'         => '',
    'theme_location' => '',
) );
```

C'est parti.

- `menu` est le nom du menu à afficher. C'est-à-dire qu'au lieu d'afficher le menu affecté à un emplacement, vous pouvez forcer l'affichage d'un menu spécifique en lui passant un identifiant, un slug ou encore une instance de `WP_Term`. Ce menu doit exister préalablement. Sinon, on aura le menu des pages ou le premier menu non-vide.
- `container` permet de spécifier la balise HTML souhaitée autour de notre menu. Il n'accepte que deux valeurs : `'div'` ou `'ul'`. Par défaut, notre menu est une `<ul>` dans une `<div>`, mais si on passe `'ul'` ici, notre `<div>` disparaît. Remarquez qu'elle disparaît aussi si on passe `false`. Donc trois valeurs possibles, en fait.
- `container_class` est une classe CSS que l'on va appliquer au conteneur `<div>`. Si vous avez passé

“ul” au paramètre précédent, alors la classe est ignorée.

- `container_id` est l’identifiant du conteneur `<div>`. Comme pour `container_class`, ce paramètre est ignoré si vous choisissez de ne pas utiliser de conteneur `<div>`.
- `menu_class` permet d’ajouter des classes CSS à la balise `<ul>` du menu.
- `menu_id` permet d’ajouter un identifiant sur la balise `<ul>` du menu.
- `echo` détermine si la fonction doit afficher ou retourner le menu.
- `fallback_cb` est la fonction de rappel à utiliser si le menu ou l’emplacement n’est pas trouvé. Par défaut, la fonction utilisée est `wp_page_menu()`, qui affiche le menu des pages. C’est le souci que nous avons rencontré juste avant. C’est ce paramètre le coupable. On peut lui passer une fonction personnalisée pour afficher un menu par défaut, ou `false` pour ne rien faire du tout.
- `before` et `after` permettent de modifier le balisage autour du lien de chaque item. C’est-à-dire qu’on peut modifier le code HTML autour de la balise `<a>`, mais tout en restant à l’intérieur de la balise `<li>`. Par défaut, ces paramètres sont vides, et les éléments de menu sont de simples `<a>` dans des `<li>`.
- `link_before` et `link_after` permettent de personnaliser le code HTML à l’intérieur de la balise `<a>` et d’ajouter un conteneur `<span>` supplémentaire à notre ancre, par exemple.
- `items_wrap` est le balisage HTML qui va entourer nos éléments. Par défaut c’est une `<ul>`, et c’est très bien comme ça. Les `%1$s`, `%2$s` et `%3$s` représentent respectivement les attributs `id`, `class`, et les éléments de menu eux-mêmes.
- `item_spacing` permet de préserver ou non les espaces dans le code HTML des éléments. Par défaut, le paramètre vaut “`preserve`” mais on peut lui passer “`discard`”.
- `depth` est la profondeur du menu. C’est-à-dire est-ce qu’on affiche uniquement les éléments de premier niveau (les parents) ou aussi les enfants ? Et jusqu’à quel niveau ? Par défaut, le paramètre vaut 0, ce qui affiche tous les niveaux.
- `walker` est l’instance de classe `WP_Walker` que l’on veut utiliser pour le menu. What ? Pour faire simple, un walker est une classe que WordPress utilise pour afficher des éléments récursivement. Les menus WordPress peuvent avoir des sous-menus, donc ils peuvent avoir des imbriquations de `<ul>` (sous-menus) dans des `<li>` (éléments du menu parent). Les menus et les commentaires sont deux exemples pour lesquels WordPress utilise des walkers pour générer le balisage. C’est un sujet complexe et on ne va pas en parler ici. Ou du moins, pas dans version 1 de ce guide . On va juste laisser la chaîne vide par défaut pour le moment.
- `theme_location` est l’emplacement de menu où il faut chercher le menu à afficher.

Je vous avais dit qu’il y avait beaucoup de paramètres !

Si on revient à notre appel de base, on indique uniquement dans quel emplacement de menu chercher. Mais on peut personnaliser un peu notre appel à la fonction.

```

<nav class="top-bar-navigation" aria-label="<?php esc_attr_e( 'Top bar Menu', 'twentynineteen-child' ); ?>">
    <?php // wp_nav_menu( array( 'theme_location' => 'top-bar-menu' ) ); ?>
    <?php
        wp_nav_menu( array(
            'container'      => 'ul',
            'menu_class'     => 'menu top-bar-menu',
            'menu_id'        => 'top-bar-menu',
            'link_before'    => '<span class="top-bar-link-text">',
            'link_after'     => '</span>',
            'theme_location' => 'top-bar-menu',
        ) );
    ?>
</nav><!-- #top-bar-navigation -->

```

Je supprime le conteneur `<div>` en passant `ul` pour contenir, j'ajoute mes classes CSS et mon identifiant personnalisé, et j'entoure mes liens d'un `<span>` avec une classe CSS.

Cela me donne un balisage comme ceci :

```

▼<div id="page" class="site">
  <a class="skip-link screen-reader-text" href="#content">Aller au contenu</a>
  ▼<nav class="top-bar-navigation" aria-label="Top bar Menu">
    ▼<ul id="top-bar-menu" class="menu top-bar-menu">
      ▼<li id="menu-item-66" class="menu-item menu-item-type-post_type menu-item-object-page menu-item-66">
        ▼<a href="https://wpcookbook.local/contact/">
          ...
          <span class="top-bar-link-text">Mon compte</span> == $0
        </a>
      </li>
      ▶<li id="menu-item-67" class="menu-item menu-item-type-post_type menu-item-object-page menu-item-67">...</li>
    </ul>
  </nav>
  <!-- #top-bar-navigation -->

```

Un balisage simple et propre.

Ajouter un `<span>` autour du texte des liens n'est pas forcément nécessaire, cela dépend de ce dont vous avez besoin en CSS. C'est juste pour l'exemple.

## Notre menu est en place

Tout fonctionne correctement, il reste maintenant à styler tout ça. Vous remarquerez que WordPress ajoute automatiquement un identifiant et plusieurs classes CSS aux éléments de menu, y compris une classe `current-menu-item` et `current_page_item` pour l'élément actif. Par défaut, il y a donc tout ce dont nous avons besoin pour styler notre menu.

```

*****
* Top bar menu
*****/
.top-bar-navigation {
    background-color: #111;
    color: white;
    padding: 0 1em;
}

.top-bar-menu {
    color: white;
    display: flex;
    font-size: 14px;
    justify-content: flex-end;
    letter-spacing: .5px;
    list-style: none;
    margin: 0;
    text-transform: uppercase;
}

.top-bar-menu li {
    margin-left: 10px;
}

.top-bar-menu a {
    transition: all .25s ease;
}

.top-bar-menu a,
.top-bar-menu a:visited {
    color: white;
    opacity: .8;
}

.top-bar-menu a:hover,
.top-bar-menu a:focus {
    opacity: 1;
    text-decoration: underline;
}

```



WPCookBook — Apprenez à développer pour WordPress  
[Page](#) [Article](#) [Catégorie](#) [Contact](#)



La top bar est en place.

Et hop, une petite top bar bien utile !

Bon, ici on ne gère pas le cas des sous-menus. On pourrait aussi simplement ne pas le gérer ! Les sous-menus dans une top bar, ça devient compliqué. On peut donc passer depth à 1 dans notre appel à `wp_nav_menu()` et on ignorera ainsi tout sous-menu déclaré. Par contre, pensez à bien le documenter dans le `readme.txt` de votre thème, si vous le publiez sur le répertoire officiel.

## À retenir

- On déclare un emplacement de menu à l'aide de `register_nav_menu()`, ou `register_nav_menus()`, hooké sur `after_setup_theme`.
- On vérifie qu'un emplacement dispose d'un menu avec `has_nav_menu()` en lui passant un slug d'emplacement de menu.
- On affiche un menu avec `wp_nav_menu()`. On peut passer un tableau de paramètres pour personnaliser le conteneur, les classes CSS, le balisage autour des `<a>` et autour du texte des `<a>`, la profondeur, etc.
- Le paramètre le plus important est `theme_location` qui doit correspondre à l'identifiant que vous avez déclaré avec `register_nav_menu()`.
- Il est recommandé de gérer l'absence de menu et l'affichage du menu des pages par défaut, dans le cas où vous souhaitez publier le thème sur le répertoire officiel. De la même façon, si le menu ne gère pas l'affichage des sous-menus, il faut le spécifier.

La fonction `wp_nav_menu()` est simple à utiliser et donne déjà pas mal de flexibilité. Pour personnaliser encore plus loin les menus (ajouter des petites icônes devant vos éléments par exemple), rendez-vous au chapitre [Comment personnaliser les menus de votre thème WordPress](#) ! Maintenant, vous êtes armé pour ajouter des menus n'importe où sur vos sites !

# Comprendre la boucle de WordPress

Vous êtes prêts ? Parce que là, c'est du lourd.

La boucle de WordPress, c'est une mécanique simple mais puissante qui est responsable de l'affichage du bon contenu sur les bonnes pages. Si vous avez lu le chapitre Comprendre son thème, vous avez normalement un idée de ce qu'est la boucle.

Dans ce chapitre, on va aller un peu plus loin et analyser plus en détails les mécaniques derrière les fonctions `have_posts()` et `the_post()`. Mais la chose la plus importante dont on va parler, c'est comment vous pouvez intervenir lors de ce processus.

Ca va être un gros chapitre, mais si vous comprenez la boucle de WordPress (au sens large), alors vous aurez le maximum de contrôle sur votre contenu. Et ça, c'est cool.

## Comment fonctionne la boucle standard

La boucle de WordPress est simplement une boucle `while` qui utilise les fonctions `have_post()` et `the_post()` pour itérer sur chaque élément de contenu et le préparer à l'affichage. Voici un exemple simple de boucle typique.

```
if( have_posts() ) {
    while ( have_posts() ){
        the_post();
        get_template_part( 'template-parts/content' );
    }
} else {
    get_template_part( 'template-parts/content', 'none' );
}
```

Prenons l'exemple de la page de votre blog, qui est gouvernée par `index.php` ou `home.php` dans votre thème.

La fonction `have_posts()` va vérifier s'il y a des articles disponibles. Plus précisément, la fonction vérifie si on est en bout de boucle (s'il y a un article suivant). Donc s'il n'y a aucun article à afficher, on est en bout de boucle.

S'il n'y a pas d'articles, on affiche un template partiel `template-parts/content-none.php` qui contient, j'espère, un message utile. S'il y a un article suivant, alors `the_post()` va le chercher et le préparer à l'affichage.

C'est tout simple, mais ce qu'il faut bien comprendre, c'est qu'à ce moment de l'exécution, c'est-à-dire quand WordPress lit le code de votre modèle de page, la requête en base pour le contenu est déjà faite et toutes les données concernant la requête courante sont dans la globale `$wp_query`.

Insérez ce bout de code dans votre modèle `index.php`, juste pour tester :

```
global $wp_query;
echo '<pre>';
var_dump($wp_query);
echo '</pre>';
```

La globale contient tous les paramètres utilisés et d'autres propriétés super utiles qui permettent de déterminer dans quel contexte se situe la boucle (page simple, article simple, liste des articles, etc.). Vous pouvez aussi voir que l'on a un tableau `posts` qui contient notre contenu.

`have_posts()` ne fait que tenir compte de l'endroit où l'on se situe dans la boucle. `the_post()` quant à elle, avance d'un article dans la boucle et prépare la globale `$post` qui contient les données de l'article courant. Tous les template tags de WordPress, comme `the_title()` et `the_content()`, vont chercher leurs données dans la globale `$post` par défaut.

## Comment personnaliser la boucle

Si à ce stade, la requête de contenu est déjà faite, comment faire pour intervenir et personnaliser la requête ? Il faut venir se greffer avant, mais quand ?

On a déjà parlé brièvement du chargement de WordPress, mais ça vaut le coup d'y revenir un peu ici. Si vous relisez rapidement le tableau des hooks principaux rencontrés, on peut résumer le chargement comme suit :

- WordPress charge ses fonctions, les extensions et le thème (jusque le hook `init`),
- Il traite les paramètres de l'URL demandée (`hookparse_request`),
- Il prépare `$wp_query` et fait la requête qui va bien,
- Il choisit le modèle de page et l'affiche.

Il faut donc intervenir juste après que WordPress ait interprété l'URL du contenu demandé, mais avant qu'il n'aille effectivement chercher le contenu.

Il utilise la méthode `get_posts()` de la globale `$wp_query` pour aller chercher le contenu et cette méthode expose un hook bien pratique au tout début : `pre_get_posts`.

## Comment se greffer sur `pre_get_posts`

```
do_action_ref_array( 'pre_get_posts', array( &$this ) );
```

Les fonctions de rappel hookées sur `pre_get_posts` reçoivent en paramètre `$this`, qui correspond à l'instance complète de `WP_Query`. Attention, elle est passée par référence ! Ce qui veut dire que ce n'est pas une copie de la variable qui est passée et que l'on doit retourner, mais la variable elle-même. Donc tout changement effectué affecte directement notre instance.

On a accès à l'instance de `WP_Query` complète, donc on peut modifier ses variables directement. Dans `functions.php`, ajoutez :

```
add_action( 'pre_get_posts', 'wpcookbook_pre_get_posts', 10, 1 );
/**
 * Modifies the query on main blog page.
 *
 * @param $query Query object
 */
function wpcookbook_pre_get_posts( $query ){
    $query->set( 'order', 'ASC' );
}
```

Ici, on va simplement se hooker sur `pre_get_posts`, et fixer la valeur de la variable de requête `order` sur `ASC`, pour classer nos articles par date croissante au lieu de décroissante par défaut.

Si vous faites un tour sur le devant du site, vous pouvez voir que nos articles sont classés du plus ancien au plus récent, mais c'est aussi le cas pour notre widget “Articles récents” !

---

## Articles récents

**Hello world!** septembre 12, 2019

**Hello Universe !** novembre 1, 2019

**Hello Galaxy !** novembre 1, 2019

Notre fonction affecte notre widget!

### Cibler nos modifications

Le hook `pre_get_posts` est déclenché à chaque fois qu'une requête est faite. Donc pour notre page de blog, il est déclenché une fois pour notre contenu principal, mais aussi une fois pour le contenu de notre widget.

Affecter toutes les requêtes est rarement désiré, donc il faut cibler plus précisément le contexte dans lequel on veut intervenir. On ne veut pas modifier la requête utilisée pour le widget, donc on va ajuster notre code ainsi :

```

function wpcookbook_pre_get_posts( $query ){
    if( $query->is_main_query() ){
        $query->set( 'order', 'ASC' );
    }
}

```

La méthode `is_main_query()` vérifie si on est bien dans la boucle principale. Nos modifications ne vont donc pas affecter la requête pour notre widget. Par contre, elle affecte celle dans l'administration !

| Titre            | Auteur  | Catégories    | Étiquettes | Date              |
|------------------|---------|---------------|------------|-------------------|
| Hello world!     | vincent | Uncategorized | —          | Publié 12/09/2019 |
| Hello Universe ! | vincent | Uncategorized | —          | Publié 01/11/2019 |
| Hello Galaxy !   | vincent | Uncategorized | —          | Publié 01/11/2019 |
| Titre            | Auteur  | Catégories    | Étiquettes | Date              |

Nos modifications affectent aussi l'administration.

Donc, on va encore ajuster notre fonction :

```

function wpcookbook_pre_get_posts( $query ){
    if( ! is_admin() && $query->is_main_query() ){
        $query->set( 'order', 'ASC' );
    }
}

```

La fonction `is_admin()` vérifie si l'on est dans l'administration de WordPress, simplement. Attention, elle ne vérifie pas si l'utilisateur est un administrateur ! Pour ça, rendez-vous au chapitre Comprendre les rôles et capacités de WordPress.

Il faut être extrêmement prudent pour ne pas affecter plusieurs requêtes et créer des effets indésirables. Pour nous aider, il existe de nombreuses fonctions qui permettent de vérifier sur quel type de page on est (si l'on veut affecter la requête principale) : `is_single()`, `is_singular()`, `is_page()`, `is_home()`, `is_category()`, `is_post_type_archive()`, etc. On peut aussi vérifier directement les propriétés de `$query` ou utiliser ses méthodes : `$query->is_home()`, `$query->is_single()`, etc.

La différence entre les fonctions `is_single()`, `is_home()`, etc. et les méthodes `$query->is_single()`,

`$query->is_home()` etc. est que les fonctions regardent toujours les paramètres de la globale `$wp_query`, donc de la requête par défaut exécutée pour la page.

Vous pouvez créer vos propres requêtes supplémentaires (voir plus loin), mais dans ce cas, on va créer une autre instance de `WP_Query`. Donc il faudra utiliser ses méthodes.

## Autres exemples

Pour connaitre l'ensemble des paramètres de requête que l'on peut appliquer, je vous renvoie à la documentation de `WP_Query`: [https://developer.wordpress.org/reference/classes/wp\\_query/](https://developer.wordpress.org/reference/classes/wp_query/).

Vous aurez une vue d'ensemble des paramètres, ainsi que plein d'exemples plus précis. Pour le sport, je vais vous donner d'autres exemples.

### Exclure une catégorie, uniquement sur la page listant les articles

```
function wpcookbook_pre_get_posts( $query ){
    if( ! is_admin() && $query->is_main_query() && $query->is_home() ){
        $query->set( 'category__not_in', array( '1' ) );
    }
}
```

Ici, on exclut la catégorie ayant pour identifiant 1 de la page du blog. Par contre, la page d'archive de la catégorie liste les articles normalement. `is_home()` permet de vérifier que l'on est bien sur le blog.

### Ajouter des types de contenus personnalisés aux résultats de recherche

```
function wpcookbook_pre_get_posts( $query ){
    if( ! is_admin() && $query->is_main_query() && $query->is_search() ){
        $query->set( 'post_type', array( 'post', 'page', 'my-post-type' ) );
    }
}
```

Ici, on utilise simplement `is_search()` pour vérifier que la requête concerne bien des résultats de recherche, et on dit à WordPress de chercher dans les pages, articles et publications du type 'my-post-type'.

### Ne chercher que les articles ayant une certaine métadonnée

```
function wpcookbook_pre_get_posts( $query ){
    if( ! is_admin() && $query->is_main_query() ){
        $query->set( 'meta_key', 'my_custom_field' );
        $query->set( 'meta_value', 'display' );
    }
}
```

Ici, on dit à WordPress de ne récupérer que les articles ayant un champ personnalisé avec pour clé `my_custom_field` ET pour valeur `display`. Normalement, tous nos articles devraient disparaître de la page d'accueil du blog. Il faut leur ajouter ce champ personnalisé pour les afficher à nouveau.

Si vous utilisez le nouvel éditeur de WordPress, il faut cliquer sur les réglages de l'éditeur, et dans les Options cocher la case Champs personnalisés pour faire apparaître l'interface par défaut de WordPress. En général, on évite de modifier les champs personnalisés via cette interface, mais cela suffit pour l'exemple.

## Ne chercher que les articles ayant une certaine combinaison de métadonnées

Ici, c'est un peu plus complexe, car les simples paramètres `meta_key` et `meta_value` ne suffisent plus. Il faut ajouter une `meta_query`.

```
function wpcookbook_pre_get_posts( $query ){
    if( ! is_admin() && $query->is_main_query() ){
        $query->set( 'meta_query', array(
            'relation' => 'AND',
            array(
                'key'    => 'my_custom_field',
                'value'  => 'display',
            ),
            array(
                'key'    => 'my_second_custom_field',
                'value'  => 'another_value',
            ),
        ) );
    }
}
```

Ici, on demande les articles ayant les deux champs personnalisés, avec les deux valeurs correspondantes. Notez que '`meta_query`' est un tableau contenant une entrée '`relation`' qui vaut '`AND`' par défaut, et des tableaux d'arguments.

Encore une fois, si on n'ajoute pas les bons champs et valeurs, nos articles n'apparaissent pas sur le blog.

Ces exemples sont très basiques et il est possible d'aller bien plus loin avec les `meta_query`. Elles sont très puissantes et permettent une logique complexe et donc vraiment un contrôle précis sur le contenu à aller chercher.

## Ne chercher que les articles d'une certaine catégorie ayant aussi un certaine étiquette

De la même façon que l'on peut composer des `meta_query` très complexes, on peut faire de même avec les taxonomies.

```

function wpcookbook_pre_get_posts( $query ){
    if( ! is_admin() && $query->is_main_query() ){
        $query->set( 'tax_query' , array(
            'relation' => 'AND',
            array(
                'taxonomy' => 'category',
                'field'      => 'slug',
                'terms'      => array( 'wordpress' ),
            ),
            array(
                'taxonomy' => 'post_tag',
                'field'      => 'slug',
                'terms'      => array( 'snippet' ),
            ),
        ) );
    }
}

```

Ici, on va chercher uniquement les articles de la catégorie ayant pour slug `wordpress`, et ayant aussi `snippet` dans ses étiquettes. Vous pouvez utiliser n'importe quelle taxonomie pré-existante ou déclarée par une extension. Comme pour les `meta_query`, les `tax_query` permettent une logique complexe et donc d'avoir un maximum de contrôle.

On n'a fait que survoler les possibilités de `WP_Query`. Vous pouvez personnaliser toutes les requêtes de votre site. Widgets, pages, produits, résultats de recherche, archives, produits Woocommerce, tout type de contenu personnalisé, etc. Vous pouvez même faire des trucs qui n'ont aucun sens, comme afficher à chaque utilisateur uniquement la liste de ses propres articles sur votre blog. Comme ça, il n'y a que lui qui peut lire, et ce, uniquement quand il est connecté. Débile. Sauf si on fait un journal intime avec WordPress. (Hey ? Une idée d'extension ? Avec des autorisations données à certains utilisateurs et tout ?)

`pre_get_posts` et `WP_Query` sont des outils essentiels qu'il vous faut maîtriser, donc allez-y, amusez-vous et détruisez votre site. Local, bien entendu !

## Comment créer de nouvelles boucles

On peut aussi créer des boucles alternatives pour afficher du contenu supplémentaire sur votre site. Beaucoup de widgets le font, comme le widget natif “Articles récents”. Il utilise une `WP_Query` personnalisée pour aller chercher les articles, simplement.

Pour s'entraîner, on va donc essayer d'ajouter en bas de chacun de nos articles une petit bloc présentant les trois derniers articles de la même catégorie. Une sorte de bloc “Articles similaires”. On va faire ça dans une petite extension.

Pour ajouter ce bloc à notre contenu, on va se hooker sur `the_content`.

```

add_filter( 'the_content', 'wpcookbook_related_posts', 10, 1 );
/**
 * Adds a related posts block to the content
 */
function wpcookbook_related_posts( $content ){
    if( is_single() ){
        $content .= '<div class="wpcookbook-related-posts">RELATED POSTS HERE</div>';
    }
    return $content;
}

```

Puisque l'on veut afficher notre bloc uniquement en bas des articles, on utilise la fonction `is_single()` pour vérifier qu'on est bien sur une page d'article simple, et on ajoute juste une `<div>` pour l'instant pour vérifier que l'on est bien au bon endroit.

The screenshot shows a WordPress dashboard with a dark header bar containing the WordPress logo, 'WPCookBook', 'Personnaliser', 'Créer', and 'Modifier l'article'. Below the header, there are navigation links: 'Page', 'Article', 'Catégorie', and 'Contact'. The main content area displays a post titled 'Hello world!'. Below the title, the author is listed as 'vincent' on 'septembre 12, 2019', with 'Un commentaire' and a 'Modifier' link. The post content is a single line: 'Welcome to WordPress. This is your first post. Edit or delete it, then start writing!'. Underneath the post, there is a section titled 'RELATED POSTS HERE' with a small downward arrow icon. Below this section, another set of post details is shown: 'vincent' on 'septembre 12, 2019', 'Uncategorized', and a 'Modifier' link. At the bottom of the page, there is a link to the next article: 'Article suivant— Hello Universe !'. A note at the bottom right states: 'Notre bloc s'affiche uniquement sur les articles simples.'

## Créons une boucle

Maintenant, on va créer une boucle pour chercher et afficher nos articles. On sait qu'on veut afficher des articles des mêmes catégories que l'article principal, donc si on cherche un peu dans la documentation de la classe `WP_Query`, on trouve les paramètres qui nous intéressent :

[https://developer.wordpress.org/reference/classes/wp\\_query/#category-parameters](https://developer.wordpress.org/reference/classes/wp_query/#category-parameters).

On va donc avoir besoin du paramètre `category__in`, très probablement.

On peut ajouter le squelette de notre boucle :

```
function wpcookbook_related_posts( $content ){
    if( is_single() ){

        $category_ids = array();

        $q = new WP_query( array(
            'posts_per_page' => 3,
            'category__in'   => $category_ids,
        ) );

        if( $q->have_posts() ){
            ob_start();
            ?>
            <div class="wpcookbook-related-posts">
                <h3><?php esc_html_e( 'Related posts', '15-loops' ) ?></h3>
                <?php while( $q->have_posts() ) : $q->the_post(); ?>
                ...
                <?php endwhile; ?>
            </div>
            <?php
            $content .= ob_get_clean();
        }
    }
    return $content;
}
```

On va créer une nouvelle instance de la classe `WP_Query` en passant un tableau d'arguments. On va lui demander trois articles dans nos catégories, que l'on doit encore aller chercher.

Puis on a une boucle classique, mais qui utilise les méthodes `have_posts()` et `the_post()` de notre nouvelle `WP_Query`, et pas les fonctions qui, je le rappelle, vont travailler sur la globale `$wp_query`.

Pour écrire un peu de code HTML plus facilement, je mets un `ob_start()` juste après le `if()`, et je récupère le contenu du tampon juste avant de sortir du `if()`. Ainsi, rien n'est mis en tampon ni affiché s'il n'y a aucun article trouvé. Enfin, je retourne le contenu de l'article auquel j'ai ajouté le contenu de mon bloc HTML, car nous sommes dans le filtre `the_content`.

## Récupérer les catégories

Pour récupérer les catégories de l'article courant, on peut utiliser `get_the_category( $id = false )`. La fonction prend un identifiant optionnel en paramètre et va retourner un tableau d'objets `WP_Term` des catégories de l'article. Si on omet l'identifiant, la fonction retourne les catégories de l'article courant de la globale `$post`.

Ce qui est pratique, c'est que l'on a les objets `WP_Term`, donc toutes les informations sur les termes. Ce qui est pénible, c'est que l'on a les objets `WP_Term`, donc toutes les informations sur les termes.

Autrement dit, vu qu'on a besoin uniquement d'un tableau des `term_id` pour notre `WP_Query`, on peut soit faire une boucle et remplir un tableau, soit utiliser une fonction de ninja, assez peu connue curieusement mais super utile qui va le faire pour nous.

```
$category_ids = wp_list_pluck( get_the_category(), 'term_id' );  
  
$q = new WP_query( $args = array(  
    'posts_per_page' => 3,  
    'category__in' => $category_ids,  
) );
```

wp\_list\_pluck( array \$list, int|string \$field, int|string \$index\_key = null ) prend un tableau en paramètre, un champ ou un index en deuxième, et va retourner un nouveau tableau avec toutes les valeurs pour le champ (ou à l'index) spécifié. On peut aussi lui passer un autre champ en troisième paramètre dont les valeurs vont servir de clés. Elle fonctionne un peu comme array\_columns() sauf qu'elle marche aussi pour les objets.

Dans notre cas, on veut créer un tableau en extrayant les term\_id des objets retournés par get\_the\_category(), donc on lui passe le tableau créé par get\_the\_category() et le champ dont on veut les valeurs, soit term\_id. Magic. Personnellement, j'adore cette fonction.

On peut maintenant afficher le titre des articles trouvés dans notre boucle.

```
function wpcookbook_related_posts( $content ){  
    if( is_single() ){  
        ...  
        ?>  
        <div class="wpcookbook-related-posts">  
            <h3><?php esc_html_e( 'Related posts', '15-loops' ) ?></h3>  
            <?php while( $q->have_posts() ) : $q->the_post(); ?>  
                <?php the_title(); ?>  
            <?php endwhile; ?>  
        </div>  
        <?php  
        ....  
    }  
}
```

Nous sommes dans une boucle utilisant \$q->the\_post(), donc on peut utiliser la fonction the\_title() qui va chercher le titre dans la globale \$post.

## Carton rouge

Cependant, nous avons oublié une chose importante ! C'est de restaurer la globale \$post à sa valeur initiale ! En effet, après notre boucle, la globale ne contient plus les données de l'article principal. Si plus tard dans le modèle, une fonction vient à utiliser \$post d'une façon où d'une autre pour afficher des informations sur l'article principal, alors ces informations seront erronées !

Pour ce faire, on va juste utiliser la fonction `wp_reset_postdata()` après la boucle `while()` mais toujours dans le `if()`. La fonction complète donne ceci :

```
function wpcookbook_related_posts( $content ){
    if( is_single() ){
        $category_ids = wp_list_pluck( get_the_category(), 'term_id' );

        $q = new WP_query( array(
            'posts_per_page' => 3,
            'category__in'   => $category_ids,
        ) );
    }

    if( $q->have_posts() ){
        ob_start();
        ?>
        <div class="wpcookbook-related-posts">
            <h3><?php esc_html_e( 'Related posts', '15-loops' ) ?></h3>
            <?php while( $q->have_posts() ) : $q->the_post(); ?>
                <?php the_title(); ?>
            <?php endwhile; ?>
        </div>
        <?php
        $content .= ob_get_clean();
        wp_reset_postdata();
    }
    return $content;
}
```



# Hello world!

vincent · septembre 12, 2019 · Un commentaire · Modifier

Welcome to WordPress. This is your first post. Edit or delete it, then start writing !

## Related posts

Hello Universe !Hello world!

vincent · septembre 12, 2019 · Uncategorized · Modifier



Nos titres s'affichent correctement.

## Améliorons notre modèle

Maintenant, on peut améliorer notre extension en ajustant deux choses :

- En ajoutant un peu de balisage et en faisant en sorte que les titres des articles soient cliquables.  
Sinon, c'est inutilisable.
- En excluant l'article courant de la boucle. L'article se recommande lui-même, c'est un peu dommage.

On va s'occuper du balisage d'abord :

```
function wpcookbook_related_posts( $content ){
    ...
    ob_start();
    ?>
    <div class="wpcookbook-related-posts">
        <h3><?php esc_html_e( 'Related posts', '15-loops' ) ?></h3>
        <ul class="wpcookbook-related-posts-list">
            <?php while( $q->have_posts() ) : $q->the_post(); ?>
                <li class="wpcookbook-related-post">
                    <a href="<?php the_permalink(); ?>">
                        <?php the_title(); ?>
                    </a>
                </li>
            <?php endwhile; ?>
        </ul>
    </div>
    <?php
    $content .= ob_get_clean();
    ...
}
```

J'ai juste rajouté une `<ul>`, les `<li>` qui l'accompagnent et les liens. Les liens utilisent `the_permalink()` pour afficher l'URL de l'article courant dans la boucle. Souvenez-vous qu'une boucle faite avec `WP_Query` permet d'utiliser tous les templates tags de WordPress, exactement comme si vous étiez dans un modèle de page d'un thème.

Note : Pas besoin d'utiliser `esc_url()` pour le permalien, car la fonction `the_permalink()` le fait déjà. Aussi, on n'utilise pas de fonction pour échapper le contenu de `the_title()`, car ses paramètres peuvent contenir du code HTML. WordPress nous permet d'utiliser du balisage HTML dans les titres des articles (comme `<em>`ou `<code>`) et certains utilisateurs le font. Donc utiliser `esc_html()` rendrait cette fonctionnalité inutilisable. On pourrait utiliser `wp_kses_post()` à la place.

D'ailleurs, j'aurai dû utiliser les paramètres de `the_title()`. Pourquoi ? Pour qu'un utilisateur/extension utilisant le filtre `the_title` (qui est dans la fonction du même nom) puisse toucher si besoin à mon HTML. (J'explique cela un peu plus en détails au chapitre 9 - Comment naviguer dans vos thèmes et extensions)

```
function wpcookbook_related_posts( $content ) {
    ...
    <div class="wpcookbook-related-posts">
        <h3><?php esc_html_e( 'Related posts', '15-loops' ) ?></h3>
        <ul class="wpcookbook-related-posts-list">
            <?php
                while( $q->have_posts() ) {
                    $q->the_post();
                    the_title( sprintf( '<li class="wpcookbook-related-post"><a href="%s" rel="bookmark">', esc_url( get_permalink() ) ), '</a></li>' );
                }
            ?>
        </ul>
    </div>
    ...
}
```

WC WPCookBook Personnaliser 0 + Crée Modifier l'article

# Hello Galaxy !

vincent novembre 1, 2019 Laisser un commentaire Modifier

This is yet another dummy post

## Related posts

- [Hello Galaxy !](#) ←
- [Hello Universe !](#)
- [Hello world!](#)

vincent novembre 1, 2019 Test, WordPress snippet Modifier

—Article précédent

[Hello Universe !](#)

C'est déjà plus joli et fonctionnel !

Voilà qui est mieux. Maintenant, excluons notre article principal. Pour cela, il suffit d'ajouter un argument à notre `wp_query`.

```

$q = new WP_query( array(
    'posts_per_page' => 3,
    'category__in'    => $category_ids,
    'post__not_in'    => array( get_the_ID() ),
) );

```

`get_the_ID()` permet de récupérer l'identifiant de l'article courant dans la boucle. À ce moment-là, on n'est pas encore dans notre boucle personnalisée donc pas de souci, on va récupérer le bon identifiant (`$post` n'est pas encore modifiée). `post__not_in` accepte un tableau d'identifiants à exclure, tout simplement.

Voilà. Notre section “Articles Similaires” est fonctionnelle ! Évidemment, vous pouvez développer cela comme vous le souhaitez, en ajoutant des paramètres pour le nombre d’articles, en étendant la recherche d’articles à ceux ayant des étiquettes similaires, etc.

## Ce qu'il faut retenir

- Ce sont les fonctions/méthodes `have_posts()` et `the_post()` qui font tourner la machine. `have_posts()` vérifie qu'on n'est pas en bout de boucle, et `the_post()` avance dans la boucle et prépare la globale `$post` utilisée par les template tags de WordPress.
- La requête pour le contenu est faite avant que WordPress ne choisisse le modèle de page, et ses données sont stockées dans une globale `$wp_query`. `have_posts()` et `the_post()` ne font que s'y référer.
- Pour personnaliser les variables de la requête, on utilise le hook `pre_get_posts`, ainsi que les méthodes `$query->get()` et `$query->set()` pour récupérer et assigner des paramètres pour la requête.
- Attention à faire toutes les vérifications nécessaires pour n'affecter que les requêtes souhaitées, car le hook est déclenché à chaque fois qu'une instance de `WP_Query` utilise sa méthode `get_posts()` pour aller chercher du contenu. On dispose pour cela de plusieurs fonctions utiles: `is_main_query()`, `is_single()`, `is_page()`, `is_admin()`, etc. ainsi que des méthodes correspondantes.
- Attention, les fonctions listées précédemment font référence à la globale `$wp_query`, qui est la requête principale de la page, alors que les méthodes font référence à l'instance qui les appelle.
- Pour créer une boucle, on crée une nouvelle instance de `WP_Query`, en lui passant tous les paramètres nécessaires. Dans notre nouvelle boucle, attention à utiliser les méthodes de notre instance, et pas les fonctions !
- Tous les paramètres de `WP_Query` sont sur cette page :  
[https://developer.wordpress.org/reference/classes/wp\\_query/](https://developer.wordpress.org/reference/classes/wp_query/).
- Dans notre boucle personnalisée, on peut utiliser normalement tous les templates tags de WordPress, comme `the_title()`, `the_content()`, et on peut même utiliser `get_template_part()` pour aller chercher un modèle personnalisé dans le thème. Exactement comme dans un modèle de page normal.
- N'oubliez pas de restaurer la globale `$post` à sa valeur initiale avec `wp_reset_postdata()` après votre boucle personnalisée ! Sinon, tout code qui vient s'exécuter après votre boucle risque de voir ses données erronées.

`WP_Query` est une classe très puissante et on n'a fait que survoler tout son potentiel. Maintenant que vous

avez une idée claire des mécaniques, c'est à vous d'aller explorer tout ce qui est possible avec cette classe !

Pour information, il existe plusieurs fonctions permettant d'aller chercher des articles, qui utilisent ou non `WP_Query`.

Par exemple, pour notre section "Articles similaires", on aurait pu utiliser la fonction `get_posts()` qui prend des arguments similaires à une `WP_Query` standard, et qui retourne un tableau de publications. Mais elle ne permet pas d'utiliser simplement `the_post()` et les templates tags de WordPress. On aurait donc dû créer notre propre boucle et itérer manuellement sur chaque élément du tableau. Chaque fonction a ses avantages et inconvénients, mais `WP_Query` est la plus puissante et la méthode recommandée pour afficher du contenu sur le devant du site.

Amusez-vous !

# Comment créer des réglages dans l'outil de personnalisation des thèmes

Dans ce chapitre, on va voir comment fonctionne l'outil de personnalisation de WordPress et comment y ajouter nos réglages.

On ne parlera pas des pages de réglages dans l'administration C'est le sujet du chapitre Créez une page de réglages pour une extension dans la section sur les extensions.

Et puis de toute façon, les thèmes n'ont pas à avoir de page de réglages. Ils ne sont pas censés complexifier l'administration de votre site et ajouter des pages et des menus partout. Votre thème est censé s'occuper uniquement de l'affichage de votre site et on a un outil parfait pour faire vos réglages cosmétiques, c'est le Customizer (l'outil de personnalisation).

## Vue d'ensemble de l'outil : réglages, sections et panneaux

On accède à l'outil de personnalisation en cliquant sur Apparence > Personnaliser dans l'administration de WordPress. Si vous êtes connectés, vous pouvez aussi y accéder directement depuis le devant du site en cliquant sur le lien Personnaliser dans la barre d'outils d'administration.

Les réglages disponibles se présentent ainsi :

The screenshot shows the WordPress Customizer interface. On the left, there's a sidebar with a yellow border around the list of sections: Thème actif (Twenty Nineteen Child), Identité du site, Couleurs, Menus, Widgets, Réglages de la page d'accueil, and CSS additionnel. The main content area displays a post titled "Hello Galaxy!" with the text "This is yet another dummy post". Below the post, there are author details: vincent (novembre 1, 2019), Test, WordPress, snippet, and a link to Laisser un commentaire.

Les réglages disponibles ici sont ceux déclarés par WordPress et supportés par le thème twentynineteen, qui est le thème parent utilisé. Cela peut donc varier selon le thème.

Les réglages sont organisés en sections et panneaux. Si on clique sur Couleurs on accède à tous les réglages de la section Couleurs du thème. Si on clique sur la section Identité du site, on accède aux

réglages correspondants à cette section.

Les sections peuvent être aussi rangées dans des panneaux. Si on clique sur Menus, alors on accède aux sections correspondantes à chaque menu existant, présentant les réglages possibles pour ce menu. Si on clique sur le bouton Ajouter un menu, on accède à d'autres réglages de cette section.

Vous voyez l'organisation ? Les réglages sont toujours dans des sections, qui sont parfois dans des panneaux.

## Ajouter un réglage

Pour ajouter des réglages, sections et panneaux, WordPress met à notre disposition plusieurs classes et leurs méthodes.

Pour accéder au `Customize Manager` et son API, il faut se hooker sur `customize_register`. Le hook passe l'instance `$wp_customize` du `Customize Manager` à notre fonction. Dans le fichier `functions.php` (ou un autre, vu que vous rangez vos fonctionnalités correctement dans des fichiers séparés) de notre thème enfant de `twentynineteen`, ajoutez le code ci-dessous:

```
add_action( 'customize_register', 'twentynineteen_child_customize_register' );
/**
 * Registers our new Customizer settings, sections and panels
 *
 * @param WP_Customize_Manager $wp_customize The Customize Manager instance
 */
function twentynineteen_child_customize_register( $wp_customize ) {

}
```

Maintenant, on peut utiliser les méthodes de l'instance de `WP_Customize_Manager` passée en paramètre pour déclarer nos réglages. Ajoutons un simple texte de copyright à placer dans le pied de page du thème.

```
function twentynineteen_child_customize_register( $wp_customize ) {
    $wp_customize->add_setting( 'wpcookbook-copyright-text', array(
        'type'              => 'theme_mod',
        'capability'        => 'edit_theme_options',
        'theme_supports'    => '',
        'default'           => '',
        'transport'         => 'refresh',
        'validate_callback' => '',
        'sanitize_callback' => '',
        'sanitize_js_callback' => '',
    ) );
}
```

La méthode `add_setting( WP_Customize_Setting|string $id, array $args = array() )` prend deux paramètres : un slug/identifiant (ou instance de `WP_Customize_Setting`) et un tableau d'arguments (optionnel).

Parmi ces arguments :

- `type` accepte deux valeurs : `theme_mod` (par défaut) ou `option`. La différence réside dans la manière dont est stockée la valeur du réglage. Les `option` sont stockées dans la table `wp_options` comme n'importe quel autre réglage de WordPress ou des extensions, et s'appliquent quel que soit le thème utilisé. Les `theme_mod` sont spécifiques au thème. Si vous changez de thème, vous perdez vos réglages. Les thèmes devraient donc normalement utiliser `theme_mod` alors qu'une extension offrant des réglages d'affichage indépendants du thème peut très bien utiliser le type `option` pour ses réglages.
- `capability` est la capacité nécessaire pour avoir accès à ce réglage. `edit_theme_options` (par défaut) est adéquate pour un thème. Pour une extension, on peut aussi utiliser `manage_options` qui est normalement une capacité des administrateurs.
- `theme_supports` est la fonctionnalité du thème nécessaire pour afficher le réglage. C'est-à-dire que le thème doit déclarer la fonctionnalité correspondante via `add_theme_support()` pour que le réglage soit accessible. WordPress prend le soin de vérifier le support pour les images d'entête personnalisées par exemple avant d'afficher les réglages correspondants. Vous pouvez faire de même pour vos réglages.
- `default` est une valeur par défaut pour le réglage.
- `transport` est la façon dont les réglages vont s'actualiser dans la fenêtre de prévisualisation. `refresh` (par défaut) va déclencher un rafraîchissement complet de la page dans la fenêtre de prévisualisation, alors que `postMessage` va permettre une mise à jour instantanée, mais va demander de coder le JavaScript ou PHP pour actualiser le contenu de la fenêtre de prévisualisation. Plus de détails sur ce sujet plus loin.
- `validate_callback` est le nom d'une fonction de rappel utilisée pour valider la valeur du réglage.
- `sanitize_callback` est le nom d'une fonction de rappel utilisée pour nettoyer la valeur du réglage.
- `sanitize_js_callback` est le nom d'une fonction de rappel utilisée pour s'assurer que la valeur du réglage est compréhensible par du JavaScript. C'est rarement nécessaire.

On parlera plus en détails des paramètres de transport, de validation et nettoyage plus tard. Pour l'instant, on va se contenter des valeurs par défaut pour tous ces paramètres. Attention, ce n'est pas une bonne pratique ! Il faut toujours passer au moins une `sanitize_callback`. On met juste ça de côté temporairement pour simplifier les explications.

Seule, cette méthode ne fait pas grand chose, si ce n'est déclarer un réglage. Mais pour que ce réglage apparaisse dans notre outil de personnalisation, il faut lui associer un `WP_Customize_Control`, c'est-à-dire un contrôleur qui va nous permettre de modifier sa valeur. Comme un simple champ texte, par exemple.

Toujours dans notre fonction de rappel hookée sur `customize_register`, ajoutez ceci juste en dessous de notre appel à `$wp_customize->add_setting()` :

```

$wp_customize->add_control( 'wpcookbook-copyright-text', array(
    'type'          => 'text',
    'priority'      => 10,
    'section'       => 'title_tagline',
    'label'         => __( 'Footer copyright text', '16-customizer' ),
    'description'   => __( 'This is the text to display in the footer instead of the theme\'s
credit line.' ),
    // 'choices'      => array(
    //     'first-value' => __( 'First value', '16-customizer' ),
    //     'another-value' => __( 'Another value', '16-customizer' ),
    //     'yet-another-value' => __( 'Yet Another value', '16-customizer' ),
    // ),
    'active_callback' => '',
    'inputAttrs'     => array(
        'class'      => 'footer-copyright',
        'style'      => 'border: 2px solid black', // Just for fun
        'placeholder' => __( 'Copyright Vincent Dubroeucq, 2019', '16-customizer' ),
    ),
) );
)
);

```

Tout comme la méthode `add_setting()`, `add_control( WP_Customize_Control|string $id, array $args = array() )` prend deux arguments : un slug/identifiant (ou instance de `WP_Customize_Control`) et un tableau d'arguments optionnels. Le slug doit correspondre à l'identifiant d'un réglage pour qu'il lui soit bien associé. On lui passe donc '`wpcookbook-copyright-text`'.

Voici les arguments :

- `type` est le type de champ. `text` est la valeur par défaut. `checkbox`, `radio`, `select`, `textarea`, `email`, `date`, `hidden`, `url`, `number`, et `dropdown-pages` sont acceptés.
- `priority` est la priorité du contrôle dans la section. Cela permet d'ajuster sa position.
- `section` est la section dans laquelle vous voulez afficher le contrôle.
- `label` est le `<label>` du champ.
- `description` est une description du champ.
- `choices` est un tableau de valeur `=> label` à ajouter quand vous utilisez un champ `radio` ou `select`. Ici, on utilise un bête champ texte, donc ce paramètre sera ignoré. Je le donne pour votre information.
- `active_callback` est une fonction de rappel déterminant quand faire apparaître le champ. Par défaut les champs sont toujours présents, mais vous pouvez par exemple passer `'is_home'` pour faire apparaître ce champ uniquement quand la fonction `is_home()` renverra `true`, donc uniquement sur la page listant les articles. Vous pouvez passer aussi toute autre fonction de WordPress renvoyant un booléen, comme `is_single()`, `is_page()`, etc ou une de vos fonctions maison (qui doit renvoyer un booléen).
- `inputAttrs` est un tableau d'attributs à utiliser pour le balisage de votre champ. Vous pouvez ainsi passer une classe CSS, un attribut `style` ou tout autre attribut que vous jugez utile. Je l'utilise personnellement très peu et je l'ai inclus ici uniquement pour l'exemple. Vous pouvez l'omettre sans souci.

Dans les deux snippets précédents, j'ai inclus tous les paramètres possibles, mais vous remarquerez que la plupart d'entre eux ont leur valeur par défaut. C'est simplement pour vous montrer l'étendue des paramètres. On aurait pu donc racourcir ces deux snippets ainsi :

```

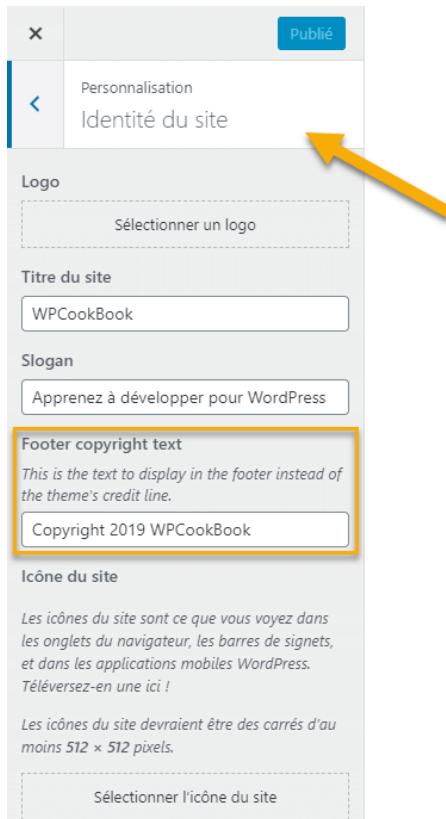
function twentynineteen_child_customize_register( $wp_customize ) {
    $wp_customize->add_setting( 'wpcookbook-copyright-text' );
    $wp_customize->add_control( 'wpcookbook-copyright-text', array(
        'section'      => 'title_tagline',
        'label'        => __( 'Footer copyright text', '16-customizer' ),
        'description' => __( 'This is the text to display in the footer instead of the theme\'s credit line.' ),
    ) );
}

```

Je vous conseille tout de même de garder les snippets complets pour le moment, car on va jouer avec les valeurs de tous les paramètres.

## Utiliser notre réglage

Si vous allez dans l'outil de personnalisation, notre champ apparaît bien dans la section Identité du site, vous pouvez changer sa valeur et sauvegarder, mais il ne se passe rien sur le devant du site. C'est normal, il faut maintenant utiliser notre réglage dans notre thème.



WPCookBook Apprenez à développer pour WordPress  
Page Article Catégorie Contact

# Hello Galaxy !

This is yet another dummy post

vincent novembre 1, 2019 Test, WordPress  
snippet Laisser un commentaire

Notre réglage apparaît bien dans l'outil de personnalisation

Pour utiliser notre réglage, on va utiliser les fonctions `get_theme_mod()` ou `get_option()` selon si on a décidé d'utiliser des `theme_mod` ou `option` pour récupérer sa valeur.

La fonction `get_theme_mod( string $name, string|false $default = false )` prend deux paramètres : le nom `$name` du réglage dont on a besoin et une valeur par défaut. Si on omet cette

deuxième valeur, alors la fonction retournera `false` si la valeur du réglage n'est pas trouvée. La fonction `get_option()` fonctionne exactement de la même manière, donc pas de souci.

On veut remplacer le texte du pied de page dans notre thème enfant, mais seulement si une valeur pour notre réglage est existante. On va donc dupliquer le fichier `footer.php` de notre thème parent dans notre thème enfant, et déplacer le code affichant le nom du site et le lien vers WordPress.org dans un `if()`.

```
// in footer.php
<footer id="colophon" class="site-footer">
    <?php get_template_part( 'template-parts/footer/footer', 'widgets' ); ?>
    <div class="site-info">
        <?php if ( $footer_text = get_theme_mod( 'wpcookbook-copyright-text' ) ) : ?>
            <p class="site-copyright">
                <strong><?php echo esc_html( $footer_text ); ?></strong>
            </p>
        <?php else: ?>
            <?php $blog_info = get_bloginfo( 'name' ); ?>
            <?php if ( ! empty( $blog_info ) ) : ?>
                <a class="site-name" href="<?php echo esc_url( home_url( '/' ) ); ?>" rel="home"><?php bloginfo( 'name' ); ?></a>,
            <?php endif; ?>
                <a href="<?php echo esc_url( __( 'https://wordpress.org/' ), 'twentynineteen' ) ); ?>" class="imprint">
                    <?php
                    /* translators: %s: WordPress. */
                    printf( __( 'Proudly powered by %s.', 'twentynineteen' ), 'WordPress' );
                    ?>
                </a>
            <?php endif; ?>
            ...
    </div>
</footer><!-- #colophon -->
```

On affecte directement la valeur de notre réglage à une variable dans le `if()` et si on a une valeur, alors on l'affiche dans un paragraphe, en prenant soin de l'échapper. Sinon, on affiche le texte par défaut fourni par le thème.

Maintenant, on peut modifier notre réglage, et voir le texte du pied de page changer après chaque rafraîchissement de la fenêtre de prévisualisation.

Publié

Personnalisation  
Identité du site

Logo

Sélectionner un logo

Titre du site

WPCookBook

Slogan

Apprenez à développer pour WordPress

Footer copyright text  
*This is the text to display in the footer instead of the theme's credit line.*

Copyright 2019 WPCookBook

Icône du site

Les icônes du site sont ce que vous voyez dans les onglets du navigateur, les barres de signets, et dans les applications mobiles WordPress.  
Téléversez-en une ici !

Les icônes du site devraient être des carrés d'au moins 512 x 512 pixels.

Sélectionner l'icône du site

Masquer les contrôles

Recherche...

Rechercher

Copyright 2019 WPCookBook

Notre réglage fonctionne correctement !

## Ajouter une section

Pour l'instant on a rangé notre réglage dans la section Identité du site, ce qui est compréhensible. Mais si on a besoin d'ajouter d'autres réglages dans le pied de page, il serait peut-être judicieux de créer une section Pied de page.

Pour ce faire, on utilise la méthode `add_section()` de notre instance `$wp_customize`. La méthode `WP_Customize_Manager::add_section( WP_Customize_Section|string $id, array $args = array() )` prend deux paramètres: un slug/identifiant pour notre section (ou instance de `WP_Customize_Section`), et un tableau d'arguments :

- `title` est le titre de la section.
- `description` est un description à afficher.
- `description_hidden` est un booléen indiquant s'il faut cacher la description derrière une petite icône d'aide.
- `type` est le type de section ('`default`' est la valeur par défaut). Utile quand on veut déclarer des sections spéciales, avec leur propre modèle. On n'en parlera malheureusement pas dans ce guide. C'est un sujet plus avancé.

- `panel` est le panneau dans lequel insérer la section. C'est vide par défaut, ce qui insère la section dans le panneau d'accueil.
- `priority` est la priorité de la section dans son panneau. Cela permet de contrôler sa position. (160 par défaut)
- `capability` est la capacité que l'utilisateur doit avoir pour accéder à la section. (`edit_theme_options` par défaut)
- `theme_supports` est la fonctionnalité du thème nécessaire pour afficher la section.
- `active_callback` est une fonction de rappel déterminant quand faire apparaître la section. Cela fonctionne exactement comme pour les contrôles.

La plupart des arguments sont assez simples et sont les mêmes que pour les contrôles. Dans notre fonction de rappel, sous notre réglage et son contrôle, ajoutez ceci :

```
// in callback hooked on customize_register
$wp_customize->add_section( 'wpcookbook-footer', array(
    'title'          => __( 'Footer settings', '16-customizer' ),
    'description'    => __( 'Adjust your footer preferences here', '16-customizer' ),
    'description_hidden' => false,
    'type'           => 'default',
    'panel'          => '',
    'priority'        => 160,
    'capability'      => 'edit_theme_options',
    'theme_supports'   => '',
    'active_callback'  => '',
) );
```

Encore une fois, on utilise beaucoup de paramètres par défaut, que l'on pourrait donc omettre. Ils sont inclus ici pour avoir une vision globale.

Si on retourne sur notre outil de personnalisation, on voit que la section n'apparait pas, simplement parce que l'outil est assez malin pour ne pas afficher de section vide. Il suffit de lui affecter notre réglage en modifiant le paramètre `section` du contrôle.

```
$wp_customize->add_control( 'wpcookbook-copyright-text', array(
    ...
    'section'        => 'wpcookbook-footer',
    ...
) );
```

The screenshot shows a sidebar titled "Personnalisation Footer settings". It contains a section for "Footer copyright text" with the placeholder "This is the text to display in the footer instead of the theme's credit line." A red box highlights the "Footer copyright text" section. Below it is a box containing the text "Copyright Vincent Dubroeucq, 2019".

WPCookBook Apprenez à développer pour WordPress  
Page Article Catégorie Contact

# Hello Galaxy !

This is yet another dummy post

vincent novembre 1, 2019 Test, WordPress  
snippet Laisser un commentaire

Notre section est accessible et contient bien notre réglage.

La section apparaît en bas de la liste des sections, juste avant CSS additionnel. On peut jouer sur la priorité pour le faire apparaître plus haut. Pour vous aider, voici un tableau des sections par défaut, de leur slug (utile pour ranger vos réglages), et de leur priorité (utile pour les ordonner).

| Titre                         | Slug              | Priorité |
|-------------------------------|-------------------|----------|
| Identité du site              | title_tagline     | 20       |
| Couleurs                      | colors            | 40       |
| Image d'en-tête               | header_image      | 60       |
| Image d'arrière-plan          | background_image  | 80       |
| Menus (Panneau)               | nav_menus         | 100      |
| Widgets (Panneau)             | widgets           | 110      |
| Réglages de la page d'accueil | static_front_page | 120      |
| CSS Additionnel               | custom_css        | 200      |

Donc, en ajustant la priorité à 90 par exemple, la section apparaîtra juste après les réglages des images d'en-tête et d'arrière-plan.

## Ajouter un panneau

Pour ajouter un panneau, donc un groupe de sections, le procédé est exactement le même. On va utiliser la méthode `add_panel()` de notre `$wp_customize`.

```
$wp_customize->add_panel( 'wpcookbook-theme-options', array(
    'title'          => __( 'Theme options', '16-customizer' ),
    'description'   => __( 'Adjust all your child theme\'s settings here.', '16-customizer'
),
    'type'           => 'default',
    'priority'       => 160,
    'capability'    => 'edit_theme_options',
    'theme_supports' => '',
    'active_callback' => '',
    'auto_expand_sole_section' => false,
) );
```

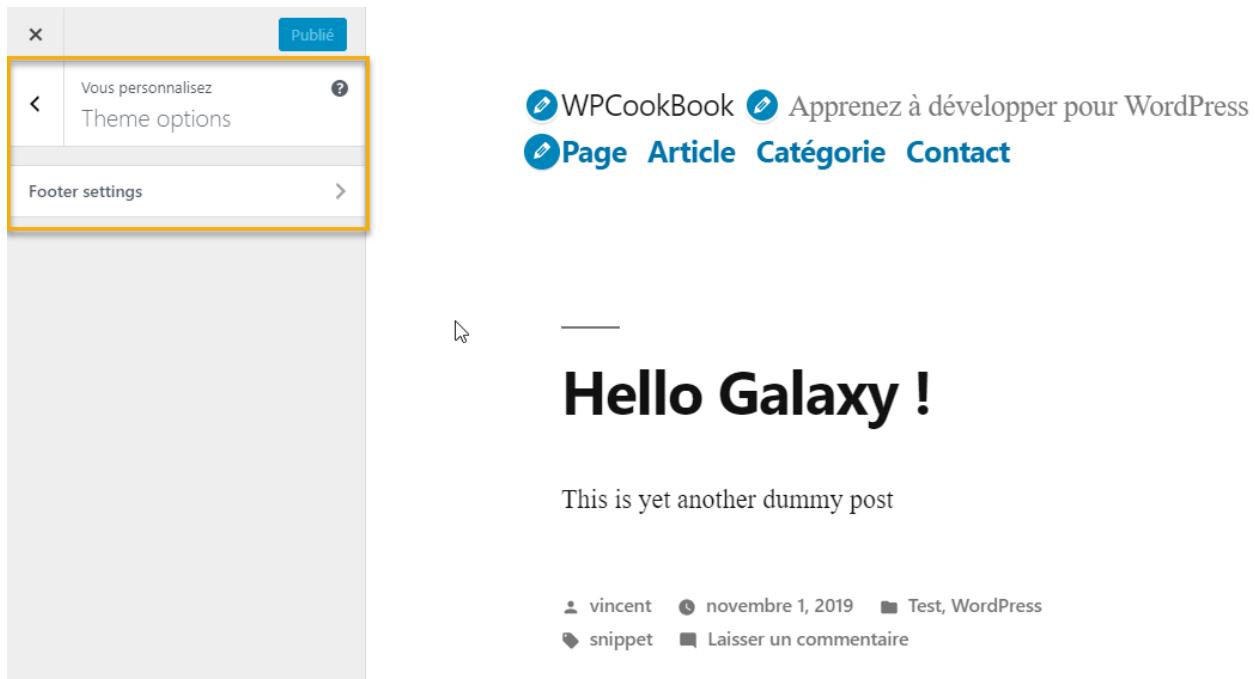
Exactement comme pour les autres méthodes, on passe un identifiant/slug et un tableau d'arguments. Ici, mis à part la description et le titre, on utilise toutes les valeurs par défaut. Aussi, les paramètres sont globalement les mêmes que pour `add_section()`.

Le seul paramètre nouveau est '`auto_expand_sole_section`' qui permet d'ouvrir automatiquement une section si elle est la seule dans le panneau. Honnêtement, si c'est pour créer un panneau avec une seule section qui s'ouvre automatiquement, autant créer une section directement. A moins que vous ne prévoyez une extension qui viendrait ajouter des sections dans ce panneau, alors que votre thème ne propose qu'une seule section de base.

Par contre, évitez de créer des panneaux si ce n'est pas nécessaire. Si vous pouvez ranger tous vos réglages dans les sections par défaut de WordPress ou alors ne créer que quelques sections supplémentaires, faites ainsi.

Comme pour notre section, le panneau n'apparaîtra que si des sections lui sont affectées. Donc on va modifier le paramètre '`panel`' de notre section.

```
$wp_customize->add_section( 'wpcookbook-footer', array(
    ...
    'panel'          => 'wpcookbook-theme-options',
    ...
) );
```



Notre panneau est accessible et contient bien notre section.

## Petit résumé

On a vu comment déclarer un réglage basique avec la méthode `add_setting()`, son contrôle avec `add_control()`, une section avec `add_section()`, et un panneau avec `add_panel()`. On a aussi vu comment récupérer la valeur de nos réglages en utilisant `get_theme_mod()` ou `get_option()` selon le type de réglage déclaré, pour l'utiliser dans notre thème.

Mécaniquement parlant, c'est (presque) tout ce qu'il y a à savoir. Prenez un peu de temps pour ajouter des réglages, en changeant le type de contrôle. Essayez un champ `textarea`, un champ `radio`, etc.

Expérimitez !

- Pour agir sur l'outil de personnalisation, on se hooke sur `customize_register`.
- Pour déclarer un réglage, on utilise la méthode `add_setting()`. On lui passe un identifiant et un tableau d'arguments.
- Pour déclarer un contrôle, on utilise la méthode `add_control()`. On lui passe l'identifiant du réglage associé et un tableau d'arguments. La majorité des champs de formulaires standards sont acceptés. N'oubliez pas de passer un tableau `choices => array( 'value' => 'label', )` si vous utilisez un `select` ou `radio`.
- Pour déclarer une section, on utilise la méthode `add_section()`.
- Pour déclarer un panneau, on utilise la méthode `add_panel()`.
- Les réglages sont toujours dans des sections, et les sections sont parfois dans des panneaux.
- On évite de multiplier les panneaux et on utilise les sections déjà présentes au maximum.
- On récupère la valeur de nos réglages avec `get_theme_mod()` ou `get_option()` selon le type de réglage.
- Pour toutes ces méthodes, les valeurs par défaut sont très bien. Sauf pour les labels et descriptions, bien entendu ! Donc c'est très facile !

Dans la suite, on va attaquer des sujets plus pointus : comment utiliser des contrôles plus complexes,

comment valider et sécuriser nos données et comment améliorer l'expérience utilisateur.

## Utiliser des contrôles avancés

### WP\_Customize\_Image\_Control

WordPress met à notre disposition des contrôles plus complexes que des simples champs de formulaire. Il nous permet aussi de créer nos propres contrôles en étendant la classe `WP_Customize_Control`. On ne créera pas de contrôles dans ce chapitre, car c'est un sujet plus complexe et avancé. Mais on peut explorer quelques-uns des contrôles fournis par WordPress.

On peut par exemple créer un réglage pour ajouter un logo dans notre pied de page. Dans notre fonction hookée sur `customize_register`, ajoutez :

```
// Let's add a footer logo setting, using WordPress' Media Control
$wp_customize->add_setting( 'wpcookbook-footer-logo', array(
    // 'sanitize_callback' => '',
) );
$wp_customize->add_control( new WP_Customize_Image_Control( $wp_customize, 'wpcookbook-footer-logo', array(
    'label'    => __( 'Footer logo', '16-customizer' ),
    'section'  => 'wpcookbook-footer',
) ) );
```

Souvenez-vous, le tableau d'arguments pour `add_control()` est optionnel. Ici au lieu de passer un identifiant, on passe une instance de `WP_Customize_Image_Control`. Et c'est à cette instance que l'on passe notre `$wp_customize` pour le contexte, l'identifiant du réglage pour lequel on déclare notre contrôle, et notre tableau d'arguments.

Si vous allez dans la section Footer settings de notre panneau Theme options, vous verrez un bouton qui permet d'ajouter une image, en utilisant la galerie média de WordPress et notre réglage est bien sauvegardé quand on clique sur le bouton Publier.

Maintenant, pour voir ce qui sauvegardé exactement, on peut simplement faire un `var_dump` de la valeur de notre réglage. Dans `footer.php`, ajoutez juste cette ligne juste avant le code utilisé pour afficher notre précédent texte de pied de page :

```
// in footer.php
...
<div class="site-info">
    <?php var_dump( get_theme_mod( 'wpcookbook-footer-logo' ) );?>
    <?php if ( $footer_text = get_theme_mod( 'wpcookbook-copyright-text' ) ) : ?>
        ...
    <?php else: ?>
```

En allant dans notre outil de personnalisation et en rafraîchissant, on obtient ceci :

The screenshot shows the WordPress Customizer interface. In the top right corner, there's a 'Publié' button. Below it, a navigation bar includes a back arrow, 'Personnalisation • Theme options', and 'Footer settings'. A search bar with a magnifying glass icon and the placeholder 'Recherche...' is positioned above a blue 'Rechercher' button. To the right, a sidebar titled 'Articles récents' lists three posts: 'Hello Galaxy ! novembre 1, 2019', 'Hello Universe ! novembre 1, 2019', and 'Hello world! septembre 12, 2019'. Below the sidebar, a URL 'string(64) "https://wpcookbook.local/wp-content/uploads/2019/11/sandwich.jpg"' is displayed, with a yellow arrow pointing to it from the left. At the bottom left, there's a 'Footer logo' section with a thumbnail of a sandwich, a 'Retirer' button, and a 'Changer l'image' button. A 'Masquer les contrôles' button is at the bottom right.

Notre réglage fonctionne, mais c'est l'URL de l'image originale qui est enregistrée.

## WP\_Customize\_Cropped\_Image\_Control

`WP_Customize_Image_Control` enregistre l'URL de l'image originale. Ce n'est pas exactement ce que l'on veut. On a plutôt besoin d'une petite image, que l'on peut rogner si besoin. On va donc utiliser un autre contrôle :

```
$wp_customize->add_control( new WP_Customize_Cropped_Image_Control( $wp_customize, 'wpcookbook-footer-logo', array(
    'label'      => __( 'Footer logo', '16-customizer' ),
    'section'    => 'wpcookbook-footer',
    'height'     => 75,
    // 'width'      => 150, // Default 150
    // 'flex_height' => false, // Default
    'flex_width' => true, // Default false
) ) );
```

Ce contrôle permet de rogner les images aux dimensions passées en paramètres. Par défaut, il permet à l'utilisateur de rogner son image en vignette de 150px de haut et de large, comme la plus petite taille d'image. On a besoin de quelque chose de moins haut, mais potentiellement plus long. On va donc lui passer `height` à 75px et `flex_width` à `true`.

Adjust your footer preferences here

**Footer copyright text**  
This is the text to display in the footer instead of the theme's credit line.

Copyright 2019 WPCookBook

**Footer logo**

Retirer Changer l'image

Recherche... Rechercher

int(96) ←

Copyright 2019 WPCookBook

On peut redimensionner notre vignette maintenant.

C'est beaucoup mieux. Si vous visitez votre bibliothèque de médias en mode liste dans l'administration, vous pouvez voir qu'on a bien une image redimensionnée, qui fait 75px de haut. De plus, comme vous pouvez le voir dans le pied de page, c'est l'identifiant de l'image qui est sauvegardé. Ça c'est cool, car c'est plus simple d'utiliser l'identifiant de l'image pour pouvoir récupérer son URL, son texte alternatif, etc.

Dans notre modèle `footer.php`, on va utiliser notre réglage et insérer notre image.

```
// in footer.php
...
<div class="site-info">
    <?php
        if( $logo = (int) get_theme_mod( 'wpcookbook-footer-logo' ) ){
            $src  = wp_get_attachment_url( $logo );
            $alt  = ! empty( get_post_meta( $logo, '_wp_attachment_image_alt', true ) ) ?
                get_post_meta( $logo, '_wp_attachment_image_alt', true ) : get_bloginfo( 'name' );
            printf( '', esc_url( $src ),
                esc_attr( $alt ) );
        }
    ?>
    <?php if ( $footer_text = get_theme_mod( 'wpcookbook-copyright-text' ) ) : ?>
    ...
    <?php else: ?>
```

Comme pour notre copyright, on vérifie d'abord qu'il y a bien une valeur enregistrée pour le réglage du logo. Normalement, on devrait avoir stocké un identifiant numérique, mais pour nous assurer que la donnée est bien un `int` et pas une `string` quelconque, on la force en `int`.

Puis, si on a une image, on utilise `wp_get_attachment_url()` pour récupérer son URL directement. Notre image est petite, elle n'a donc pas de taille intermédiaire et on peut directement récupérer l'URL de l'image originale redimensionnée. On utilise `get_post_meta()` pour récupérer le texte alternatif. S'il n'y en a pas, on utilise le titre du site à la place, avec `get_bloginfo()`. Puis un `printf()` pour afficher la balise `<img>` et le tour est joué. On parlera plus en détails des post meta plus loin dans ce guide.

Vous pouvez complexifier/améliorer ce petit morceau de template, évidemment. Je vous laisse faire ! HTML, CSS, tout ce que vous voulez !

Au fait, on n'a toujours pas donné de valeur pour notre paramètre `sanitize_callback`. Maintenant qu'on sait que c'est un identifiant numérique qui est sauvegardé, on peut corriger cela et lui passer `absint`.

```
$wp_customize->add_setting( 'wpcookbook-footer-logo', array(
    'sanitize_callback' => 'absint',
) );
```

## WP\_Customize\_Color\_Control

Déclarons un nouveau réglage pour changer la couleur du texte. Toujours dans notre fonction hookée sur `customize_register` :

```
// Let's add another color setting, using WordPress' Color Control
$wp_customize->add_setting( 'wpcookbook-text-color', array(
    'sanitize_callback' => 'sanitize_hex_color',
) );
$wp_customize->add_control( new WP_Customize_Color_Control( $wp_customize, 'wpcookbook-
text-color', array(
    'label'    => __( 'Text Color', '16-customizer' ),
    'section' => 'colors',
) ) );
```

On déclare un réglage et une fonction de rappel de nettoyage appelée `sanitize_hex_color()` qui, comme son nom l'indique, va nettoyer un code couleur hexadécimal.

Si vous allez dans la section Couleurs, vous verrez un beau color picker bien pratique et la valeur de notre réglage est bien sauvegardée quand on clique sur le bouton Publier.

Maintenant, il nous faut utiliser ce réglage. Pour l'exemple, on va faire quelque chose de bien flagrant et modifier la couleur de tout le texte du site. Dans `functions.php`, ajoutez :

```
// in functions.php
add_action( 'wp_head', 'wpcookbook_text_color' );
/**
 * Prints <style> tag to change the body text color
 */
function wpcookbook_text_color(){
    $color = get_theme_mod( 'wpcookbook-text-color' );
    if( $color ){
        echo '<style>body{color:' . sanitize_hex_color( $color ) . '};</style>';
    }
}
```

On vérifie la valeur de notre réglage, et on ajoute une balise `<style>`. Tout simple.

The screenshot shows the WordPress Customizer interface. On the left, the 'Personnalisation' (Customization) section is selected, and within it, the 'Couleurs' (Colors) panel. A color picker dialog is open, showing a color preview, a hex code field containing '#81d742', and a 'Sélectionner une couleur' button. Below the picker is a color palette and a checked checkbox labeled 'Appliquer un filtre aux images mises en avant en utilisant la couleur principale' (Apply a filter to featured images using the main color). To the right, a post preview is shown with the title 'Hello Galaxy!' in a large green font and the text 'This is yet another dummy post'. Below the preview, author information shows 'vincent' posted on 'novembre 1, 2019' under 'Test, WordPress' category, with a snippet and a comment link.

Le color picker est pratique et simple à mettre en place.

## WP\_Customize\_Date\_Time\_Control

WordPress met aussi à notre disposition un champ de sélection de date. Honnêtement, ce n'est pas le meilleur date picker du monde, et un bête champ `type="date"` pourrait faire l'affaire, mais bon. Voilà tout de même un exemple :

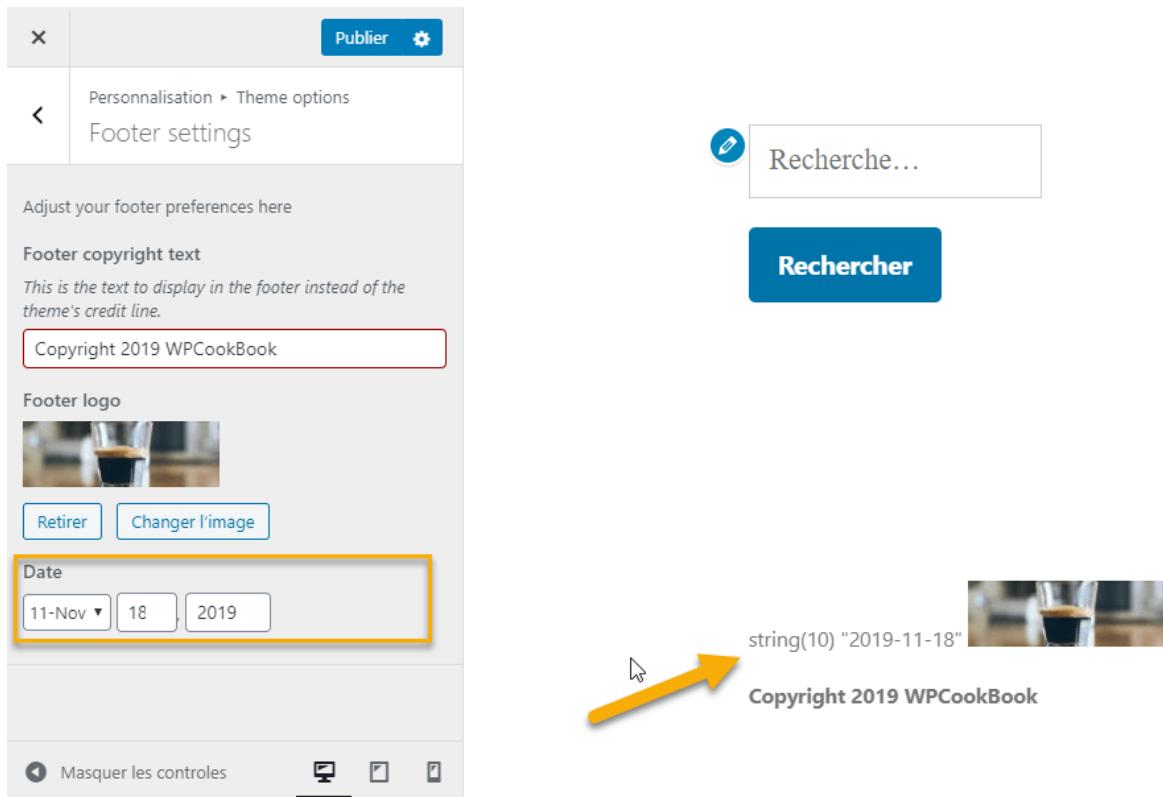
```

$wp_customize->add_setting( 'wpcookbook-date', array(
    'sanitize_callback' => 'sanitize_text_field',
) );
$wp_customize->add_control( new WP_Customize_Date_Time_Control( $wp_customize, 'wpcookbook-
date', array(
    'label'          => __( 'Date', '16-customizer' ),
    'section'        => 'wpcookbook-footer',
    'include_time'   => false, // Default true
    'min_year'       => 2019, // Default 1000
    'allow_past_date' => false // Default true
) ) );

```

Les paramètres sont assez simples. Ici, `include_time` permet d'ajouter un champ heure ou non, et on peut choisir l'année minimum et d'autoriser ou non les dates passées. Ce qui est intéressant, c'est que le rafraîchissement de la fenêtre de prévisualisation ne se déclenche que quand la date est valide, donc dans notre cas, une date future. On a donc une validation dans les coulisses. Plutôt cool.

Pour le moment, on nettoie uniquement avec `sanitize_text_field()` car la valeur sauvegardée est une chaîne ayant le format YYYY-MM-DD. Le traitement des dates mérite un chapitre complet, donc on ne va pas s'étendre dessus maintenant.



Le date picker par défaut. Pas très impressionnant.

## Valider, nettoyer et échapper nos champs

Jusqu'à maintenant, on avait mis plus ou moins de côté les paramètres `sanitize_callback`, `validate_callback`, et `sanitize_js_callback`. Ces paramètres sont très importants

On ne va pas forcément aller dans le détail ici, car la sécurité c'est un sujet qui mérite un ebook entier, et d'autres personnes pourraient en parler bien mieux que moi. Je vous renvoie vers [Julio Potier](#). Cela dit, j'aimerais attirer votre attention sur la différence entre valider et nettoyer et pourquoi c'est important.

Valider une entrée utilisateur, c'est s'assurer qu'elle soit bien dans la forme attendue. Ça se passe généralement sur le devant du site, avant de sauvegarder quoi que ce soit en base de données.

Par exemple, sur un formulaire classique un champ mot de passe peut exiger de l'utilisateur que le mot de passe fasse plus de 8 caractères, contiennent des chiffres, minuscules, majuscules et caractères spéciaux. Un petit bout de JavaScript va vérifier ce qui est entré par l'utilisateur et s'il y a un souci, va lui signaler son erreur et va bloquer la sauvegarde. Le but est d'améliorer l'expérience utilisateur, en évitant d'envoyer vers le serveur une donnée non utilisable, et que ce dernier renvoie l'utilisateur vers le formulaire avec un message d'erreur.

Nettoyer est une opération qui est effectuée côté serveur, quand la valeur est soumise. Juste avant de la sauvegarder en base, on va vérifier que la valeur soumise ne contient pas d'éléments compromettants ou non autorisés, et on va les supprimer le cas échéant. On se situe donc après que l'utilisateur ait soumis son formulaire, et on prépare la donnée pour la sauvegarder. Mais la valeur sauvegardée en base peut être différente de la valeur entrée par l'utilisateur !

Échapper signifie prendre les mesures nécessaires pour que la donnée puisse être utilisée en toute sécurité dans son contexte. Par exemple, `esc_html()` va remplacer les < par &lt; pour éviter d'interpréter le code HTML de la chaîne passée en paramètre quand elle est affichée.

## Échappement

Commentez le paramètre `sanitize_callback` du contrôle pour notre champ `wpcookbook-copyright-text`, et dans `footer.php`, enlevez l'appel à `esc_html()`. Le code HTML sera donc affiché et interprété tel quel. En gros, on ne fait plus rien côté serveur pour nettoyer, ni sur le devant du site pour échapper.

```
$wp_customize->add_setting( 'wpcookbook-copyright-text', array(
    'validate_callback' => '',
    'sanitize_callback' => '',
    'sanitize_js_callback' => '',
) );
```

```
// Footer.php
<?php if ( $footer_text = get_theme_mod( 'wpcookbook-copyright-text' ) ) : ?>
<p class="site-copyright">
    <strong><?php echo $footer_text; ?></strong>
</p>
<?php else: ?>
```

Maintenant, dans notre outil de personnalisation de WordPress, vous pouvez entrer du code HTML dans le champ texte et il est interprété quand il est affiché. Y compris les balises <script> !

The screenshot shows the WordPress 'Footer settings' page. In the 'Footer copyright text' field, the value is set to '`<em><script>alert('Hello')</script></em>`'. A yellow arrow points from this text to a modal dialog box titled 'wpcookbook.local indique'. The dialog contains the text 'Hello' and an 'OK' button. To the right of the dialog, the footer content is displayed as 'Hello Galaxy Hello Universe 2019 Hello world'. Below the footer content, the text 'Copyright 2019 WPCookBook' is visible.

On met ce qu'on veut. C'est la fête ! Yikes !

Est-ce que ce comportement est désirable ? Il y a peu de chances ! Pour éviter ça, on doit échapper la donnée sur le devant du site. Si vous réajoutez l'appel à `esc_html()` dans le fichier `footer.php`, alors notre HTML est encodé pour ne pas être interprété.

```
// Footer.php
<?php if ( $footer_text = get_theme_mod( 'wpcookbook-copyright-text' ) ) : ?>
    <p class="site-copyright">
        <strong><?php echo esc_html( $footer_text ); ?></strong>
    </p>
<?php else: ?>
```



`<em>Copyright 2019 WPCookBook<script> alert('hello')</script></em>`

Notre valeur s'affiche sans risque

## Nettoyage

C'est déjà mieux, car notre valeur est affichée sans risque. Mais en aucun cas cela ne nous empêche de sauvegarder des données non souhaitées en base de données.

Pour nettoyer le champ texte de ses balises HTML au moment de la sauvegarde, on peut utiliser `sanitize_text_field()`.

```
$wp_customize->add_setting( 'wpcookbook-copyright-text', array(
    'validate_callback' => '',
    'sanitize_callback' => 'sanitize_text_field',
    'sanitize_js_callback' => '',
) );
```

Maintenant, même si on écrit des balises HTML dans notre réglage, celles-ci sont purement et simplement supprimées et il ne nous reste que le texte dans la fenêtre de prévisualisation.

The screenshot shows the 'Footer settings' section of the WordPress theme options. A yellow box highlights the 'Footer copyright text' input field, which contains the HTML code: <em>Copyright 2019 WPCookBook</script>alert. To the right, a preview window shows a logo and the text 'Copyright 2019 WPCookBook' instead of the intended alert message. A large orange arrow points from the highlighted field to the preview area.

On a du code HTML dans le champ, mais rien dans la fenêtre de prévisualisation.

Mais surprise, quand on publie nos changements et qu'on rafraîchit la page, il ne nous reste que notre **texte dans la valeur du champ** ! C'est simplement parce que quand nous avons cliqué sur Publier, on a envoyé notre valeur avec son HTML, `sanitize_text_field()` a tout nettoyé, puis notre donnée a été sauvegardée et enfin la fenêtre de prévisualisation a été rafraîchie en affichant donc notre nouvelle valeur. Mais seule la zone de prévisualisation a été rafraîchie. Pas l'outil de personnalisation, ni la valeur des champs.

Note : `sanitize_text_field()` n'est qu'une des fonctions fournies par WordPress pour nous aider.  
Voici la liste : <https://developer.wordpress.org/?s=sanitize>

## Validation

Tout ça s'est effectué sans feedback côté utilisateur. Entrer une valeur et avoir quelque chose d'autre sur le devant du site peut être déroutant ! C'est pourquoi parfois une étape de validation peut être nécessaire.

```
// in callback hooked on customize_register
$wp_customize->add_setting( 'wpcookbook-copyright-text', array(
    'validate_callback' => 'wpcookbook_validate_footer_text',
    'sanitize_callback' => 'sanitize_text_field',
    'sanitize_js_callback' => '',
) );
```

```
// in functions.php
/**
 * Validates our footer text setting
 *
 * @param WP_Error      $validity  Empty WP_Error initially. If there are problems,
use `add()` method to add error messages.
 * @param mixed          $value     The value to validate.
 * @param WP_Customize_Setting $setting  The corresponding setting.
 */
function wpcookbook_validate_footer_text( $validity, $value, $setting ) {
    if( sanitize_text_field( $value ) !== $value ){
        $validity->add( 'invalid', __( 'Invalid string. Please do not try to use HTML or
emoji.', 'twentynineteen-child' ) );
    }
    return $validity;
}
```

Chaque fonction personnalisée utilisée pour valider nos réglages reçoit trois paramètres.

- `$validity` est une instance vide de `WP_Error`. Il faut absolument la retourner. Si elle reste vide, alors cela signifie que tout est ok, et notre réglage est valide.
- `$value` est la donnée à valider.
- `$setting` est l'instance de `WP_Customize_Setting` correspondante à notre réglage. On pourrait en avoir besoin pour vérifier l'identifiant ou d'autres éléments comme le type ou les capacités requises.

Ici, pour faire simple, on vérifie que la version saine de notre valeur est identique à la valeur soumise. Si l'utilisateur n'écrit que du texte dans le champ, pas de souci. Dès qu'il va utiliser un caractère potentiellement dangereux que `sanitize_text_field()` est censé nettoyer, alors les deux chaînes seront différentes et on ajoute un message d'erreur à notre `$validity` car le petit roublard a sûrement voulu insérer du code HTML.

Publier

Personnalisation > Theme options

Footer settings

Adjust your footer preferences here

**Footer copyright text**

Invalid string. Please do not try to use HTML.

This is the text to display in the footer instead of the theme's credit line.

Copyright 2019 WPCookBook<script><

Footer logo

Retirer Changer l'image

Date

11-Nov 18 2019

Masquer les contrôles

Pas le droit au HTML dans le champ.

Cette opération bloque la sauvegarde et renvoie un message. Remarquez que notre valeur en base est inchangée et qu'il n'y a pas eu de rafraîchissement de la fenêtre de prévisualisation. En coulisse, un appel AJAX est fait pour valider le réglage via notre fonction de rappel, et si l'instance `WP_Error` renvoyée est non-vide, alors on affiche son message et on annule la sauvegarde.

## Autorisons un peu de code HTML

WordPress met à notre disposition une famille de fonctions utiles qui permettent de nettoyer du contenu tout en permettant une certaine flexibilité : les fonctions `wp_kses()`. Pour l'histoire, `kses` est un acronyme récursif signifiant KSES Strips Evil Scripts.

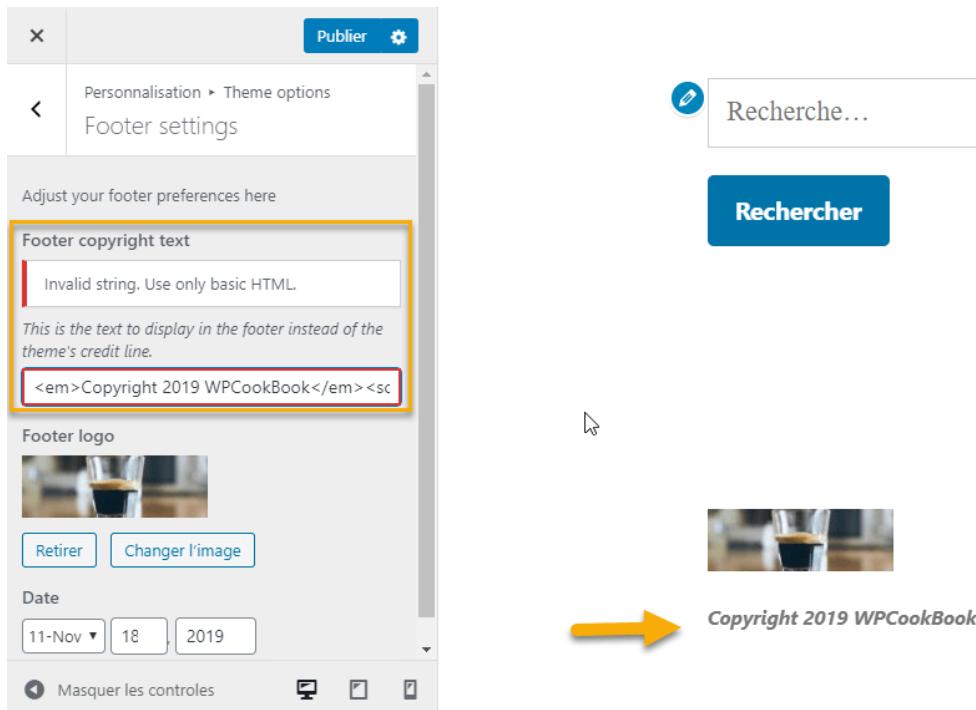
```
wp_kses( string $string, array[]|string $allowed_html, string[]
$allowed_protocols = array() ) est une fonction super utile qui prend une chaîne de caractères
$string et qui la nettoie en autorisant uniquement les balises et attributs passés dans $allowed_html
et les protocoles autorisés $allowed_protocols.
```

Vu qu'entrer toutes les balises et attributs HTML peut être un peu pénible et qu'on peut rapidement en oublier plusieurs, on va utiliser `wp_kses_post()`, qui est une fonction wrapper pour `wp_kses()` qui autorise toutes les balises HTML simples que l'on peut trouver dans un article ou page. Aussi, on va ajuster notre fonction de validation.

```
$wp_customize->add_setting( 'wpcookbook-copyright-text', array(
    'validate_callback' => 'wpcookbook_validate_footer_text',
    'sanitize_callback' => 'wp_kses_post',
    'sanitize_js_callback' => '',
) );
```

```
// in functions.php
function wpcookbook_validate_footer_text( $validity , $value , $setting ){
    if( wp_kses_post( $value ) !== $value ){
        $validity->add( 'invalid', __( 'Invalid string. Use only basic HTML.', 'twentynineteen-child' ) );
    }
    return $validity;
}
```

Mais pour afficher notre champ correctement, il faut ne plus échapper la valeur de notre réglage dans notre fichier footer.php. Notre champ accepte maintenant les balises `<em>`, `<strong>`, `<a>` et leurs attributs, mais pas les `<script>` qui sont nettoyées par `wp_kses_post()`.



On a droit à plus de code HTML. Notez que notre texte s'est mis à jour quand j'ai mis les balises `<em>`, mais n'accepte pas les `<script>`.

Il y a toute une série de fonctions `wp_kses()` très pratiques. Je vous laisse les découvrir dans la documentation. Elles sont toutes plus ou moins des raccourcis utilisant `wp_kses()` :

[https://developer.wordpress.org/?s=wp\\_kses](https://developer.wordpress.org/?s=wp_kses)

## Utiliser des fonctions personnalisées pour nettoyer

On peut aussi utiliser nos fonctions personnalisées pour nettoyer. Il faut juste savoir que ces fonctions reçoivent la valeur à nettoyer, mais aussi l'instance de `WP_Customize_Setting` correspondant à notre réglage.

```
$wp_customize->add_setting( 'wpcookbook-copyright-text', array(
    'validate_callback' => 'wpcookbook_validate_footer_text',
    'sanitize_callback' => 'wp_cookbook_sanitize_footer_text',
    'sanitize_js_callback' => '',
) );
```

```
// in functions.php
/**
 * Sanitizes our footer text setting
 * Now our field only accepts <em> and <a> tags with href attributes
 *
 * @param mixed $value The value to sanitize.
 * @param WP_Customize_Setting $setting The corresponding setting.
 */
function wpcookbook_sanitize_footer_text( $value, $setting ) {
    return wp_kses( $value, array(
        'em' => array(),
        'a' => array(
            'href' => array(),
        ),
    ) );
}

/**
 * Validates our footer text setting
 *
 * @param WP_Error $validity Empty WP_Error initially. If there are problems,
 * use `add()` method to add error messages.
 * @param mixed $value The value to validate.
 * @param WP_Customize_Setting $setting The corresponding setting.
 */
function wpcookbook_validate_footer_text( $validity, $value, $setting ) {
    if( wpcookbook_sanitize_footer_text( $value, $setting ) !== $value ){
        $validity->add( 'invalid', esc_html__( 'Invalid string. You are only allowed to use <em> and <a> tags.', 'twentynineteen-child' ) );
    }
    return $validity;
}
```

Ici, notre fonction de nettoyage ne va autoriser que les balises `<em>` sans attribut ainsi que les balises `<a>` avec leur attribut `href` uniquement. Notre fonction de validation sert ici juste à empêcher l'utilisateur de mettre autre chose.

Publier

Personnalisation > Theme options  
Footer settings

Adjust your footer preferences here

**Footer copyright text**

Invalid string. You are only allowed to use <em> and <a> tags.

This is the text to display in the footer instead of the theme's credit line.

<em>Copyright 2019 <a class="" href="https://vincentdubroeucq.com/wpcookbook">

**Footer logo**



**Retirer** **Changer l'image**

**Date**

11-Nov 18 2019

**Masquer les contrôles**

<https://vincentdubroeucq.com/wpcookbook>

Recherche... **Rechercher**

Copyright 2019 **WPCookBook**

On a le droit aux liens, mais pas aux classes CSS.

## sanitize\_js\_callback

Je suis désolé, mais on ne parlera pas de ce paramètre. Il permet de spécifier une fonction de rappel qui sera utilisée pour nettoyer notre valeur pour qu'elle soit compréhensible par le JavaScript de l'outil de personnalisation. Par défaut, le réglage est encodé proprement pour du JSON, donc il n'y a pas besoin d'intervenir dans 99% des cas. Le 1% restant, c'est quand vous devrez traiter votre valeur parce que vous avez créé un type de réglage particulier (il est possible de créer ses propres types de réglages), avec un contrôle personnalisé (possible aussi).

Créer des types de réglages et des nouveaux contrôles sort de l'étendue des sujets traités par ce guide. Mais qui sait ? Un autre ebook, peut-être ?

## Enregistrer des partiels

Notre champ texte est blindé. On procède à une validation pour empêcher l'utilisateur de mettre des balises qui vont être supprimées par notre fonction de nettoyage. Et l'utilisateur est averti s'il utilise des attributs ou balises non autorisées. C'est cool.

Mais le rafraîchissement de la page de prévisualisation à chaque fois que notre valeur change, c'est un peu pénible et lent, non ?

Pour éviter de rafraîchir TOUTE la page, on peut rafraîchir uniquement la partie du modèle affectée par notre réglage. Pour cela, on va enregistrer un template partial.

Dans notre fonction hookée sur `customize_register`, encore et toujours, ajoutez ceci :

```
// in functions.php
// Add a partial for our footer copyright text setting
$wp_customize->selective_refresh->add_partial( 'wpcookbook-copyright-text', array(
    'selector'          => '.site-copyright',
    'container_inclusive' => true, // Default false
    'render_callback'    => 'wpcookbook_site_copyright',
    'fallback_refresh'   => true // Default
) );
```

On appelle la méthode `add_partial()` de l'instance de la classe `WP_Selective_Refresh()` qui est contenue dans la propriété `selective_refresh` de l'instance de notre `WP_Customize_Manager`, `$wp_customize`. Ouf, des objets dans des objets, donc.

Bref, la méthode prend en premier paramètre un identifiant (qui doit correspondre au réglage auquel on veut associer notre partielle) et un tableau d'arguments. Voici uniquement les arguments les plus utiles :

- `selector` est le sélecteur CSS qui va être utilisé par jQuery pour trouver et rafraîchir notre bloc partielle `copyright`. C'est le conteneur à cibler.
- `render_callback` est la fonction de rappel à utiliser pour afficher notre bloc partielle.
- `fallback_refresh` est un booléen qui détermine si oui ou non on rafraîchit toute la page en cas de pépin si le rafraîchissement partielle échoue et que la fonction de rappel renvoie `false`. On va laisser sur `true`, c'est mieux.
- `container_inclusive` indique si oui ou non le conteneur ciblé est inclus dans la fonction de rappel, ou si la fonction n'affiche que son contenu. Si vous avez un souci d'imbrication multiple de votre conteneur, changez ce paramètre. J'ai mis `true` car je veux rafraîchir tout le conteneur `.site-copyright`.

On a passé une fonction de rappel `wpcookbook_site_copyright()`. Il faut donc la créer dans notre fichier `functions.php`, et l'utiliser dans notre template `footer.php`.

```
// In functions.php
/**
 * Displays the custom footer text.
 * Can be used as a partial for the customizer.
 */
function wpcookbook_site_copyright(){
    echo '<p class="site-copyright">';
    if( $footer_text = get_theme_mod( 'wpcookbook-copyright-text' ) ) {
        echo '<strong>' . $footer_text . '</strong>';
    } else {
        $blog_info = get_bloginfo( 'name' );
        if ( ! empty( $blog_info ) ) : ?>
            <a class="site-name" href=<?php echo esc_url( home_url( '/' ) ); ?>" rel="home">
<?php bloginfo( 'name' ); ?></a>,
            <?php endif; ?>
            <a href=<?php echo esc_url( __( 'https://wordpress.org/' , 'twentynineteen' ) ); ?>" class="imprint">
                <?php
                    /* translators: %s: WordPress. */
                    printf( __( 'Proudly powered by %s.', 'twentynineteen' ), 'WordPress' );
                ?>
            </a>
        <?php
        echo '</p>';
    }
}
```

Notre template tag `wpcookbook_site_copyright()` affiche soit la valeur de notre réglage soit le texte par défaut du thème. Du coup, on peut grandement simplifier le fichier `footer.php`.

```
// in footer.php
...
<div class="site-info">
    <?php
        if( $logo = (int) get_theme_mod( 'wpcookbook-footer-logo' ) ){
            ...
        }

        wpcookbook_site_copyright();

        if ( function_exists( 'the_privacy_policy_link' ) ) {
            the_privacy_policy_link( '' , '<span role="separator" aria-hidden="true">
</span>' );
        }
    ?>
```

Aussi, pour que tout ça fonctionne, il faut indiquer que notre réglage va utiliser du JavaScript pour gérer les rafraîchissements. Il faut ajuster notre appel à `$wp_customize->add_setting()`, en lui passant `'transport' => 'postMessage'` au lieu de `'refresh'`.

```
// Add our setting
$wp_customize->add_setting( 'wpcookbook-copyright-text', array(
    ...
    'transport'          => 'postMessage',
    'validate_callback' => 'wpcookbook_validate_footer_text',
    'sanitize_callback' => 'wpcookbook_sanitize_footer_text',
    'sanitize_js_callback' => '',
) );
```

Maintenant, l'outil de personnalisation rafraîchit uniquement le bloc <div class="site-copyright"> via JavaScript et affiche la valeur par défaut du thème si le réglage est vide. C'est beaucoup plus rapide, et notre validation fonctionne toujours !

C'est tout l'intérêt d'utiliser un maximum de templates tags personnalisés ! Vu que ce sont de simples fonctions, c'est très simple de les déclarer en partiels pour accélérer leur rafraîchissement dans la fenêtre de prévisualisation.

## Utiliser le JavaScript pour une prévisualisation instantanée

Mais on peut aussi directement affecter certaines valeurs ou certains attributs du DOM encore plus rapidement avec du JavaScript !

Reprendons l'exemple de notre champ Text Color. Il fonctionne, mais nécessite un rafraîchissement de la fenêtre de prévisualisation, pour inscrire une balise <style> dans l'entête de la page. C'est ok, mais on peut aussi faire que le changement soit instantané.

Pour ça, il faut passer 'transport' => 'postMessage' au lieu de 'refresh' à notre réglage et écrire nous-même le petit bout de JavaScript qui va gérer ça. Donc, on va ajuster la déclaration de notre réglage :

```
// Let's add another color setting, using WordPress' Color Control
$wp_customize->add_setting( 'wpcookbook-text-color', array(
    'sanitize_callback' => 'sanitize_hex_color',
    'transport'          => 'postMessage',
) );
```

Et on va charger un petit bout de JavaScript. Pour cela, on ne va pas utiliser le hook `wp_enqueue_scripts` ni `admin_enqueue_scripts` (gotcha !), mais le hook `customize_preview_init`. Dans `functions.php`, ajoutez :

```
add_action( 'customize_preview_init', 'wpcookbook_enqueue_customizer_js' );
/**
 * Enqueues our Customizer JavaScript handler
 */
function wpcookbook_enqueue_customizer_js() {
    wp_enqueue_script( 'wpcookbook_customizer', get_theme_file_uri( 'assets/js/customizer.js' ),
        array( 'customize-preview', 'jquery' ), null, true );
}
```

On va charger un script appelé `customizer.js`, placé dans le dossier `assets/js`. Ce script a pour dépendance `jQuery` et `customize-preview`.

Créons donc ce script :

```
(function ($) {
    wp.customize('wpcookbook-text-color', function (value) {
        value.bind(function (to) {
            $('body').css('color', to);
        });
    });
})(jQuery);
```

Ça a l'air compliqué comme ça, car il faut utiliser `bind()`. Honnêtement, ne vous inquiétez pas trop, car ce même snippet peut servir de boilerplate pour tous vos réglages. Ce qu'il suffit de comprendre, c'est que la valeur de notre réglage est contenu dans la variable `to`, et pas dans `value`.

Dans la fonction de rappel `jQuery`, on applique simplement la propriété CSS `color` à la balise `<body>` directement. Une capture d'écran ici ne servirait à rien pour vous montrer que ça marche. On peut changer notre couleur et le changement est reflété instantanément !

Vous pouvez aussi ajouter ou supprimer des classes en fonction de la valeur des réglages très simplement en utilisant `jQuery`. Par exemple, ajouter une classe `sidebar-right` ou `sidebar-left` correspondant à un réglage de positionnement de la colonne latérale est très simple.

## Ce qu'il faut retenir

- WordPress fournit des contrôles avancés que l'on peut utiliser, y compris pour les téléversements de fichiers, ce qui est bien pratique. De plus, ils ne sont pas plus difficiles à déclarer que les autres !
- Il faut **absolument** utiliser une fonction de nettoyage même si ce n'est qu'une simple `sanitize_text_field()` ou `absint()`.
- Utiliser une fonction de validation permet d'améliorer l'expérience utilisateur. C'est recommandé, mais pas nécessaire.
- Il faut échapper la donnée sur le devant du site.
- On peut utiliser des fonctions de validation et de nettoyage personnalisées.
- On peut déclarer des partiels pour éviter de rafraîchir toute la page à chaque changement de valeur de nos réglages. Utiliser au maximum des templates tags simplifie grandement ce procédé.
- On peut aussi utiliser du JavaScript pour appliquer nos changements directement dans la fenêtre de prévisualisation. Il faut juste écrire nous-même ce script. Cela est recommandé pour les changement simples : classes CSS à ajouter ou supprimer, ou propriétés CSS à appliquer directement.

Par contre, attention ! Avant de foncer dans le JavaScript pour améliorer la prévisualisation, assurez-vous que tous vos réglages fonctionnent avec `transport => refresh` !

Ajouter des partiels et faire les prévisualisations en live avec `jQuery` n'est qu'une amélioration de l'expérience utilisateur. Il faut que quand la page se charge normalement, votre PHP soit en mesure de gérer vos réglages et d'ajuster votre affichage comme il se doit. C'est pourquoi il faut absolument garder notre fonction `wpcookbook_text_color` qui affiche la balise `<style>` pour modifier notre couleur de texte. Si on la supprime, notre réglage va fonctionner uniquement dans l'outil de personnalisation !

Donc il faut travailler dans l'ordre suivant :

- On met en place le réglage de façon très basique, en mode refresh, dans l'outil de personnalisation et dans notre thème.
- On met en place les fonctions de nettoyage et validation. **Le nettoyage est très important.**
- Si possible, on enregistre le partiel ou on crée le script qui pour améliorer l'UX.

C'était un très long chapitre. Maintenant, vous êtes armés pour créer tout type de réglages, les traiter de façon sécurisée, en fournissant la meilleure expérience utilisateur possible.

Honnêtement, ce n'est pas rien ! Faites une pause, vous l'avez bien méritée !

# Comment rendre son thème compatible pour le nouvel éditeur

Le nouvel éditeur de WordPress est dans le cœur depuis la version 5.0. C'est une avancée majeure dans l'évolution de WordPress, dans la façon dont on va concevoir et rédiger notre contenu, mais c'est aussi une petite révolution pour les concepteurs de thèmes.

On a beaucoup de nouvelles fonctionnalités et blocs disponibles, ça c'est cool. Mais en tant que développeur, il faut coder le support pour toutes ces fonctionnalités. Et ça c'est un travail supplémentaire non négligeable !

## Les blocs sur le devant du site

Le nouvel éditeur fonctionne par blocs. Bloc Paragraphe, bloc Image, bloc Média + Texte, etc. C'est cool, ça permet de structurer notre contenu et chaque bloc a ses propres fonctionnalités, donc c'est très simple d'ajouter notre contenu et de régler son affichage et ses options comme on le souhaite.

Mais ces blocs génèrent chacun un balisage HTML qui leur est propre sur le front avec leurs propres classes CSS. Par exemple, là où l'éditeur classique WYSIWYG insérait une bête balise `<img>`, nous avons ceci pour un bloc `Image`:

```
<figure class="wp-block-image size-large">
    
    <figcaption>...</figcaption>
</figure>
```

Et ça, c'est pour une simple image sans alignement. Si on ajoute un alignement (centré par exemple), le balisage change :

```
<div class="wp-block-image">
    <figure class="aligncenter size-large">
        
        <figcaption>...</figcaption>
    </figure>
</div>
```

On a une `<div>` supplémentaire, et en plus, la classe `wp-block-image` a bougé sur cette `<div>`. Ce n'est pas forcément intuitif ou simple. On peut facilement se faire piéger !

Autre exemple, un bloc Bouton standard:

```
<div class="wp-block-button">
    <a class="wp-block-button__link">...</a>
</div>
```

Et un autre bouton aligné à droite.

```
<div class="wp-block-button align-right">
    <a class="wp-block-button__link">...</a>
</div>
```

Ici, on a moins de surprise. On a juste une classe qui s'ajoute. Ouf. Mais le bouton n'est pas un `<button>`, mais un lien avec une classe, ce qui peut se comprendre. Aussi, le lien est dans une `<div>` en `display: block` et pas dans un élément qui aurait `display: inline-block`. C'est encore une fois compréhensible, mais bon. Un `<button>` standard est censé être en `display: inline-block`.

Aussi, si on compare avec le bloc image, la classe `.alignright` n'est pas située au même niveau dans le code HTML du bloc. Cela peut poser souci ou pas, mais il faut être prudent. Le balisage peut varier énormément d'un bloc à l'autre, et pour un même bloc, d'un réglage à l'autre.

Dans votre feuille de styles, cela signifie que simplement inclure des styles pour les classes `.alignleft`, `.alignright` et `.aligncenter` n'est pas suffisant. Il y aura probablement des cas particuliers et styles descendants à prévoir selon les blocs.

Le temps où il suffisait de styler les éléments HTML simples parce que c'est ce que l'éditeur de contenu générait en front est révolu ! En plus de fournir des styles pour les éléments HTML simples, il faudra aussi styler TOUS les blocs du nouvel éditeur et TOUTES les variations de chaque bloc.

Des styles par défaut sont disponibles et vous pouvez vous en servir comme point de départ. Mais vous pouvez aussi redéfinir tous les styles pour tous les blocs selon vos ambitions. Ayez juste conscience que tous les blocs doivent s'afficher correctement, surtout si vous comptez publier votre thème sur le répertoire officiel ou le distribuer de quelque autre manière.

Bon, maintenant que vous avez une vue d'ensemble du travail supplémentaire, remontons nos manches, et c'est parti.

## Ce qui fonctionne bien par défaut

Pour l'exemple, on va travailler sur un thème qui ne déclare aucun support pour le nouvel éditeur. Par ~~flemme~~ chance, il se trouve que mon propre thème Flex-lite n'a aucun aménagement pour Gutenberg. C'est cool, non ? Ahem...

# Flex-Lite

By [Vincent Dubroeucq](#)



**Flex**  
A simple and clean theme for your personal blog

Blog About Products Resources Contact

**Flex**  
A simple and clean theme for your personal blog

[Twitter](#) [Facebook](#) [Google+](#) [Digg](#) [Email](#)

**Welcome to Flex**  
By [Vincent](#) on [July 2016](#). Posted in [Uncategorized](#).

Woohoo! I'm proud to announce my new theme is outing the world! Welcome to flex!

[Preview](#)

[Download](#)

Version: 1.6

Last updated: December 21, 2017

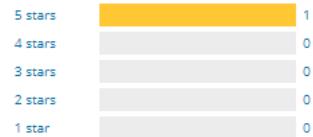
Active Installations: 200+

[Theme Homepage →](#)

## Ratings

[See all >](#)

5 out of 5 stars.



[Add my review](#)

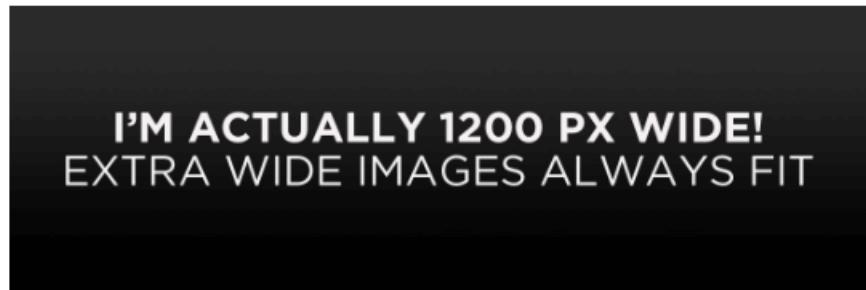
A simple and clean responsive theme for your blog.

*Pas mis à jour depuis 2017...*

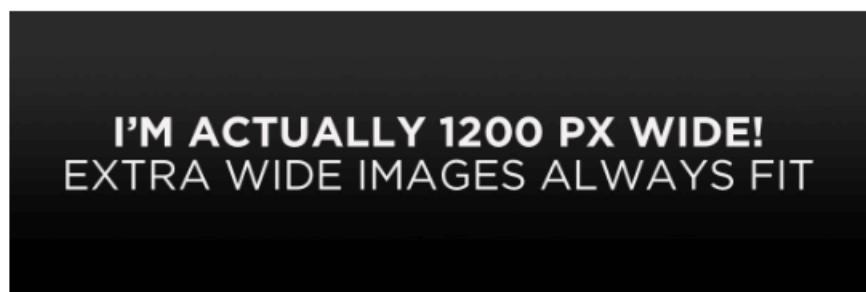
Pour admirer l'étendue des dégâts, on va installer la batterie d'articles tests proposés par la Theme Review Team de WordPress.org, que vous pouvez trouver ici : <https://github.com/WPTRT/theme-unit-test>

Il suffit d'utiliser l'importateur de WordPress, et d'importer le fichier .xml fourni, sans oublier de cocher la case permettant de télécharger toutes les images. Cela va installer des dizaines d'articles, de pages et de menus pour tester tous les usages courants de WordPress. Mais ce qui nous intéresse pour le moment, ce sont les blocs du nouvel éditeur.

Imagine that we would find a use for the extra wide image! This image has the *wide width* alignment:



Can we go bigger? This image has the *full width* alignment:



And that's a wrap, yo! You survived the tumultuous waters of alignment. Image alignment achievement unlocked! One last thing: The last item in this post's content is a thumbnail floated right. Make sure any elements after the content are clearing properly.

Les nouveaux alignements ne sont pas pris en compte

Comme vous pouvez le voir sur la page du blog, les nouveaux alignements ne sont pas pris en compte, les blocs Bouton ne sont pas accordés avec les boutons standards, le texte des blocs Bannière est à revoir, les styles des blocs Citation aussi, etc... Et quand on ouvre le bloc Citation et que l'on va dans l'administration, on peut voir que l'apparence des blocs dans l'administration et sur le devant du site n'a rien à voir ! Bref, il y a du travail.

Par contre, certaines choses fonctionnent. Les blocs "Colonnes" fonctionnent, les galeries aussi (à peu près), etc. Il y a des styles à revoir, certes. Mais au niveau purement fonctionnel, ça marche out of the box.

Si certains blocs fonctionnent c'est parce que le nouvel éditeur inclus des feuilles de styles de base pour les blocs, qui permettent d'avoir quelque chose de fonctionnel sans trop toucher à votre thème, et ça c'est plutôt une bonne nouvelle. Donc on n'a pas forcément besoin de fournir des styles pour TOUTES les classes CSS que l'éditeur injecte dans nos blocs sur le devant du site. Cela dit, un travail sur le CSS est quand même nécessaire.

Donc, on va rapidement créer un thème enfant pour faire nos ajustements, car on a du pain sur la planche ! Je vous renvoie au chapitre Comment créer un thème enfant pour ce faire.

## Ajouter le support pour les styles supplémentaires

Certains blocs ont des styles supplémentaires cosmétiques. Reprenons l'exemple du bloc Citation. Naviguez vers l'article correspondant, et ouvrez-le aussi dans l'éditeur dans un nouvel onglet. Dans l'éditeur, pour cet article, on obtient ceci :

The screenshot shows the WordPress editor interface. On the left, the sidebar has 'Articles' selected. The main area displays the title 'Block: Quote'. A quote block is inserted, containing the text 'Gutenberg is more than an editor.' attributed to 'The Gutenberg Team'. The quote block has a black border on the left. To the right, the 'Bloc' panel is open, showing the 'Citation' style definition: 'Donnez une emphase visuelle à vos citations. « En citant les autres, nous nous citons nous mêmes » — Julio Cortázar'. The 'Styles' section shows two options: 'Par défaut' (selected) and 'Grand'. A yellow arrow points from the text 'and large:' to the 'Grand' style option in the panel.

On a une bordure noire sur la gauche des blocs Citation

Vous pouvez voir qu'on a une bordure noire sur la gauche des blocs Citation, mais que cette bordure n'est pas présente sur le devant du site. Elle est remplacée par une bordure orange qui est celle par défaut ajoutée aux éléments <blockquote> par le thème.

The quote block has two styles, regular:

Gutenberg is more than an editor.

*The Gutenberg Team*

and large:

Le thème utilise par défaut une bordure orange pour les blocs Citation

Ces styles sont chargés dans l'éditeur, mais pas sur le devant du site. Donc si vous voulez ajouter ces styles sur le devant de votre site, le thème doit déclarer le support pour cette fonctionnalité. Ceci se fait à l'aide de la fonction `add_theme_support()`.

```
// In functions.php
add_action( 'after_setup_theme', 'flex_lite_child_setup' );
/**
 * Adds support for various features of the new editor
 */
function flex_lite_child_setup(){
    // Add support for additional block styles
    add_theme_support( 'wp-block-styles' );
}
```

On se hooke sur `after_setup_theme` pour déclarer la plupart les fonctionnalités du thème: logo, image d'en-tête personnalisé, image d'arrière-plan, etc. Le nouvel éditeur ne fait pas exception, et on va plus tard ajouter plusieurs appels à `add_theme_support()` dans notre fonction de rappel.

`add_theme_support( string $feature, mixed $args )` prend deux paramètres: une chaîne de caractères `$feature` qui est le nom de la fonctionnalité à ajouter, et un tableau d'arguments optionnel. Selon la fonctionnalité, vous aurez besoin ou non d'ajouter ces arguments. Pour `wp-block-styles`, pas besoin d'arguments.

Maintenant, si on revient sur le devant de notre site, vous pouvez voir qu'on a la même bordure noire que dans l'éditeur. Vous pourrez aussi remarquer que le texte de la source est plus petit.

The quote block has two styles, regular:

Gutenberg is more than an editor.

The Gutenberg Team

and large:



Bloc Citation avec les styles de l'éditeur

On ne va pas passer en revue tous les blocs possibles et leurs variations avec ou sans les styles supplémentaires. Sachez juste que ces styles supplémentaires sont chargés en administration, mais pas sur le devant du site par défaut. Si vous les voulez sur le devant du site, il faut ajouter l'appel à `add_theme_support()` correspondant. Mais c'est à vous de décider si vous voulez ajouter ces styles ou non, ou si vous préférez simplement ajouter vos propres styles pour les blocs.

## Supprimer tous les styles ajoutés par le nouvel éditeur

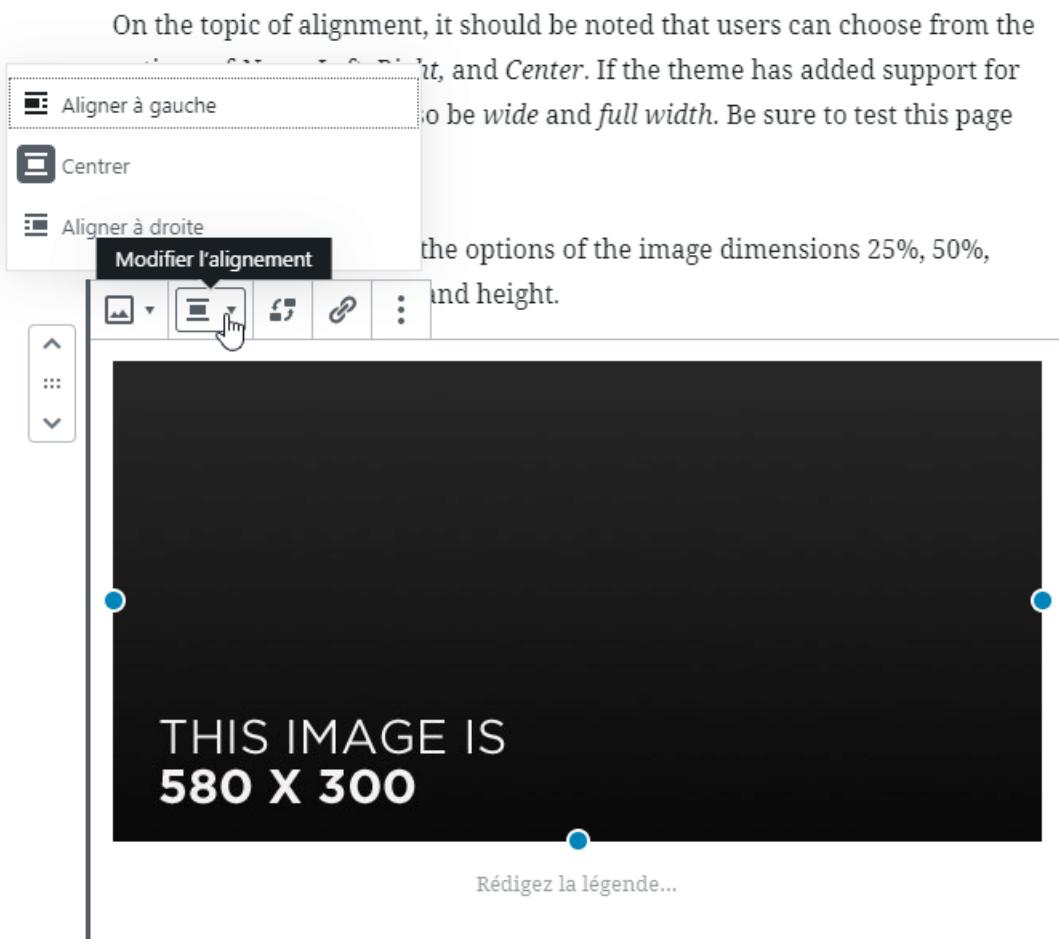
Si vous êtes courageux, vous pouvez aussi désactiver tous les styles de Gutenberg sur le devant du site. Dans la fonction chargeant vos styles dans votre thème enfant, un simple appel à `wp_dequeue_style()` suffit :

```
// in functions.php
add_action( 'wp_enqueue_scripts', 'flex_lite_child_scripts' );
/**
 * Loads parent styles
 */
function flex_lite_child_scripts(){
    wp_enqueue_style( 'flex-lite-styles', get_parent_theme_file_uri( 'style.css' ) );
    wp_dequeue_style( 'wp-block-library' );
}
```

Là, vous vous donnez une charge de travail plus importante, mais vous avez le contrôle sur tout. Attention cependant car vous devrez surveiller les mises à jour de WordPress et de l'éditeur de plus près pour maintenir votre thème à jour.

## Ajouter le support pour les alignements Grande largeur et Pleine largeur

Certains blocs, comme le bloc `Image` donnent la possibilité de contrôler leur alignement. Ouvrez l'article `Block: Image`, sur le devant et dans l'éditeur.



The image above happens to be **centered**.

Par défaut, trois options d'alignement sont disponibles

Par défaut, vous pouvez aligner une image à droite, à gauche, la centrer ou ne pas donner d'alignement. Si le thème supporte la fonctionnalité, l'éditeur peut aussi mettre à notre disposition deux options supplémentaires : Grande largeur et Pleine largeur. Il suffit d'une ligne dans `functions.php` :

```
// in functions.php
function flex_lite_child_setup(){
    ...
    // Add support for wide and full alignments
    add_theme_support( 'align-wide' );
}
```

Ajouter cette ligne rend les options d'alignement disponibles dans l'éditeur, et sur le devant du site les blocs concernés ont une nouvelle classe CSS `alignwide` ou `alignfull`. Ce qui signifie qu'il faut ajouter le CSS nécessaire dans votre thème pour que cela fonctionne correctement. Cela va fortement dépendre du thème.

Parfois, un snippet comme celui-ci peut suffire :

```
/* in style.css */
.alignwide,
.alignfull {
    width: auto;
    max-width: 100vw;
}

.alignwide img,
.alignfull img {
    object-fit: cover;
    width: 100%;
}

.alignwide {
    margin-left: calc(25% - 25vw);
    margin-right: calc(25% - 25vw);
}

.alignfull {
    margin-left: calc(50% - 50vw);
    margin-right: calc(50% - 50vw);
}
```

The screenshot shows a WordPress dashboard with a dark theme. At the top, there's a header bar with the WPCookBook logo, user info, and navigation links like 'Personnaliser', 'Créer', and 'Modifier l'article'. Below the header, there's a message: 'Imagine that we would find a use for the extra wide image! This image has the *wide width* alignment:'. Below this message is a large black rectangular image containing the text 'I'M ACTUALLY 1200 PX WIDE!' and 'EXTRA WIDE IMAGES ALWAYS FIT' in white. Underneath this image, there's a small note: 'Can we go bigger? This image has the *full width* alignment:'. Below this note is another large black rectangular image with the same text 'I'M ACTUALLY 1200 PX WIDE!' in white. At the bottom of the dashboard, there's a footer message: 'Les nouveaux alignements fonctionnent'.

Cela va fonctionner pour le bloc Image, mais attention si vous utilisez une colonne latérale ! Ce snippet ne fonctionnera plus comme espéré ! C'est à vous de gérer tous ces cas particuliers selon votre thème. Par exemple, vous pouvez ajouter un filtre sur `body_class` qui vient ajouter une classe CSS `no-sidebar` si la colonne latérale est vide de widgets, et ajuster votre css ainsi :

```

/* in style.css */
.no-sidebar .alignwide {
    margin-left: calc(25% - 25vw);
    margin-right: calc(25% - 25vw);
}

.no-sidebar .alignfull {
    margin-left: calc(50% - 50vw);
    margin-right: calc(50% - 50vw);
}

```

Pour ce thème, le contenu est affiché en une seule colonne si la colonne latérale est vide de widgets. Donc cela fonctionnera.

Encore une fois, tout dépend de votre thème. Activer les alignements est très simple en PHP, mais nécessite plusieurs ajustements de CSS pour prendre en compte tous les cas possibles.

## Ajouter le support pour les couleurs

Pour cet exemple, ouvrez l'article Block category: Common dans l'éditeur, et sur le devant du site.

L'éditeur offre la possibilité de faire en sorte que les couleurs de votre thème soient utilisables dans l'éditeur. On déclare nos couleurs simplement avec `add_theme_support()`. Toujours, dans notre fonction de rappel hookée sur `after_setup_theme`, ajoutez :

```

// in functions.php
// Add support for theme colors
add_theme_support( 'editor-color-palette', array(
    array(
        'name'  => __( 'Flex blue', 'flex-lite-child' ),
        'slug'  => 'flex-blue',
        'color' => '#0080ff',
    ),
    array(
        'name'  => __( 'Flex orange', 'flex-lite-child' ),
        'slug'  => 'flex-orange',
        'color' => '#FF7F00',
    ),
) );

```

Ici, on passe `editor-color-palette` à la fonction `add_theme_support()`, accompagné d'un tableau des couleurs à ajouter. `name` est le nom à afficher, `slug` l'identifiant, et `color` le code hexadécimal de la couleur. C'est très simple.

The screenshot shows the WordPress editor interface. On the left is a sidebar with navigation links: Articles, Tous les articles, Ajouter, Catégories, Étiquettes, Médias, Pages, Commentaires (3), Apparence, Extensions, Utilisateurs, Outils, Réglages, and Réduire le menu. The main area contains two paragraphs of text. The first paragraph has a flex-orange background and white text. The second paragraph has a white background and flex-blue text. A central toolbar includes icons for bold, italic, underline, and more. To the right, a panel titled "Régagements du texte" is open, showing "Régagements de couleur". It has two sections: "Couleur d'arrière-plan" (background color) and "Couleur du texte" (text color). Both sections show blue and orange color swatches with a "Couleur personnelle" button and an "Effacer" button. A note at the bottom of the panel says: "Cette combinaison de couleurs est difficilement lisible. Essayez d'utiliser une couleur d'arrière-plan et/ou de texte plus sombre." (This color combination is difficult to read. Try using a darker background color and/or a darker text color.)

Nos couleurs apparaissent dans l'éditeur.

Sur le devant du site, des classes CSS correspondantes à nos couleurs sont ajoutées à nos blocs, et il faut maintenant ajuster notre feuille de styles pour que cela fonctionne correctement, en ajoutant pour chaque couleur deux classes CSS : `has-${slug}-color` et `has-${slug}-background-color`. Remarquez aussi que dans l'éditeur, on a un peu de padding sur les paragraphes quand ils ont une couleur d'arrière-plan. Il faut aussi en ajouter sur le devant du site, en utilisant la classe `has-background`. Pour l'exemple, j'ai ajouté exactement le même padding que l'éditeur.

```
/* in style.css */
/* Support for theme colors */
.has-background {
    padding: 20px 30px;
}
.has-flex-blue-color {
    color: #0080ff;
}
.has-flex-blue-background-color {
    background-color: #0080ff;
}
.has-flex-orange-color {
    color: #FF7F00;
}
.has-flex-orange-background-color {
    background-color: #FF7F00;
}
```

This paragraph is colorful, with a flex-orange background and white text. Colored blocks should have a high enough contrast, so that the text is readable.



This paragraph is colorful, with a white background and flex-blue text. Colored blocks should have a high enough contrast, so that the text is readable.

Les couleurs fonctionnent sur le devant du site.

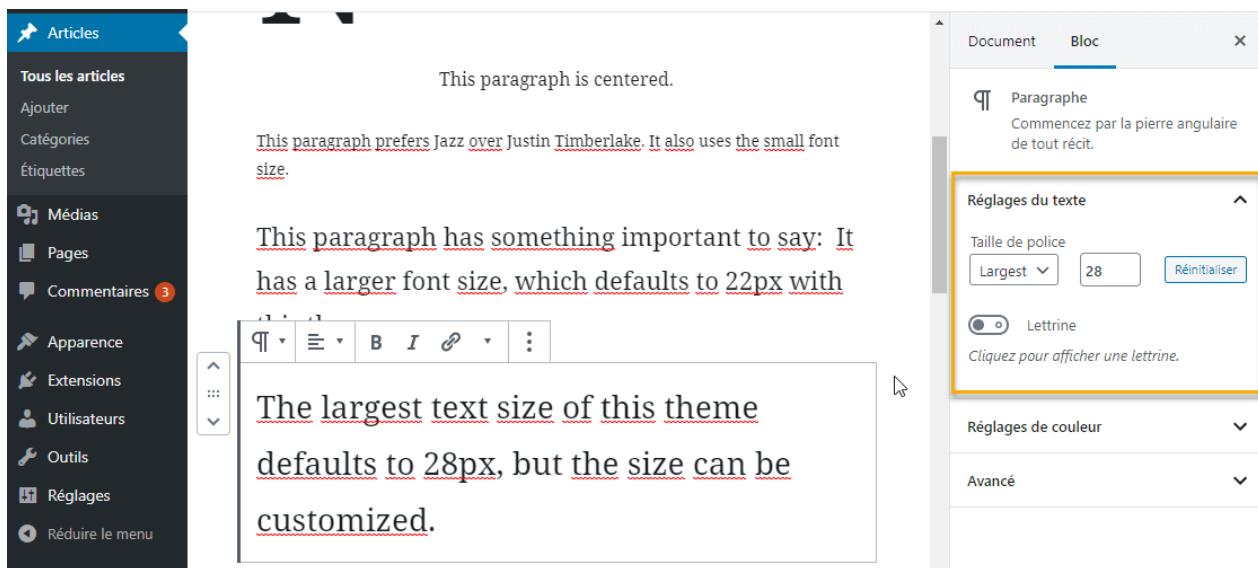
Si vous souhaitez ne pas laisser à l'utilisateur la possibilité de choisir une couleur personnalisée pour le texte ou l'arrière-plan, vous pouvez désactiver cette option :

```
// in functions.php, flex_lite_child_setup() function
// Remove custom color options
add_theme_support( 'disable-custom-colors' );
```

## Ajouter le support pour les tailles de texte

Comme pour les couleurs, on peut déclarer les tailles de texte utilisées par le thème, exactement de la même façon que les couleurs sont déclarées. On passe editor-font-sizes, et un tableau d'arguments à add\_theme\_support():

```
// in functions.php, flex_lite_child_setup() function
// Add support for theme font sizes
add_theme_support( 'editor-font-sizes', array(
    array(
        'name' => __( 'Small', 'flex-lite-child' ),
        'slug' => 'small',
        'size' => 14,
    ),
    array(
        'name' => __( 'Normal', 'flex-lite-child' ),
        'slug' => 'normal',
        'size' => 18,
    ),
    array(
        'name' => __( 'Larger', 'flex-lite-child' ),
        'slug' => 'larger',
        'size' => 22,
    ),
    array(
        'name' => __( 'Largest', 'flex-lite-child' ),
        'slug' => 'largest',
        'size' => 28,
    )
) );
```



Déclarer des tailles de texte est simple.

Le `slug` sera utilisé dans les classes CSS ajoutées sur les blocs sur le devant du site. Donc comme pour les couleurs, il faut ajouter quelques classes CSS pour gérer toutes vos tailles.

```
/* Support for theme font sizes */
.has-small-font-size {
    font-size: 14px;
}
.has-larger-font-size {
    font-size: 22px;
}
.has-largest-font-size {
    font-size: 28px;
}
```

Ici, les tailles utilisent des pixels, mais vous pouvez très bien utiliser `desem` ou autre, voire utiliser des `media-queries` pour rendre votre typographie responsive. Malheureusement, cela n'est possible que sur le devant du site. Le tableau de paramètres pour `add_theme_support()` attend une valeur numérique pour `size` et l'applique en pixels. Donc dans l'éditeur, vous aurez effectivement des tailles en pixels.

Comme pour les couleurs, vous pouvez aussi désactiver les tailles de polices personnalisées, en ajoutant la ligne suivante :

```
// in functions.php, in after_theme_setup callback
add_theme_support( 'disable-custom-font-sizes' );
```

## Ajouter le support pour les styles de l'éditeur

Vous avez sûrement remarqué que les styles dans l'éditeur sont très différents des styles sur le devant du site ? On a ajouté des fonctionnalités à l'éditeur et on les a reflété sur le devant du site, mais on n'a pas encore touché aux styles de l'éditeur.

Or le but de cet éditeur est d'améliorer l'expérience d'édition de contenu en faisant en sorte que les styles de l'éditeur soient le plus proche possible de ceux du devant du site. WYSIWYG (What You See Is What You Get), simplement.

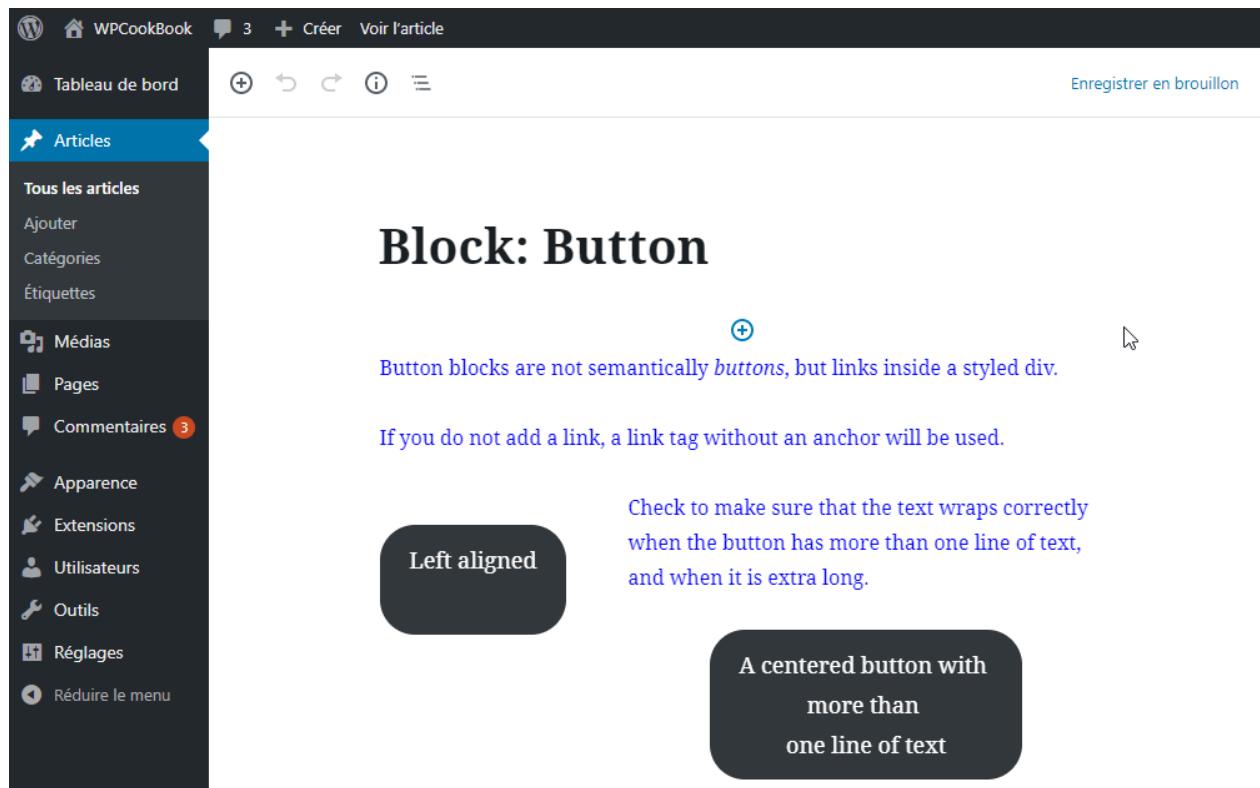
On doit donc ajouter une feuille de styles pour l'éditeur, pour harmoniser le frontend et le backend. Pour ça, on utilise encore `add_theme_support()`:

```
// in functions.php, flex_lite_child_setup() function
// Add support for editor styles
add_theme_support( 'editor-styles' );
add_editor_style();
```

La fonction `add_editor_style( array|string $stylesheet = 'editor-style.css' )` va enregistrer une feuille de styles pour la charger dans l'éditeur. On peut lui passer un chemin vers la feuille de styles en question, relatif à la racine du thème, ou un tableau de feuilles de styles. On peut aussi ne rien lui passer du tout et par défaut, elle ira chercher le fichier `editor-style.css` à la racine du thème.

Créons donc ce fichier `editor-style.css` :

```
// editor-style.css
body {
    color: blue;
}
```



Nice. En fait non.

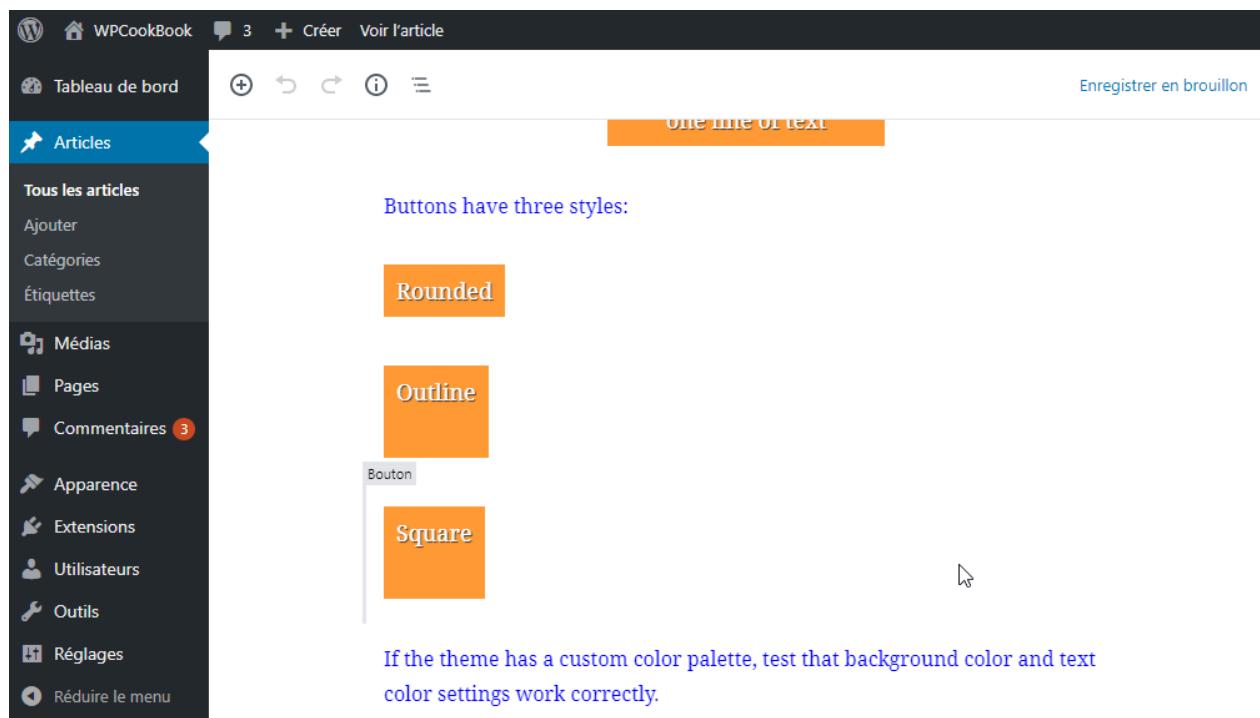
La fonction `add_editor_style()` est utilisée pour charger la feuille de styles de l'éditeur classique à l'origine. Pour le nouvel éditeur, les styles étant gérés d'une autre façon, il faut ajouter le support pour `editor-styles`.

Aussi, nos sélecteurs CSS sont transformés pour cibler `.editor-styles-wrapper` et ses enfants avant d'être injectés, ce qui est bien pratique car on a beaucoup moins besoin d'utiliser des sélecteurs très spécifiques pour cibler le contenu de l'éditeur.

Sur la page d'édition de l'article Block: Button, je peux cibler les blocs Bouton assez facilement :

```
/* in editor-style.css AND child theme style.css */
.wp-block-button__link {
    background-color: #ff9933;
    border-radius: 0;
    border: none;
    color: #fff;
    display: inline-block;
    padding: 5px 10px;
    margin: .3em 0;
    text-shadow: 1px 1px 1px #333;
    transition: all .3s;
}
```

Et je surcharge aussi les styles du bloc sur le devant du site dans mon thème enfant. Si vous créez votre propre thème, c'est plus simple, car pas besoin de surcharger ! Utilisez les classes `.wp-block-button` et `.wp-block-button__link` quand vous définissez les styles de vos boutons.



L'apparence des boutons est un peu plus cohérente par rapport au devant du site.

Maintenant, les boutons dans l'éditeur ressemblent bien plus aux boutons standards du devant du site. Mais il me reste à styler les états :hover, :focus, et les différents styles offerts par le bloc : arrondi, outline, etc.

Une fois que j'aurais fini le bloc Bouton et toutes ses options, je pourrais m'attaquer à un autre bloc, comme Citation par exemple, qui a aussi bien besoin d'attention.

Dans l'éditeur, on peut aussi modifier la largeur de chaque bloc pour mieux refléter la largeur de la zone de contenu sur le devant du site :

```
/* in editor-style.css */
/* Adjust main block width */
@media screen and (min-width: 1280px){
    .wp-block {
        max-width: 800px;
    }
}
/* Adjust wide blocks */
.wp-block[data-align="wide"] {
    max-width: 1080px;
}
/* Adjust fullwidth blocks */
.wp-block[data-align="full"] {
    max-width: none;
}
```

## Par où commencer ?

On a vu comment ajouter des fonctionnalités à l'éditeur et les intégrer dans votre thème, et vous avez pu voir qu'il y a du travail ! Rien que pour le bloc Bouton, il me reste un peu de travail. Pour les alignements, c'est le même constat. Il me reste pas mal de CSS à faire.

Cela peut rapidement devenir très compliqué à gérer, je vous conseille du pas-à-pas. Aussi, la complexité va dépendre de si vous travaillez sur un thème enfant ou si vous modifiez votre propre thème.

Personnellement, j'y vais bloc par bloc. C'est-à-dire que je vais ajuster mes styles en me concentrant d'abord sur le bloc Paragraphe, puis Image, Citation, etc.

Plus précisément :

- Assurez-vous que tous les éléments HTML basiques soient stylés <hn>, <p>, <img>, <figure>, etc.
- Passez en revue les styles de base ajoutés sur le devant du site, pour déterminer si vous allez les surcharger ou les réécrire. Essayez aussi les styles supplémentaires pour voir.
- Choisissez les fonctionnalités que vous allez ajouter. En général, align-wide est attendu, mais peut être difficile à gérer selon la mise en page de votre thème. Les couleurs et les tailles de police sont recommandées, mais pas forcément essentielles. Par contre, elles sont plus faciles à mettre en oeuvre que align-wide.
- Prenez un des articles de la batterie de tests, comme Block category: Common, et passez en revue les styles sur le devant du site pour un seul type de bloc au départ, par exemple le bloc Paragraphe. Ajustez tous vos styles pour prendre en compte toutes les options offertes par l'éditeur pour ce bloc, et ce sur le devant du site.
- Pour ce même bloc, ajustez ensuite les styles dans l'éditeur.

- Vérifier que les différents réglages du thème ne cassent pas tout. Je pense encore à align-wide. Vérifiez aussi la responsive.
- Répétez le même travail pour un autre bloc.

L'idée est en fait de travailler bloc par bloc, d'abord sur le devant du site, puis dans l'éditeur. Si les styles sur le devant du site sont ok, alors les reproduire dans l'éditeur n'est pas forcément compliqué. Oui c'est laborieux mais au moins, vous offrez à vos utilisateurs le maximum de flexibilité !

## Ce qu'il faut retenir

- Déclarer le support pour les différentes fonctionnalités de l'éditeur se fait avec `add_theme_support()`, hookée sur `after_setup_theme`. On lui passe l'identifiant de la fonctionnalité à déclarer et un tableau d'arguments si besoin.
- On peut activer des styles supplémentaires ou désactiver complètement les styles par défaut.
- Certaines fonctionnalités se gèrent de façon simple sur le devant du site, en ajoutant les styles pour des classes CSS que l'éditeur injecte dans notre contenu.
- Attention à la gestion des alignements. Le balisage des blocs pouvant changer en fonction des réglages de ce dernier, il faut être prudent.
- Ajouter des styles dans l'éditeur nécessite un appel à `add_theme_support()` et à `add_editor_style()`.
- Travaillez bloc par bloc. En stylant toutes les combinaisons d'options possibles pour chaque bloc. D'abord pour le devant du site, puis pour l'éditeur.

Bon courage ! Pour ma part, j'ai encore du travail à rattraper pour ce thème !

# Comment personnaliser les menus de votre thème WordPress

Le système de menu natif de WordPress est plutôt bien fait. En quelques clics, on peut créer un menu, lui donner des éléments et sous-éléments (et donc créer des sous-menus), l'affecter à une ou plusieurs zones de menu selon le besoin (et selon votre thème), et on peut même le personnaliser en donnant des titres ou des classes CSS personnalisées aux éléments. WordPress nous donne même la liste des éléments de menus que l'on peut ajouter. Il suffit de glisser-déposer. C'est plutôt pratique.

Mais pour certains besoins, ce n'est pas suffisant. Ajouter une icône ? Pas évident de base. Des classes CSS aux liens ? Par défaut, les classes CSS sont ajoutées à l'élément `<li>`, et pas `<a>`. La fonction `wp_nav_menu()` que l'on a étudiée dans le chapitre *Comment déclarer une zone de menu dans votre thème* offre plusieurs options de personnalisations bien utiles et c'est déjà une première solution.

## Ce qui est disponible de base.

Si on se rend dans la zone d'administration des menus, voilà ce qu'on obtient :

The screenshot shows the WordPress admin interface for managing menus. On the left, the sidebar is visible with the 'Apparence' (Appearance) menu selected. The main content area has a title 'Menus' with a sub-section 'Gérer avec la prévisualisation en direct' (Manage with direct preview). Below this are two tabs: 'Modifier les menus' (Edit menus) and 'Gérer les emplacements' (Manage locations). A note says 'Sélectionnez le menu à modifier : Custom Menu (Principal)' and ' Sélectionner' or 'créez un nouveau menu'. The 'Nom du menu' is set to 'Custom Menu' with a 'Enregistrer le menu' (Save menu) button. The 'Ajouter des éléments de menu' section on the left lists 'Pages', 'Articles', 'Liens personnalisés', and 'Catégories'. Under 'Pages', there's a list of recent pages: Connexion, Profil, Inscription, Sample Page, with a 'Tout sélectionner' (Select all) checkbox and an 'Ajouter au menu' (Add to menu) button. The 'Structure du menu' panel on the right shows a single item 'Sample Page' with fields for 'Titre de la navigation' (Navigation title) containing 'Sample Page', 'Attribut de titre' (Title attribute) containing 'Title attribute', and a checked 'Ouvrir le lien dans un nouvel onglet' (Open link in new tab) option. It also includes 'Classes CSS (facultatives)' (Optional CSS classes) with 'test' and 'Relation avec le propriétaire du site lié (XFN)' (Relationship with owner site XFN) with 'nofollow'. A 'Description' field contains 'This is the menu item description'. At the bottom, there's an 'Original' link and buttons for 'Retirer' (Remove) and 'Annuler' (Cancel).

Administration des menus

Sur cet écran, j'ai activé toutes les options disponibles pour personnaliser les menus. Si vous n'avez pas les mêmes options, allez faire un petit tour du côté des Options de l'écran en haut à droite, et cochez ce que vous voulez voir apparaître.

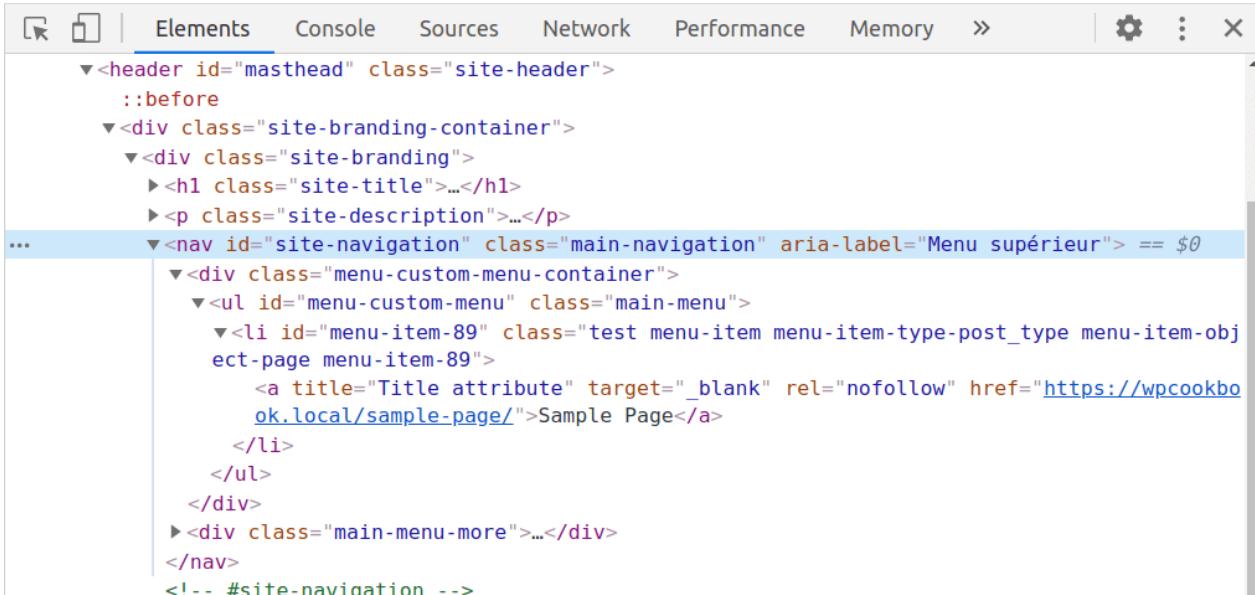
Vous pouvez aussi personnaliser vos menus dans l'outil de personnalisation de WordPress, et voir donc vos changements instantanément. Mais attention, car les Options de l'écran ne sont pas disponibles dans cet outil ! Ce qui signifie que si une des options proposées dans les Options de l'écran n'est pas cochée, elle n'apparaîtra pas dans l'Outil de personnalisation et vous ne pourrez pas la rajouter. Vous serez obligés de revenir dans l'administration.

Par défaut, vous pouvez :

- modifier le titre de l'élément de navigation,
- ajouter un attribut `title` au lien (qui sert à ajouter des précisions visibles au survol sur le lien ou si l'élément est caché visuellement),
- choisir que le lien s'ouvre dans un nouvel onglet (cela ajoute un attribut `target=_blank` sur l'élément `<a>`),
- ajouter une ou plusieurs classes CSS (qui vont s'ajouter sur l'élément `<li>`),
- ajouter un attribut `rel` au lien (champ Relation avec le propriétaire du site lié)
- ajouter une description pour l'élément du menu. Son affichage est complètement dépendant du thème. Beaucoup de thèmes ne l'utilisent pas, mais bon. On a un champ dédié si jamais on a besoin d'ajouter un peu de contenu.

C'est déjà pas mal !

En remplissant tous les champs, l'HTML généré sur le thème Twenty Nineteen ressemble à ceci :



```
<header id="masthead" class="site-header">
  ::before
  <div class="site-branding-container">
    <div class="site-branding">
      <h1 class="site-title">...</h1>
      <p class="site-description">...</p>
    <nav id="site-navigation" class="main-navigation" aria-label="Menu supérieur" style="display: flex; justify-content: space-between; align-items: center;">
      <div class="menu-custom-menu-container">
        <ul id="menu-custom-menu" class="main-menu">
          <li id="menu-item-89" class="test menu-item menu-item-type-post_type menu-item-object-page menu-item-89">
            <a title="Title attribute" target="_blank" rel="nofollow" href="https://wpcookbook.local/sample-page/">Sample Page</a>
          </li>
        </ul>
      </div>
      <div class="main-menu-more">...</div>
    </nav>
  <!-- #site-navigation -->
<!-- #masthead -->
```

L'HTML du menu sur le devant du site.

On a bien nos attributs `title`, `rel` et `target` sur notre lien, et notre classe CSS sur le `<li>` correspondant, mais comment faire si on veut ajouter des classes CSS dynamiques ? Ou des icônes ?

Alors, pour les icônes, on peut insérer du code HTML dans le champ Titre de la navigation. Donc techniquement, on peut y insérer directement notre `<svg>`, et ça marche ! Mais le champ étant tout petit, ça peut vite devenir compliqué de travailler l'HTML ici.

Mais vu qu'on a envie de faire ça proprement, pour la suite de l'article on va simplement oublier cette

solution.

## Modifier l'HTML global du menu avec `wp_nav_menu()`

On en a déjà parlé au chapitre Comment déclarer une zone de menu dans votre thème mais la fonction `wp_nav_menu()` peut prendre plusieurs paramètres pour nous permettre de personnaliser l'HTML du menu. On peut choisir :

- la balise HTML utilisée pour le conteneur,
- les classes CSS du conteneur,
- les classes CSS de l'élément `<ul>`,
- les éléments textes/HTML à afficher avant ou après les liens (la balise `<a>`),
- les éléments textes/HTML à afficher avant ou après le texte des liens (utilisés pour mettre les textes des liens dans des `<span>`, par exemple ),
- un Walker personnalisé pour les plus téméraires.

On va parler un peu des Walkers plus loin. On ne va pas en créer un de toute pièce, mais on va utiliser des hooks disponibles dans le Walker par défaut utilisé par WordPress pour créer les menus dans un exemple.

Pour rappel, un Walker est une classe PHP utilisée pour créer des éléments récursivement. Les éléments de menus WordPress peuvent avoir des sous-éléments, et donc des imbriques de `<ul>` (sous-menus) dans des `<li>` (éléments du menu parent). Un menu dans un menu, en fait. C'est pour générer ce type de structure qu'un Walker est utilisé.

La fonction permet de personnaliser un peu l'HTML des menus à l'aide de ses paramètres. Par exemple, on peut ajouter un peu d'HTML autour des liens et des textes des liens à l'aide des paramètres `before`, `after`, `link_before` et `link_after`.

Pour modifier les paramètres de la fonction, il y a deux façons. La première consiste à dupliquer le fichier responsable de l'affichage du menu de votre thème dans notre thème enfant, et à modifier l'appel à la fonction.

Pour Twenty Nineteen, par exemple, il nous faut dupliquer le fichier `template-parts/header/site-branding.php` dans notre thème enfant. Puis, on peut ajouter nos paramètres personnalisés :

```
wp_nav_menu(  
    array(  
        'theme_location' => 'menu-1',  
        'menu_class'      => 'main-menu custom-menu-class',           // Add a class  
        'items_wrap'       => '<ul id="%1$s" class="%2$s">%3$s</ul>', // Add extra container  
        'container_class' => 'custom_container_class',            // Add extra wrappers  
        'before'          => '<span class="before">',  
        'after'           => '</span>',  
        'link_before'     => '<span class="link-before">',  
        'link_after'      => '</span>',  
    )  
);
```

Ici, on modifie la classe CSS du conteneur, et on ajoute des `<span>` autour des liens et du texte des liens. Cela donne l'HTML suivant :

```
</a>
▼<header id="masthead" class="site-header">
  ::before
  ▼<div class="site-branding-container">
    ▼<div class="site-branding">
      ▶<h1 class="site-title">...</h1>
      ▶<p class="site-description">...</p>
      ▼<nav id="site-navigation" class="main-navigation" aria-label="Menu supérieur">
        ...
        ▼<div class="custom_container_class" style="display: flex; align-items: center; justify-content: space-between; gap: 10px; margin-bottom: 10px;">
          ...
          ▼<ul id="menu-custom-menu" class="main-menu custom-menu-class">
            ▼<li id="menu-item-89" class="test menu-item menu-item-type-post_type menu-item-object-page menu-item-89">
              ▼<span class="before">
                ▼<a title="Title attribute" target="_blank" rel="nofollow" href="https://wpcookbook.local/sample-page/">
                  <span class="link-before">Sample Page</span>
                </a>
              </span>
            </li>
          </ul>
        </div>
      </div>
    </div>
  </div>
</header>
```

L'HTML comprends bien nos `<span>` et nos classes CSS.

Cela fonctionne correctement, mais dupliquer tout un fichier pour modifier seulement quelques paramètres, c'est un peu dommage. Mais heureusement, il y a un hook pour ça ! Le filtre `wp_nav_menu_args` nous permet de modifier les arguments sans avoir à dupliquer tout un fichier.

On peut donc faire exactement la même chose que précédemment avec une seule fonction dans `functions.php`, sans dupliquer le fichier `template-parts/header/site-branding.php`.

Dans votre fichier `functions.php`, ajoutez ceci :

```

add_filter( 'wp_nav_menu_args', 'wpcookbook_nav_menu_args', 10, 1 );
/**
 * Filters the wp_nav_menu() arguments
 *
 * @param array $args Arguments passed to wp_nav_menu()
 * @return array $args
 */
function wpcookbook_nav_menu_args( $args ){
    if( 'menu-1' === $args['theme_location'] ){
        $args['menu_class']      = 'main-menu custom-menu-class';
        $args['container_class'] = 'custom-container-class';
        $args['before']          = '<span class="before">';
        $args['after']           = '</span>';
        $args['link_before']     = '<span class="link-before">';
        $args['link_afterafter'] = '</span>';
    }

    return $args;
}

```

Ici, on s'assure simplement d'être sur le bon emplacement (`$args['theme_location']`) de menu pour éviter de modifier tous les menus du thème, et on ajoute nos paramètres, simplement. C'est le thème qui gouverne l'identifiant de la zone de menu quand il déclare ses menus.

Grâce à ce snippet, on a exactement le même résultat, mais on n'a pas dupliqué de fichier dans notre thème enfant, ce qui est bien mieux ! Si on duplique des fichiers dans le thème enfant, ce sont ces derniers qui prennent la main sur ceux du thème parent. Donc en cas de mise à jour du thème parent, on ne bénéficie pas des améliorations apportées à ce fichier. Donc moins on surcharge, mieux c'est !

La fonction `wp_nav_menu()` est puissante, mais elle traite tous les éléments de menu de la même façon. Comment faire si on veut appliquer une classe CSS à un seul élément particulier ? Ajouter des classes et attributs dynamiques ?

## Ajouter des classes CSS aux éléments de menu <li>

Les éléments de menu sont générés par le `Walker_Nav_Menu`, et celui-ci expose plusieurs hooks bien utiles.

Pour ajouter des classes CSS, on dispose du filtre `nav_menu_css_class`.

```

add_filter( 'nav_menu_css_class' , 'wpcookbook_menu_item_classes', 10, 4 );
/**
 * Filters the menu <li> CSS classes
 *
 * @param array $classes Array of classes to apply to <li>
 * @param WP_Post $item Menu item data
 * @param object $args Arguments passed to corresponding `wp_nav_menu()` call
 * @param int $depth Menu depth
 */
function wpcookbook_menu_item_classes( $classes, $item, $args, $depth ) {
    if( 'menu-1' === $args->theme_location ){
        $classes[] = 'new-class';
    }
    return $classes;
}

```

Le filtre fournit 4 paramètres à notre fonction de rappel :

- `$classes` est le tableau des classes par défaut appliquées sur les `<li>`
- `$item` est l'objet `WP_Post` correspondant à notre élément de menu. Oui, les éléments de menu sont des types de contenus personnalisés, et ont donc les mêmes données que des articles.
- `$args` sont les paramètres passés à l'appel de `wp_nav_menu()`. Mais attention, ici, ils sont présentés sous forme d'objet et pas de tableau !
- `$depth` est la profondeur de l'élément de menu

Ici, on vérifie qu'on est bien sur le menu que l'on veut modifier, et on ajoute notre classe CSS au tableau de classes existant. Attention car les classes sont présentées sous forme de tableau, et il y a déjà des classes par défaut. Dans l'exemple, on ajoute une classe sans toucher aux classes par défaut.

Dans ce filtre, on a accès à énormément d'informations. Il est donc très facile de générer des classes CSS personnalisées en fonction de l'élément de menu, de l'URL vers lequel il pointe, du type de page vers lequel il mène, etc... Il suffit de lire les paramètres `$item` et `$args` fournis !

## Ajouter des attributs aux liens `<a>`

Pour ajouter des attributs personnalisés aux liens, on utilise le filtre `nav_menu_link_attributes`.

Ce dernier fonctionne exactement comme le précédent, et fournit exactement les mêmes informations. Il est donc tout aussi puissant et permet d'appliquer une logique personnalisée de la même façon.

```

add_filter( 'nav_menu_link_attributes', 'wpcookbook_menu_link_attributes', 10, 4 );
/**
 * Adds attributes to menu <a> element
 *
 * @param array $atts      Array of link attributes
 * @param WP_Post $item    The current menu item.
 * @param array $args      An object of wp_nav_menu() arguments.
 * @param int $depth       Depth of menu item. Used for padding.
 * @return array           $atts      Array of link attributes
 */
function wpcookbook_menu_link_attributes( $atts, $item, $args, $depth ){
    if( 'menu-1' === $args->theme_location ){
        $atts['class'] = 'custom-link-class';
    }
    return $atts;
}

```

Ici, on ajoute simplement un attribut `class` aux balises `<a>`. On peut ajouter tout type d'attribut, et modifier ceux que l'on a donnés dans l'interface d'administration des menus. Attention ici, on manipule les attributs de la balise `<a>`, et pas du `<li>`.

## Ajouter des classes CSS aux sous-menu `<ul>`

Si vous utilisez des sous-menus, WordPress ajoutera par défaut la class 'sub-menu' aux éléments `<ul>` correspondants. Si vous voulez ajouter votre propre classe, c'est possible. Il faut utiliser le hook `nav_menu_submenu_css_class`.

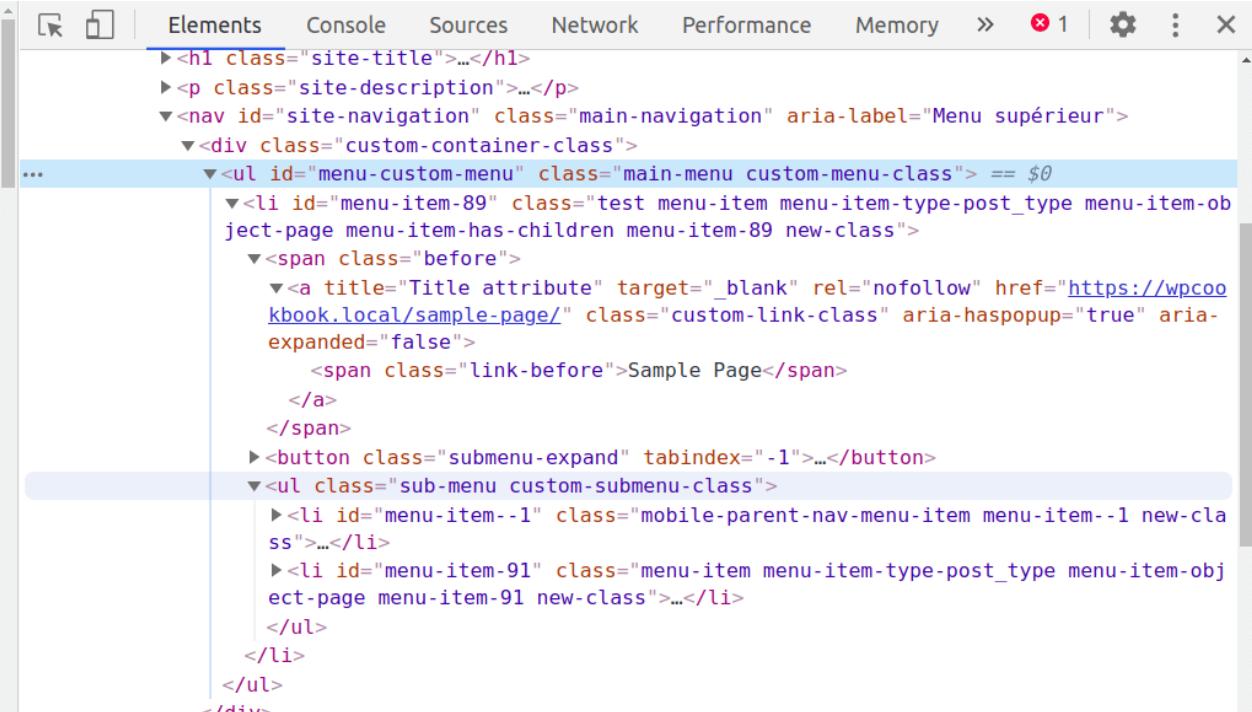
```

add_filter( 'nav_menu_submenu_css_class' , 'wpcookbook_submenu_classes', 10, 3 );
/**
 * Adds CSS classes to submenu <ul>
 *
 * @param array $classes  Array of CSS classes
 * @param object $args     An object of wp_nav_menu() arguments.
 * @param int $depth      Depth of menu. Used for padding.
 * @return array           $classes  Array of CSS classes
 */
function wpcookbook_submenu_classes( $classes, $args, $depth ){
    $classes[] = 'custom-submenu-class';
    return $classes;
}

```

La fonction de rappel reçoit les mêmes arguments que pour les autres hooks, donc de la même façon, il est très facile d'ajouter une logique personnalisée plus complexe ici.

Cela donnera la structure HTML suivante :



The screenshot shows the Chrome DevTools Elements tab with the page source expanded. A specific section of the code is highlighted in blue, representing the custom menu structure. The highlighted code is:

```
><ul id="menu-item-89" class="test menu-item menu-item-type-post_type menu-item-object-page menu-item-has-children menu-item-89 new-class">
    <li id="menu-item-1" class="mobile-parent-nav-menu-item menu-item--1 new-class">...</li>
    <li id="menu-item-91" class="menu-item menu-item-type-post_type menu-item-object-page menu-item-91 new-class">...</li>
</ul>
```

This highlights the main menu item and its first child item, both of which have the class "new-class" applied.

L'élément `<ul>` du sous menu a bien notre classe personnalisée.

## Ajouter des éléments HTML dans le menu.

Si vous observez bien l'HTML généré par Twenty nineteen sur la capture d'écran précédente, vous remarquerez qu'un bouton est ajouté juste avant notre sous-menu `<ul>`. Comment fait-on cela ?

Selon la complexité de l'HTML que vous voulez ajouter, vous avez deux solutions. Si l'élément que vous souhaitez ajouter doit se placer juste avant ou après le titre de la navigation, mais dans la balise `<a>` vous pouvez utiliser le filtre `nav_menu_item_title`. Mais ce filtre n'expose que le titre de la navigation non modifié. Rien d'autre.

Si vous avez besoin de voir un peu plus large, vous pouvez utiliser `walker_nav_menu_start_el`, qui expose la balise ouvrante `<a>`, le texte du lien et la balise fermante `</a>`. Mais attention car les arguments `before`, `after`, `link_before` et `link_after` de l'appel à la fonction `wp_nav_menu()` seront aussi fournis !

## Changer le texte du lien

Pour accéder au titre de la navigation, on utilise le filtre `nav_menu_item_title`. Celui-ci permet de filtrer le texte du titre, et de lui ajouter de l'HTML, si besoin.

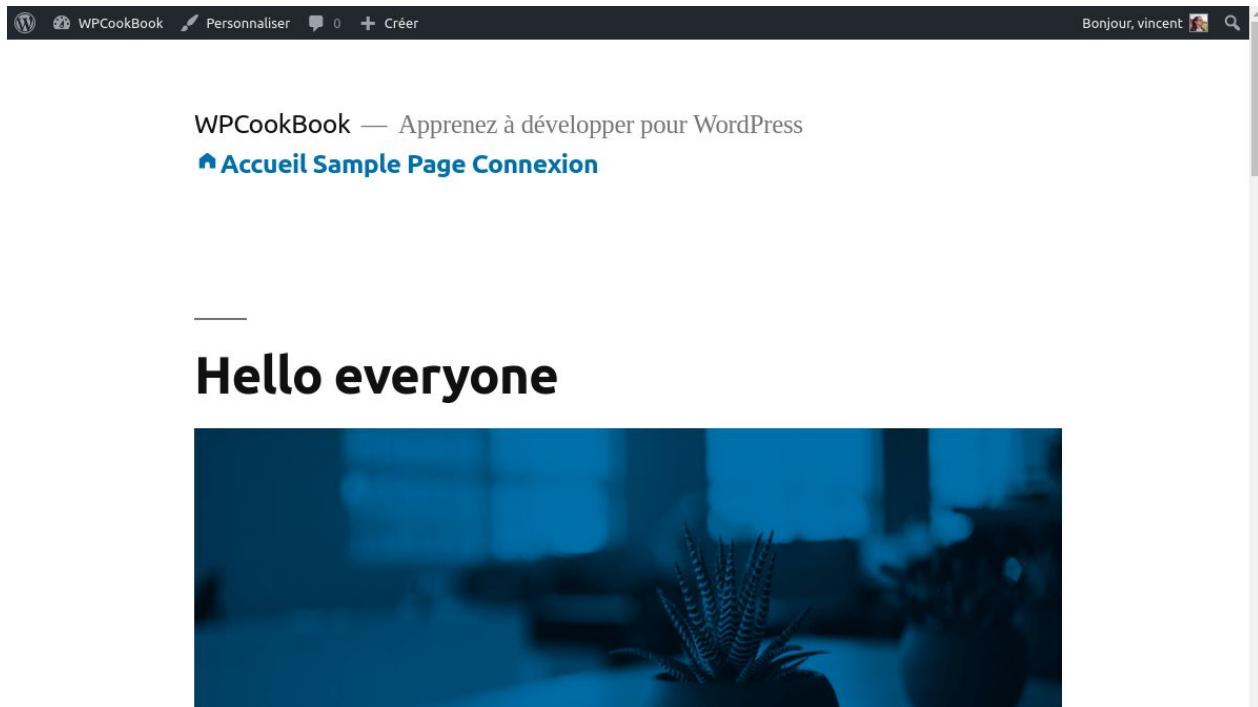
Mais attention, car le titre fourni à la fonction de rappel est “brut” et ne comprend pas les arguments `link_before` et `link_after` !

```

add_filter( 'nav_menu_item_title' , 'wpcookbook_nav_menu_item_title', 10, 4 );
/**
 * Filters the menu item text
 *
 * @param string $title The menu item's title.
 * @param WP_Post $item The current menu item.
 * @param array $args An object of wp_nav_menu() arguments.
 * @param int $depth Depth of menu item. Used for padding.
 * @return string $title The menu item's final title.
 */
function wpcookbook_nav_menu_item_title( $title, $item, $args, $depth ){
    if( get_home_url() === $item->url ){
        $title = '<svg style="width: 1em; height: 1em" viewBox="0 0 16 16"><rect x="0" fill="none" width="16" height="16"/><g><path d="M7.4 3.5l-4 3.2c-.3.2-.4.5-.4.8V13h3.5v-2.5C6.5 9.7 7.2 9 8 9s1.5.7 1.5 1.5V13H13V7.5c0-.3-.1-.6-.4-.8l-4-3.2c-.3-.3-.9-.3-1.2 0z"/></g></svg>' . $title;
    }
    return $title;
}

```

Le filtre prend encore les mêmes paramètres que pour les exemples précédents. Pas de surprises ici. Je vérifie si l'élément de menu pointe vers l'accueil du site en utilisant `get_home_url()`, et si c'est le cas, je modifie le titre en lui accolant une petite icône de maison directement grâce à un `<svg>`.



Cela fonctionne, mais si vous observez attentivement la structure HTML, vous allez pouvoir constater que notre `<svg>` est inséré à l'intérieur des `<span>` que nous avons passés en paramètres dans `wp_nav_menu()`. Cela peut être désirable ou non, à vous de voir. Il faut simplement en avoir conscience.

```

<ul id="menu-item-94" class="menu-item menu-item-type-custom menu-item-object-cu
stom current-menu-item current_page_item menu-item-home menu-item-94 new-class">
    <span class="before">
        <a href="https://wpcookbook.local" aria-current="page" class="custom-link-cla
ss">
            <span class="link-before" style="width: 1em; height: 1em; viewBox="0 0 16 16"></span>
                Accueil
            </span>
        </a>
    </span>
</li>

```

Notre icone est insérée à l'intérieur du `<span>`

On peut ajuster notre fonction pour que le titre ait aussi sa balise HTML. Cela permettra de le styler plus facilement.

```

function wpcookbook_nav_menu_item_title( $title, $item, $args, $depth ){
    if( get_home_url() === $item->url ){
        $icon = '<svg style="width: 1em; height: 1em" viewBox="0 0 16 16"><rect x="0"
fill="none" width="16" height="16"/><g><path d="M7.4 3.5l-4 3.2c-.3.2-.4.5-.4.8V13h3.5v-2.5C6.5
9.7 7.2 9 8 9s1.5.7 1.5 1.5V13H13V7.5c0-.3-.1-.6-.4-.8l-4-3.2c-.3-.3-.9-.3-1.2 0z"/></g>
</svg>';
        $title = sprintf(
            '%s<span class="item-title">%s</span>',
            $icon,
            $title
        );
    }
    return $title;
}

```

En ajoutant un `<span>` avec une classe CSS de `item-title`, notre HTML est un peu plus clair.

Le texte et l'icone sont maintenant deux éléments HTML adjacents.

## Filtrer tout l'HTML de l'élément de menu

Si vous avez besoin d'un peu plus de contexte, alors vous pouvez utiliser le filtre `walker_nav_menu_start_el`. Ce filtre ne fournit pas seulement le texte du lien, mais tout l'HTML du lien, y compris les arguments passés dans `wp_nav_menu()`.

On a donc tout l'HTML correspondant à :

- l'argument `before` passé à `wp_nav_menu()`
- puis la balise `<a>` ouvrante et ses attributs,
- puis l'argument `link_before` passé à `wp_nav_menu()`
- puis le texte du lien
- puis l'argument `link_after`,
- enfin l'argument `after`.

En gros, on dispose de tout l'HTML entre les balises `<li>`. Si l'élément dispose d'un sous-menu, on ne dispose pas de l'élément `<ul>` ouvrant ce sous-menu.

C'est donc ce hook qu'il faut utiliser si vous avez besoin d'insérer de l'HTML avant ou après le lien complet.

C'est d'ailleurs comme ça que le thème Twenty Nineteen insère un bouton pour faire apparaître les sous-menus.

```

add_filter( 'walker_nav_menu_start_el' , 'wpcookbook_nav_menu_start_el', 10, 4 );
/**
 * Filters the menu item HTML
 *
 * @param string $output The menu item's starting HTML output.
 * @param WP_Post $item The current menu item.
 * @param int $depth Depth of menu item. Used for padding.
 * @param array $args An object of wp_nav_menu() arguments.
 * @return string $output The menu item's final HTML output.
 */
function wpcookbook_nav_menu_start_el( $output, $item, $depth, $args ) {
    if( in_array( 'menu-item-has-children', $item->classes, true ) ) {
        // Add SVG icon to parent items.
        $icon = twentynineteen_get_icon_svg( 'keyboard_arrow_down', 24 );

        $output .= sprintf(
            '<button class="submenu-expand" tabindex="-1">%s</button>',
            $icon
        );
    }
    return $output;
}

```

Ici, on vérifie si l'élément de menu en question a des enfants (la classe CSS `menu-item-has-children` est appliquée par défaut si c'est le cas), et si oui, alors on ajoute notre bouton après le lien. Ce bouton viendra juste APRES le lien (la balise `<a>`), mais AVANT le sous-menu (la `<ul>` imbriquée).

## Filtrer les éléments de menus

On a vu comment modifier l'HTML des menus : via `wp_nav_menu()` et ses arguments, via les différents filtres présents dans la fonction et le `Walker_Nav_Menu`. Mais la fonction `wp_nav_menu()` expose aussi un filtre pratique permettant de filtrer les éléments de menu eux-mêmes, et ce avant même qu'ils soient affichés.

C'est très pratique pour, par exemple, cacher des éléments de menu selon le statut de l'utilisateur sans avoir recours à des `display: none` ; maladroits.

```
add_filter( 'wp_nav_menu_objects', 'wpcookbook_nav_menu_objects', 10, 2 );
/**
 * Filters the nav menus to hide items based on logged-in status.
 *
 * @param array $menu_items Menu items
 * @param array $args        Args passed in to wp_nav_menu
 * @return array $menu_items
 */
function wpcookbook_nav_menu_objects( $menu_items, $args ) {
    if( ! is_user_logged_in() ){
        $menu_items = array_filter( $menu_items, function( $item ){
            return in_array( 'logged-in-only', $item->classes );
        });
    }
    return $menu_items;
}
```

Le filtre `wp_nav_menu_objects` nous fournit les objets à afficher, ainsi que les arguments passés à `wp_nav_menu()`. Dans ce snippet, on vérifie si l'utilisateur est connecté, et s'il ne l'est pas, on va enlever tous les éléments ayant pour classes CSS 'logged-in-only', en utilisant un simple `array_filter()`.

Si vous ne connaissez pas trop `array_filter()`, c'est une fonction PHP qui permet de filtrer un tableau. On lui passe un tableau à filtrer et une fonction de rappel qui sera appelée pour chaque élément de ce tableau, et la fonction renvoie le tableau filtré. La fonction de rappel prend un élément du tableau en paramètre et doit renvoyer `true` si vous voulez garder l'élément, et `false` sinon.

Dans notre exemple, on renvoie simplement `false` si l'élément de menu sur lequel on se situe dans notre filtre ne doit pas être affiché.

Maintenant, il suffit d'ajouter les classes CSS `logged-in-only` aux éléments que l'on veut cacher aux utilisateurs non-connectés, via l'interface standard d'administration des menus.

Publié

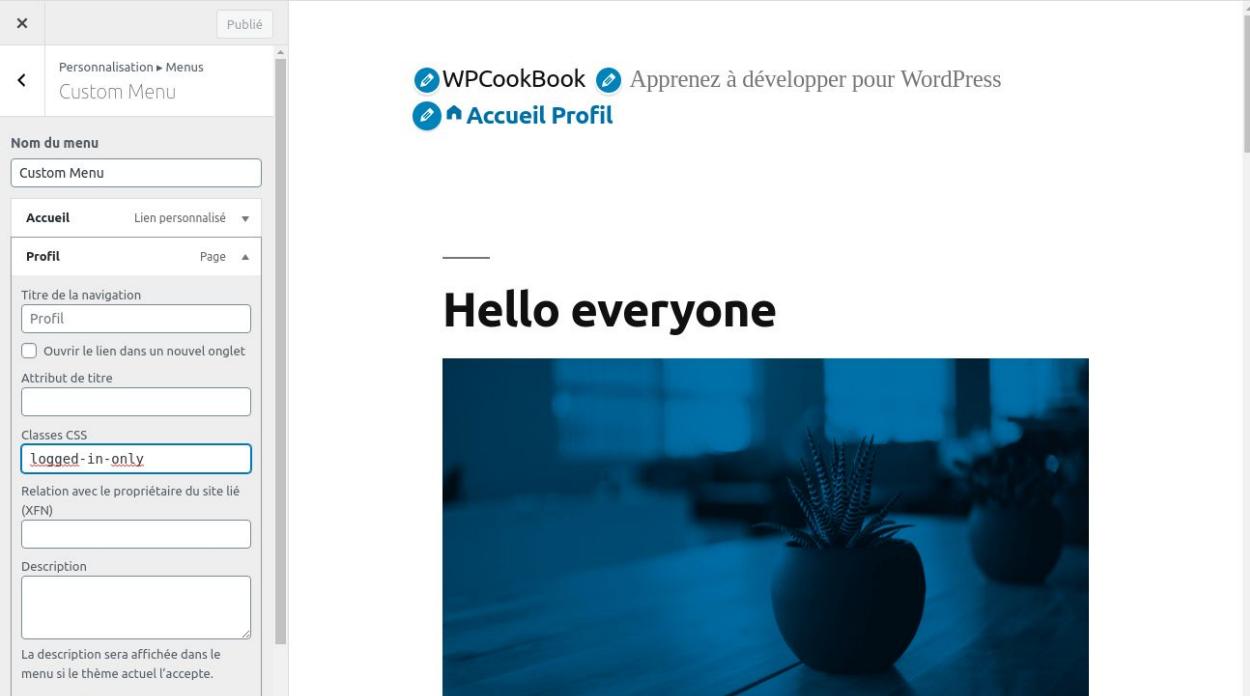
Personnalisation ▶ Menus  
Custom Menu

Nom du menu  
Custom Menu

Accueil Lien personnalisé ▾  
Profil Page ▾

Titre de la navigation Profil  
 Ouvrir le lien dans un nouvel onglet  
Attribut de titre  
Classes CSS logged-in-only  
Relation avec le propriétaire du site lié (XFN)  
Description  
La description sera affichée dans le menu si le thème actuel l'accepte.  
Original: Profil

**Hello everyone**



L'élément de menu "Profil" a bien sa classe *logged-in-only*

WPCookBook — Apprenez à développer pour WordPress  
◀ Accueil

## Hello everyone



Et on ne le voit pas quand on n'est pas connecté !

## Filtrer TOUT l'HTML du menu

Pour les plus téméraires, il est également possible de filtrer tout l'HTML généré par `wp_nav_menu()`, juste avant de l'afficher. Cela peut être utile pour ajouter du contenu avant ou après, ou effectuer des modifications chirurgicales à coups de `str_replace()` ou `preg_replace()`.

```
add_filter( 'wp_nav_menu', 'wpcookbook_nav_menu', 10, 2 );
/**
 * Filters the whole nav menu HTML
 *
 * @param string $nav_menu Menu HTML
 * @param array  $args      Args passed in to wp_nav_menu()
 * @return string $nav_menu
 */
function wpcookbook_nav_menu( $nav_menu, $args ){
    return $nav_menu . '<button>Click me</button>';
}
```

Je n'ai personnellement encore jamais eu besoin d'utiliser ce filtre, mais bon. On ne sait jamais !

## Ce qu'il faut retenir

- `wp_nav_menu()` offre déjà plusieurs options de personnalisation, et un filtre `wp_nav_menu_args` pour modifier ces arguments. Commencez par explorer cette option.
- Vous pouvez ajouter des classes CSS personnalisées aux éléments de menu `<li>` à l'aide du hook `nav_menu_css_class`.
- Vous pouvez modifier les attributs des liens (les balises `<a>`) à l'aide du hook `nav_menu_link_attributes`.
- Ces fonctions fournissent l'élément de menu, ainsi que tous les arguments passés à la fonction `wp_nav_menu()`, ce qui permet un ciblage très précis.
- Vous pouvez personnaliser la classe CSS des sous-menus (balises `<ul>`) grâce à `nav_menu_submenu_css_class`.
- Le titre des éléments de menu est personnalisable à l'aide du filtre `nav_menu_item_title`. Ce hook permet aussi d'ajouter de l'HTML à l'intérieur de la balise `<a>` de l'élément de menu.
- `walker_nav_menu_start_el` vous donne TOUT le code HTML d'un élément de menu (sauf la balise `<li>` fermante, et le sous-menu `<ul>` le cas échéant). Cela vous permet donc d'ajouter du code ENTRE les éléments de menus.
- Vous pouvez aussi filtrer les éléments à afficher eux-même à l'aide de `wp_nav_menu_objects`.
- Enfin, vous pouvez filtrer le code HTML du menu entier à l'aide de `wp_nav_menu`, si vous aimez le danger.

Entre les arguments filtrables de `wp_nav_menu()`, et ceux du `Walker_Nav_Menu`, vous avez énormément de possibilité de personnalisation. Et il y a encore d'autres hooks que nous n'avons pas explorés ! Mais ceux listés ici vous permettront de combler 95% de vos besoins.

# Comment fonctionnent les hooks d'activation / de désactivation / de désinstallation

Vous avez sûrement déjà remarqué que certaines extensions agissent dès l'activation. Souvent, elles ont besoin de "préparer le terrain" pour être utilisables immédiatement. Je pense en premier à la création de tables personnalisées dans la base de données, ou la création de nouveaux rôles utilisateurs.

Ces actions doivent être faites une seule fois, et l'activation de l'extension est un bon moment pour le faire. Aussi, parfois elles ont besoin de faire le ménage derrière elles quand elles sont désactivées ou désinstallées.

Toutes ces actions uniques sont gérées à l'aide de hooks spéciaux: les hooks d'activation, de désactivation et de désinstallation. Ces hooks sont particuliers, car on n'utilise pas `add_action()` directement pour se greffer dessus (même si on pourrait), mais ils sont déclarés en utilisant leur propre fonction spécifique.

## `register_activation_hook()`

Pour enregistrer une fonction à déclencher dès l'activation, on utilise `register_activation_hook()`.

```
register_activation_hook( string $file, callable $function )
```

La fonction `register_activation_hook()` prend le nom de votre extension en premier paramètre et une fonction de rappel à exécuter en second.

En fait, quand vous activez une extension, le hook `activate_PLUGINNAME` est déclenché, où `PLUGINNAME` est le nom du fichier racine de votre extension. En coulisse, tout ce que fait la fonction est d'ajouter votre fonction de rappel à ce hook avec un `add_action()`. Rien de foufou.

Vu qu'il faut passer le nom de votre extension en premier paramètre, une solution simple est d'appeler cette fonction dès le fichier racine de votre extension, en lui passant `__FILE__`.

```
// In plugin bootstrap file
register_activation_hook( __FILE__ , 'wpcookbook_activation' );
/**
 * Actions to perform on activation
 */
function wpcookbook_activation(){
    ...
}
```

Cette fonction est utile pour toutes les procédures à n'exécuter qu'une seule fois, ou pour rafraîchir les permaliens quand vous déclarez un type de contenus personnalisés, par exemple.

Pour l'exemple, imaginons que nous voulions créer un rôle utilisateur. Les rôles sont inscrits en base de données, donc on peut soit vérifier que notre nouveau rôle existe à chaque chargement de page en se hookant sur `init` par exemple, ou alors on peut le faire une seule fois à l'activation.

Pour nous aider, créons un petit code court qui va afficher les rôles disponibles, et créons une page avec ce code court. Dans le fichier `functions.php` de notre thème actif, ajoutez :

```
// in functions.php
add_shortcode( 'wpcookbook_roles', 'wpcookbook_roles' );
/**
 * Callback function for our shortcode
 * Displays available user roles
 *
 * @param array $atts Shortcode attributes
 */
function wpcookbook_roles( $atts ){
    ob_start();
    $roles = wp_roles();
    echo '<pre>';
    var_dump($roles->roles);
    echo '</pre>';
    return ob_get_clean();
}
```

Le shortcode ne fait qu'un var\_dump des rôles et leurs capacités. C'est juste pour vérifier que nos rôles sont bien enregistrés.

Dans une extension toute fraîche, ajoutez :

```
register_activation_hook( __FILE__ , 'wpcookbook_activation' );
/**
 * Actions to perform on activation
 */
function wpcookbook_activation() {
    if ( get_role( 'client' ) ) {
        return;
    }
    add_role( 'client', 'Client', array(
        'read' => true,
    ) );
}
```

Pour ajouter un nouveau rôle, on utilise la fonction `add_role()`. Tout simplement. On prend juste le soin de vérifier que le rôle n'existe pas avant. On ne sait jamais.

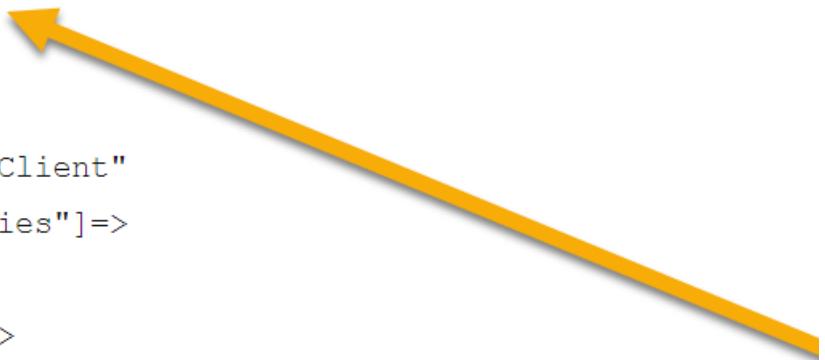
```
add_role( string $role, string $display_name, array $capabilities = array() )
```

Il suffit de lui passer un identifiant `$role`, un nom d'affichage `$display_name` et un tableau de capacités `$capabilities`. L'objet de ce chapitre n'est pas d'explorer les rôles utilisateurs en détails, donc pas de panique si tout n'est pas clair. On aura l'occasion de voir tout ça en détails plus tard.

Si on active notre extension, notre rôle est ajouté et apparaît bien dans la liste des rôles utilisateurs disponibles.

```
}

["subscriber"]=>
array(2) {
    ["name"]=>
        string(10) "Subscriber"
    ["capabilities"]=>
        array(2) {
            ["read"]=>
                bool(true)
            ["level_0"]=>
                bool(true)
        }
    }
["client"]=>
array(2) {
    ["name"]=>
        string(6) "Client"
    ["capabilities"]=>
        array(1) {
            ["read"]=>
                bool(true)
        }
    }
}
```



[Modifier](#)

Notre rôle est bien enregistré.

Pour tester, modifiez la fonction de rappel de notre shortcode comme suit :

```
function wpcookbook_roles( $atts ){
    ob_start();

    $current_user = wp_get_current_user();
    $user_role = ( array ) $current_user->roles;
    printf( __( '<p>Your current role is: <strong>%s</strong></p>' ), '21-activation-hooks'
    ), join( ' ', $user_role ) );

    $roles = wp_roles();
    echo '<pre>';
    var_dump($roles->roles);
    echo '</pre>';
    return ob_get_clean();
}
```

On peut donc maintenant créer un utilisateur avec ce nouveau rôle et se connecter avec ce profil, et sur la page où notre code court est présent, on peut voir que notre nouvel utilisateur connecté a bien pour rôle 'client'.

The screenshot shows a WordPress website with the title "WPCookBook — Apprenez à développer pour WordPress". The navigation menu includes "Page Article Catégorie Contact". The user "vdub" is logged in, as shown in the top right corner. The main content area has a heading "Roles" and displays the message "Your current role is : client". Below this, a large block of PHP code is shown, which is a dump of the \$roles variable from the wp\_roles() function. The code output is:  
array(6) {  
 ["administrator"]=>  
 array(2) {  
 ["name"]=>  
 string(13) "Administrator"  
 ["capabilities"]=>  
 array(61) {  
 ["switch\_themes"]=>  
 bool(true)  
 }  
 }  
}  
Notre utilisateur est bien un client.

Si on désactive l'extension, le rôle est toujours présent car il est inscrit en base. Du coup, il n'y a aucun changement pour les utilisateurs présents. Ceux étant client avant restent client.

## register\_deactivation\_hook()

```
register_deactivation_hook( string $file, callable $function )
```

Cette fonction s'utilise exactement de la même manière que `register_activation_hook()`. On lui passe le nom de notre extension, et une fonction de rappel à exécuter quand l'extension est désactivée. Par exemple, rafraîchir la structure des permaliens avec `flush_rewrite_rules()` peut être utile aussi si votre extension déclarait des types de contenus personnalisés.

C'est à peu près tout ce qu'il y a à savoir !

## register\_uninstall\_hook

```
register_uninstall_hook( string $file, callable $callback )
```

Vous l'aurez probablement deviné, cette fonction déclare une fonction de rappel qui sera déclenchée à la désinstallation de l'extension, c'est-à-dire quand vous cliquerez sur "Supprimer" dans la page d'administration des extensions.

C'est très important. Si vous supprimez votre extension via FTP alors le hook n'est pas déclenché !

Le plus souvent, ce hook est utilisé pour faire un peu de ménage : supprimer des options en base de données, d'autres données, ou même les tables supplémentaires.

Pour notre exemple, quand notre extension est désinstallée, cela peut être judicieux d'ajouter un rôle par défaut comme 'Subscriber', vu qu'il n'y a plus de code actif gérant le rôle 'client'.

```
register_uninstall_hook( __FILE__ , 'wpcookbook_deactivation' );
/**
 * Do not remove 'client' role, but adds a basic 'subscriber' role to user who only had the
 'client' role before
 */
function wpcookbook_deactivation(){
    $clients = get_users( array(
        'role' => 'client',
    ) );
    if ( $clients ){
        foreach ( $clients as $client ) {
            if( 1 == count( $client->roles ) ){
                $client->add_role( 'subscriber' );
            }
        }
    }
}
```

Ici, on essaie de chercher tous les clients, puis on leur ajoute le rôle `subscriber` si `client` était leur seul rôle.

## uninstall.php

Une autre méthode pour déclencher des procédures de désinstallation est d'utiliser un fichier `uninstall.php` à la racine du dossier de votre extension.

Si un tel fichier est présent, alors il sera automatiquement exécuté lors de la désinstallation de votre extension. Désinstaller une extension en la supprimant via FTP ne déclenche pas le code contenu dans ce fichier, évidemment.

Il faut simplement faire attention à vérifier que la constante `WP_UNINSTALL_PLUGIN` est bien définie avant d'entamer vos procédures de nettoyage, pour éviter l'accès direct au fichier.

On peut donc supprimer notre hook de désactivation, créer un fichier `uninstall.php` à la racine, et y coller notre code.

```
<?php
/**
 * Uninstallation file.
 * This code is executed when clicking the delete link on the plugins page.
 */

// If uninstall.php is not called via standard WordPress uninstall process, die
if ( ! defined( 'WP_UNINSTALL_PLUGIN' ) ) {
    die;
}
$clients = get_users( array(
    'role' => 'client'
) );
if ( $clients ){
    foreach ( $clients as $client ) {
        if( 1 === count( $client->roles ) ){
            $client->add_role( 'subscriber' );
        }
    }
}
```

## Pour quelles opérations utiliser quel hook ?

Lors de la désactivation d'une extension, il ne faut surtout pas effacer de réglages ou de contenus entrés par l'utilisateur. Parfois, une désactivation est temporaire. Imaginez la frustration de l'utilisateur s'il doit complètement reparamétrer toute l'extension ou entrer de nouveau son contenu !

Par contre, nettoyer les transients (données temporaires) ou rafraîchir les permaliens avec `flush_rewrite_rules()` sont tout indiqués ici.

Lors de la désactivation d'une extension, vous pouvez supprimer les réglages supplémentaires insérés en base de données. Vous pouvez aussi supprimer les tables devenues inutiles. **Par contre attention au contenu entré par l'utilisateur !** Par exemple, si votre extension permet d'ajouter des champs supplémentaires à vos articles et que votre thème utilise ces champs supplémentaires, supprimer l'extension va supprimer l'interface pour ajouter ces champs mais supprimer la valeur de ces champs pour les articles existants serait une très mauvaise idée. Tout serait cassé sur le devant du site !

De la même façon, attention aux rôles utilisateurs créés ! En désinstallant l'extension, il est judicieux de rendre le rôle indisponible dans l'administration de WordPress, mais que faire des utilisateurs ayant uniquement ce rôle ? On leur supprime le rôle ? Si le thème utilise le rôle ? Ajouter le rôle par défaut est une solution.

Certaines extensions laissent le choix à l'administrateur en créant un réglage permettant d'indiquer ce que l'on souhaite supprimer à la désinstallation. C'est une bonne solution si votre extension est complexe et que choisir pour l'utilisateur peut être problématique.

## Utiliser le hook de désinstallation ou le fichier `uninstall.php`

Ce n'est pas exactement équivalent.

Utiliser une fonction hookée avec `register_deactivation_hook` va vous permettre d'utiliser les fonctions de votre extension, car cette dernière est chargée complètement. Aussi, un hook '`'uninstall_${file}'`' est déclenché, `$file` étant le nom de l'extension désinstallée. Donc les autres extensions qui s'intègrent avec la vôtre peuvent agir quand vous la désinstallez.

Ce n'est pas le cas quand vous utilisez le fichier `uninstall.php`. Aucun hook n'est déclenché et votre extension n'est pas chargée.

A première vue, utiliser le fichier `uninstall.php` n'a aucun intérêt. Mais par exemple, utiliser le hook de désactivation charge toute votre extension donc hooke ses fonctions. Y compris celles enregistrant des types de contenus personnalisés. Or ce serait bien que les types de contenus ne soient pas enregistrés et les règles de réécriture d'URL rafraîchies.

La décision est à prendre selon le besoin, simplement. Si vous avez absolument besoin des fonctions de votre extension, utilisez le hook. Si votre procédure est simple et que vous n'avez aucune dépendance, utilisez le fichier.

## Ce qu'il faut retenir

- `register_activation_hook` et `register_deactivation_hook` permet d'enregistrer des fonctions à exécuter à l'activation et désactivation de l'extension, respectivement.
- `register_uninstall_hook` permet d'enregistrer une fonction qui va s'exécuter quand l'extension est supprimée.
- En règle général, il convient de faire le ménage derrière soi. Le plus souvent, le hook de désinstallation est plus pertinent, car l'utilisateur ne veut plus utiliser l'extension.
- On ne fait pas le grand ménage sur une simple désactivation, car parfois celle-ci est temporaire: on désactive pour debugger. Donc pour éviter les pertes de données malheureuses, on évite de supprimer réglages et données sur le hook de désactivation.
- On peut utiliser un fichier `uninstall.php` à la racine du dossier de l'extension pour la procédure de désinstallation. Attention à vérifier la constante `WP_UNINSTALL_PLUGIN`.

# Comprendre les rôles et capacités de WordPress

WordPress utilise un ensemble de rôles et capacités pour gérer les utilisateurs et leurs droits, et met à notre disposition 5 rôles de base: `subscriber` (abonné), `contributor` (contributeur), `author` (auteur), `editor` (éditeur), `administrator` (administrateur).

Les rôles de WordPress représentent un peu le titre ou le job de l'utilisateur, et les capacités représentent ce que l'utilisateur peut faire sur le site.

Les capacités sont décrites sous forme de mots-clés, comme `edit_posts` ou `manage_options`, et WordPress va pouvoir vérifier si l'utilisateur a certaines capacités pour l'autoriser à effectuer certaines actions, ou lui montrer certains éléments.

Par exemple, un utilisateur sans la capacité `manage_options` ne pourra accéder ni modifier les réglages du site ou des extensions, et sans `edit_posts` il ne pourra pas modifier ses publications.

## Rôles et capacités par défaut

À chaque rôle sont associées des capacités par défaut, si bien que chaque utilisateur qui s'est vu affecter un rôle s'est automatiquement vu affecter l'ensemble des capacités par défaut de ce rôle.

Pour voir toutes les capacités par défaut de chaque rôle, on va créer un petit code court qui va afficher un tableau avec les utilisateurs, leurs rôles, et leurs capacités.

Dans une nouvelle petite extension, ajoutez le code suivant :

```

// in main plugin file
add_shortcode( 'wpcookbook_roles', 'wpcookbook_roles' );
/**
 * User Table shortcode display callback
 *
 * @param array $atts Shortcode attributes
 */
function wpcookbook_roles( $atts ){
    $users = get_users();
    ob_start();
    ?>
    <table>
        <thead>
            <tr>
                <th style="width:25%"><?php _e( 'Username', '22-roles-capabilities' ); ?>
            </tr>
        </thead>
        <tbody>
            <?php foreach( $users as $user ) : ?>
            <tr>
                <td style="word-break: break-word;"><?php echo esc_html( $user->user_login ); ?></td>
                <td style="word-break: break-word;"><?php echo join( '<br>', $user->roles ); ?></td>
                <td style="word-break: break-word;"><?php echo join( '<br>', array_keys( $user->allcaps ) ); ?></td>
            </tr>
            <?php endforeach; ?>
        </tbody>
    </table>
    <?php
    return ob_get_clean();
}

```

Pardonnez les attributs style, mais pour l'exemple, pas besoin d'une feuille de styles.

L'extension déclare simplement un nouveau code court `wpcookbook_roles` qui va afficher un tableau d'utilisateurs.

Les utilisateurs sont récupérés avec la fonction `get_users()`. Celle-ci prend un tableau d'arguments qui permet de préciser notre requête. Ainsi, on peut aller chercher un nombre précis d'utilisateurs, ayant un rôle précis, une certaine métadonnée, ou encore les ordonner de façon spécifique. La fonction est très puissante, mais pour notre besoin les arguments par défaut font très bien l'affaire.

`get_users()` renvoient un tableau d'objets `WP_USER` dans lequel on va aller piocher les informations dont on a besoin : le `user_login`, les rôles et les capacités. Ces dernières sont organisées sous la forme d'un tableau. Pour chaque capacité, on a une entrée `#{capacité} => true`, d'où le `array_keys()`.

Créez 5 utilisateurs ayant chacun un des 5 rôles. Puis créez une page avec le code court. En allant sur cette page, vous devriez obtenir ceci :

# Roles

| Username    | Role        | Capabilities   |
|-------------|-------------|--|
| author      | author      | upload_files<br>edit_posts<br>edit_published_posts<br>publish_posts<br>read<br>level_2<br>level_1<br>level_0<br>delete_posts<br>delete_published_posts<br>author |
| contributor | contributor | edit_posts<br>read<br>level_1<br>level_0<br>delete_posts<br>contributor  |
|             |             | moderate_comments  |

Tableau des utilisateurs et leurs rôles

Vous pouvez donc voir toutes les capacités associées à chaque rôle. Vous pouvez remarquer qu'en plus des capacités standards comme `read` ou `edit_posts`, les rôles ont aussi des capacités notées `level_n`,  $n$  allant de 0 à 10. C'est juste une trace de l'ancien système de capacités, dans lequel les utilisateurs avaient un niveau pour les différencier, mais c'est totalement déprécié. Donc n'y prêtez pas attention.

On peut aussi voir que les utilisateurs disposent de capacités ayant le même nom que leur rôle. Ce qui peut à la fois être pratique et trompeur.

Le rôle `subscriber` est le plus faible, ne disposant que du droit de lecture. Le rôle suivant dans la hiérarchie, `contributor`, dispose des droits de lecture mais aussi le droit d'éditer et d'effacer ses propres publications, mais ne peut pas publier. Chaque rôle suivant dans la hiérarchie hérite des capacités des rôles précédents. Le rôle le plus fort, `administrator` peut quant à lui tout faire. Gérer les options, les publications de n'importe quel utilisateur, etc.

Vous pouvez aussi aller voir sur le [codex francophone de WordPress](#). Cette page présente un tableau montrant bien la hiérarchie des rôles et leurs capacités.

## Admins et super admins

Dans une installation de WordPress simple, l'administrateur a tous les droits. Sur une installation multisite, la situation est un peu différente.

Un rôle `super-admin`, au-dessus d'`administrator` est nécessaire. Seul le `super-admin` peut toucher aux réglages du réseau, aux réglages de chaque site, et peut installer thèmes et extensions et les activer ou non sur tous les sites du réseau.

Les `administrator` de chaque site individuel peuvent choisir d'activer ou non les extensions disponibles qui ne sont pas activées d'office sur le réseau, de choisir le thème utilisé parmi ceux installés, et de gérer les options spécifiques au site. Mais ils ne peuvent téléverser des thèmes et extensions, ni mettre à jour WordPress.

## Ajouter et supprimer des rôles

### Créer des nouveaux rôles

Il est tout à fait possible d'ajouter de nouveaux rôles utilisateurs. On l'a d'ailleurs déjà fait dans le chapitre précédent.

Ce qu'il faut bien avoir en tête c'est qu'il vaut mieux créer de nouveaux rôles que de modifier les rôles par défaut de WordPress.

Modifier les rôles par défaut de WordPress peut être très dangereux Si on ne fait pas attention, on a vite fait d'accorder des droits non-souhaités aux utilisateurs, ou d'enlever des droits nécessaires à la bonne administration du site.

Et vu que les rôles utilisateurs sont inscrits en base, il ne suffit pas de supprimer la fonction ajoutant une capacité pour rectifier le tir en cas de souci. Il faut effectivement ajouter du code pour enlever la capacité et restaurer manuellement le rôle à ses capacités d'origine.

Pour être clair, je ne recommande pas du tout l'extension `User Role Editor` ou les extensions similaires. Tout le monde la trouve géniale, car elle permet de modifier les capacités de chaque rôle dans l'administration.

C'est très cool, jusqu'à ce qu'elle tombe en production, et que le client final ou un administrateur lambda (qui n'a rien à faire là en tant qu'administrateur au passage) vienne tout casser en bidouillant les réglages de cette extension. Cette extension permet juste de faire n'importe quoi. Elle n'a rien à faire sur un site en production et si vraiment (mais alors VRAIMENT) vous avez besoin de toucher aux capacités, c'est que vous avez besoin d'un nouveau rôle.

Pour ce faire, on utilise la fonction `add_role()`. Les rôles étant inscrits en base de données (contrairement à d'autres choses, comme les types de contenus personnalisés ou les taxonomies), on a besoin d'utiliser la fonction une seule fois.

On peut le faire en se hookant sur `init`, en vérifiant que le rôle n'existe pas déjà, ou alors à l'activation de l'extension.

Imaginons que l'on ait besoin qu'un utilisateur puisse gérer tout le contenu, mais aussi les utilisateurs d'un site. Par exemple, un responsable des publications, qui pourrait gérer les auteurs et collaborateurs, créer

des comptes collaborateurs, publier leur contenu, etc.

Typiquement, le rôle dont on a besoin pour gérer le contenu est `editor`. Ici, on a besoin d'un `editor`, mais qui puisse aussi gérer les utilisateurs.

Au lieu de bidouiller le rôle `editor`, on va donc créer un rôle `chief-editor`, qui aura toutes les capacités de l'`editor`, mais qui en plus aura les capacités relatives aux utilisateurs.

```
add_action( 'init', 'wpcookbook_add_role' );
/**
 * Adds a new user role
 */
function wpcookbook_add_role(){
    if ( get_role( 'chief-editor' ) ){
        // remove_role( 'chief-editor' );
        return;
    }

$editor_capabilities = array(
    'moderate_comments'      => true,
    'manage_categories'     => true,
    'manage_links'           => true,
    'upload_files'           => true,
    'unfiltered_html'        => true,
    'edit_posts'              => true,
    'edit_others_posts'      => true,
    'edit_published_posts'   => true,
    'publish_posts'          => true,
    'edit_pages'              => true,
    'read'                   => true,
    'edit_others_pages'      => true,
    'edit_published_pages'   => true,
    'publish_pages'          => true,
    'delete_pages'           => true,
    'delete_others_pages'    => true,
    'delete_published_pages' => true,
    'delete_posts'            => true,
    'delete_others_posts'    => true,
    'delete_published_posts' => true,
    'delete_private_posts'   => true,
    'edit_private_posts'     => true,
    'read_private_posts'     => true,
    'delete_private_pages'   => true,
    'edit_private_pages'     => true,
    'read_private_pages'     => true,
    'editor'                  => true,
);
```

```

$additional_capabilities = array(
    'edit_users'      => true,
    'delete_users'    => true,
    'create_users'    => true,
    'list_users'      => true,
    'remove_users'    => true,
    'promote_users'   => true,
    'chief-editor'    => true,
);

add_role( 'chief-editor', 'Chief Editor', array_merge( $editor_capabilities,
$additional_capabilities ) );
}

```

`add_role( string $role, string $display_name, array $capabilities = array() )`

La fonction prend trois paramètres. `$role` est simplement l'identifiant (le slug) du nouveau rôle. `$display_name` est le nom à afficher dans l'administration de WordPress, et `$capabilities` est simplement un tableau de capacités à fournir en donnant les capacités comme clés et un booléen en valeur.

## Hey ! Mais le `$display_name` est en dur ! Donc pas traduisible ?

Alors oui, si vous avez repéré ça, vous avez raison.

En fait, vu que ce `$display_name` est inscrit en base, et pas dans le code source qui s'exécute "en live", il n'est pas possible de récupérer sa traduction via `__()` par exemple. Mais si on met une chaîne traduisible ici et que l'on charge correctement nos traductions avant l'appel à la fonction, alors c'est cette chaîne traduite qui sera inscrite en base.

Si vous créez le rôle alors que votre site est en français, c'est la chaîne française qui est en base. Si votre site est en anglais quand vous créez votre rôle, alors c'est la chaîne anglaise en base, etc. Ouch.

Ce que WordPress fait dans l'administration pour traduire les rôles, c'est d'utiliser la fonction `translate_user_role()` là où il faut les afficher rôles.

`translate_user_role( string $name, string $domain = 'default' )`

La fonction prend un slug de rôles en premier paramètre et va chercher la bonne traduction dans le `$domain` passé en second. Le souci c'est que par défaut, dans l'administration, c'est le domaine de WordPress qui est utilisé, et pas celui de votre extension. Donc même si vous avez traduit le `$display_name` de votre rôle dans vos fichiers de traduction, WordPress ne trouvera pas la bonne traduction, car il cherche au mauvais endroit, simplement.

Une solution est de filtrer le résultat de `translate_user_role()` en se hookant sur `gettext_with_context`, qui est dans la fonction `translate_with_gettext_context()` qui est utilisée par `translate_user_role()`.

```

add_filter( 'gettext_with_context', 'wpcookbook_translate_role', 20, 4 );
/**
 * @param string $translation Translated text
 * @param string $text Text to translate
 * @param string $context Context for translators
 * @param string $domain Text domain
 * @return string $translation
 */
function wpcookbook_translate_role( $translation, $text, $context, $domain ){
    if( 'Chief Editor' === $text && 'User role' === $context && '22-roles-capabilities' !==
$domain ){
        $translation = translate_with_gettext_context( $text, $context, '22-roles-capabilities'
);
    }
    return $translation;
}

```

La fonction de rappel hookée sur `gettext_with_context` prend 4 paramètres. On vérifie donc rapidement si le contexte est bon et si la chaîne à traduire est la bonne (celle entrée en dur dans le `add_role()`, du coup), et si on ne cherche pas déjà dans le bon domaine. Si tout est vérifié, on va chercher la bonne traduction dans le bon domaine.

Ensuite, il suffit qu'il y ait quelque part dans votre code un appel à une fonction de traduction avec votre rôle pour que la chaîne puisse apparaître dans vos fichiers de traduction. Ajoutez simplement `_x( 'Chief Editor', 'User role', '22-roles-capabilities' );` quelque part.

En gros, c'est un peu l'usine à gaz ! Tout ça parce que les rôles sont en base de données, malheureusement.

## Revenons à notre fonction `wpcookbook_add_role()`

La fonction est déclenchée sur `init`, donc à chaque chargement de page. Comme on l'a vu, les rôles sont inscrits en base, donc en théorie on a besoin de le faire une seule fois. D'où le bloc conditionnel qui va vérifier si le rôle n'existe pas déjà.

La ligne commentée `// remove_role( 'chief-editor' );` peut être utile, car si le rôle existe déjà, la fonction `add_role()` ne fait rien. Donc si nous avons besoin d'ajuster ses capacités, il faut supprimer le rôle et le redéclarer ensuite. Donc pendant le développement, cela peut être utile de commenter le `return` pour garder le `remove_role()` à la place. De cette façon, le rôle est "tout frais" à chaque chargement de page.

Normalement, vous devriez être en mesure de créer un nouvel utilisateur avec notre nouveau rôle, ce dernier apparaissant traduit dans l'administration. Aussi, il apparaît bien sur notre page avec notre code court, avec toutes ses capacités. Ouvrez une nouvelle fenêtre dans votre navigateur, connectez-vous avec ce nouvel Editeur en chef, et voilà !

The screenshot shows the WordPress admin interface under the 'Utilisateurs' (Users) section. A user named 'chief-editor' is selected, indicated by a yellow box around the row. The user's details are shown: Nom (Name) is 'Chief Editor', Rôle (Role) is 'Editeur en chef', and Publication(s) (Posts) is 0. Other users listed include 'author', 'contributor', 'editor', and 'subscriber'.

| Identifiant  | Nom          | Adresse de messagerie        | Rôle            | Publication(s) |
|--------------|--------------|------------------------------|-----------------|----------------|
| author       | Author       | author@wpcookbook.local      | Auteur          | 0              |
| chief-editor | Chief Editor | chief-editor@wpcookbook.com  | Editeur en chef | 0              |
| contributor  | Contributor  | contributor@wpcookbook.local | Contributeur    | 0              |
| editor       | Editor       | editor@wpcookbook.local      | Éditeur         | 0              |
| subscriber   | Subscriber   | subscriber@wpcookbook.local  | Abonné          | 0              |

Dans l'administration, notre nouvel utilisateur a bien accès aux Utilisateurs et notre rôle est traduit !

Phew. Compliqué, hein ? Pas tant que ça au final. Il faut juste utiliser `add_role()` en lui donnant les bonnes capacités. C'est la traduction qui est un peu pénible...

## Assigner des rôles à des utilisateurs

Vous pouvez parfaitement assigner un rôle à un utilisateur via l'admin de WordPress. Pas besoin de vous montrer comment on fait. Le souci c'est quand vous avez besoin que certains utilisateurs aient plusieurs rôles.

Dans ce cas, il faut aussi utiliser `add_role()`, mais attention, pas la simple fonction, mais la méthode de la classe `WP_User` !

Il faut donc d'abord récupérer l'utilisateur auquel vous voulez ajouter un rôle. Vous pouvez le faire avec `get_userdata()` en lui passant un identifiant, `get_users()` si vous voulez affecter plusieurs utilisateurs, ou alors `wp_get_current_user()` pour vous occuper uniquement de l'utilisateur connecté.

Pour vous donner un exemple, nous savons que notre Editeur en chef à un identifiant de 8. Sur votre installation, il est peut-être différent. Il suffit de passer la souris sur le lien menant à sa page d'édition de profil et examiner l'URL de la destination pour trouver son `user_id`.

The screenshot shows the WordPress Admin interface under the 'Utilisateurs' (Users) section. A list of users is displayed with columns for user ID (checkbox), profile picture, username, role, and email. The users listed are: editor (Editor, editor@wpcookbook.local), subscriber (Subscriber, subscriber@wpcookbook.local), themedemos (—, themeshop@wpcookbook.local), themereviewteam (Theme Review, themereviewteam@wpcookbook.local), and vincent (—, vincent@wpcookbook.local). The URL https://wpcookbook.local/wp-admin/user-edit.php?user\_id=8 is highlighted in a yellow box at the bottom left.

On peut trouver l'identifiant d'un utilisateur facilement.

Maintenant, nous pouvons lui donner un rôle supplémentaire. Encore une fois, pas besoin de le faire à chaque chargement de page.

```
add_action( 'init', 'wpcookbook_add_another_user_role' );
/**
 * Adds the 'subscriber' role to chief editor
 */
function wpcookbook_add_another_user_role(){
    $chief_editor = get_userdata(8);
    if ( $chief_editor && ! in_array( 'subscriber', $chief_editor->roles ) ) {
        $chief_editor->add_role('subscriber');
    }
}
```

On récupère l'utilisateur à l'aide de `get_userdata()` qui nous retourne une instance de `WP_User`, et on utilise `add_role()` pour lui ajouter le rôle `subscriber` s'il ne l'a pas déjà. Donc à chaque chargement de page, cette vérification aura lieu.

On est d'accord que l'exemple est bidon. Vu que de toute façon, le `chief-editor` a déjà toutes les capacités d'un `editor`, donc d'un `subscriber` ! C'est juste pour l'exemple.

## Supprimer des rôles

Pour supprimer un rôle, on utilise simplement `remove_role()`, comme dans la ligne commentée au début de notre fonction `wpcookbook_add_role()`.

```
function wpcookbook_add_role(){
    if ( get_role( 'chief-editor' ) ){
        remove_role( 'chief-editor' );
        // return;
    }
    ...
}
```

Cela va presque sans dire, mais je le dis quand même : Il est absolument déconseillé de supprimer les rôles par défaut de WordPress !

Pour enlever un rôle d'un utilisateur ou groupe d'utilisateurs, on procède exactement comme pour leur en ajouter, sans oublier de vérifier s'ils ont le rôle en question et si oui, on le supprime avec `remove_role()`.

```
add_action( 'init', 'wpcookbook_remove_another_user_role' );
/**
 * Removes the 'subscriber' role to chief editor
 */
function wpcookbook_remove_another_user_role(){
    $chief_editor = get_userdata(8);
    if ( $chief_editor && in_array( 'subscriber', $chief_editor->roles ) ) {
        $chief_editor->remove_role('subscriber');
    }
}
```

## Ajouter et supprimer des capacités

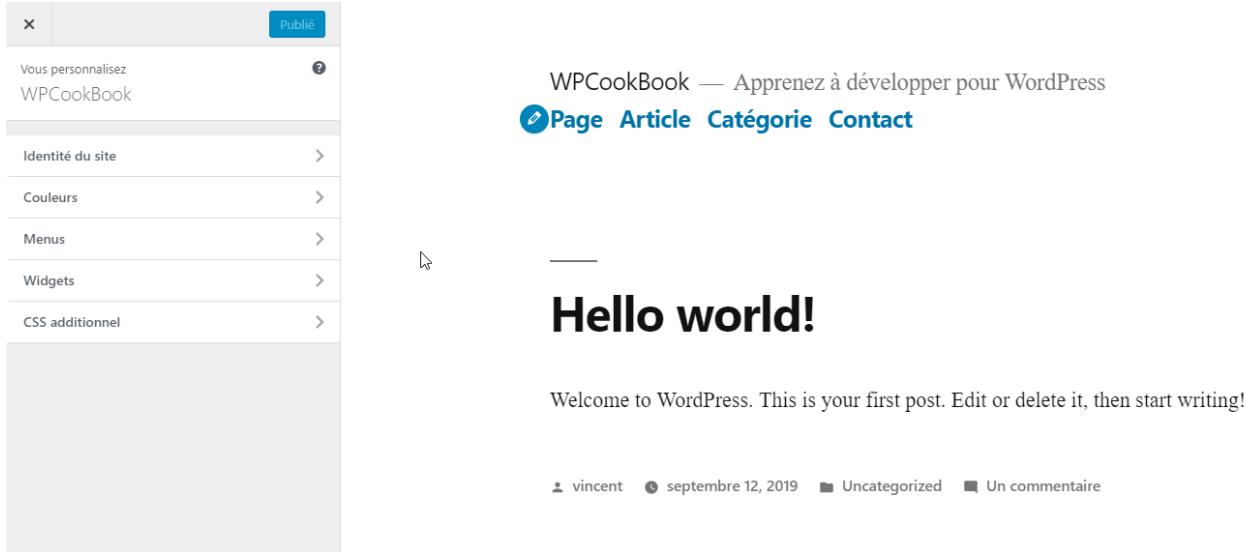
### Ajouter des capacités à un rôle

Pour ajouter des capacités à un rôle, on utilise la méthode `add_cap()` de la classe `WP_Role`. Il faut donc d'abord récupérer l'instance correspondante au rôle auquel on veut ajouter des capacités, et on peut le faire à l'aide de `get_role()`.

```
get_role( string $role )
```

Il suffit de passer un slug de rôle, et la fonction nous renvoie un objet `WP_Role` si le rôle existe. Du coup, on peut par exemple ajouter l'accès à l'outil de personnalisation de WordPress à notre éditeur en chef, vu qu'il est sympa.

```
add_action( 'init', 'wpcookbook_add_capabilities' );
/**
 * Adds theme-related capabilities to chief editor
 */
function wpcookbook_add_capabilities(){
    $chief_editor = get_role( 'chief-editor' );
    if( $chief_editor ){
        $chief_editor->add_cap( 'edit_theme_options' );
        $chief_editor->add_cap( 'customize' );
    }
}
```



Notre éditeur en chef a accès à l'outil de personnalisation

Par contre, si vous comparez avec ce à quoi un administrateur a accès, vous verrez qu'il n'a pas accès à toutes les options de l'outil de personnalisation. Notamment les réglages de la page d'accueil. C'est absolument normal, il lui manque la capacité `manage_options` pour gérer les réglages. Mais si vous lui donnez, c'est la porte ouverte à tout. Donc prudence !

`manage_options` est à mon sens l'une des capacités les plus importantes (donc dangereuse à donner), et elle doit être réservée aux administrateurs du site. Point final.

Vu que les capacités pour un rôle ne devraient être ajoutées qu'une seule fois, autant les ajouter dès la création du rôle personnalisé, comme dans notre fonction `wpcookbook_add_role()`. Donc si vous vous rendez compte que vous avez oublié des capacités pour votre rôle, il faudra utiliser un `remove_role()` suivi d'un `add_role()`, comme expliqué précédemment.

Pour supprimer des capacités, on utilise simplement `remove_cap()`.

## Ajouter des capacités à un utilisateur

Pour ajouter des capacités à un utilisateur, on procède exactement comme pour les rôles. C'est-à-dire qu'on récupère l'instance de `WP_User` correspondant à l'utilisateur à modifier, et on utilise la méthode `add_cap()`

```
add_action( 'init', 'wpcookbook_add_user_cap' );
/**
 * Adds the 'manage_my_plugin' capability to chief editor
 */
function wpcookbook_add_user_cap(){
    $chief_editor = get_userdata(8);
    if ( $chief_editor && ! $chief_editor->has_cap( 'manage_my_plugin' ) ) {
        $chief_editor->add_cap( 'manage_my_plugin' );
    }
}
```

Ici, la capacité ajoutée ne correspond encore à rien, et n'est pas utilisée dans WordPress. Ce qui veut dire que vous pouvez parfaitement ajouter aussi vos capacités personnalisées. C'est d'ailleurs tout l'intérêt !

Pour enlever des capacités à un utilisateur, on utilise la méthode `remove_cap()`. Aussi, l'exemple introduit la méthode `has_cap()` qui permet de vérifier si l'utilisateur dispose de la capacité.

## Vérifier les rôles et capacités

Créer des rôles et ajouter des capacités c'est bien beau, mais comment ça marche concrètement ?

À de nombreux moments lors de l'exécution du code, WordPress vérifie si vous avez les capacités nécessaires pour la tâche en question. Si non, vous aurez un message d'erreur ou rien ne se passera.

Par exemple, il ne montre certains éléments du menu de l'administration qu'aux personnes ayant les bonnes capacités. Il ne donne accès aux réglages qu'aux utilisateurs ayant `manage_options`. Sur les listes de publications, il va aussi vérifier les droits utilisateurs pour afficher les différentes actions possibles. Les capacités sont utilisées littéralement partout dans WordPress.

Pour vérifier qu'un utilisateur dispose de certaines capacités, vous pouvez utiliser la fonction `user_can()`.

```
user_can( int|WP_User $user, string $capability )
```

La fonction prend un identifiant d'utilisateur, et une chaîne représentant la capacité à vérifier, et retourne un booléen nous disant si oui ou non l'utilisateur en question dispose de cette capacité.

Au lieu de récupérer manuellement l'utilisateur courant, on peut utiliser `current_user_can( string $capability )` qui va directement tester l'utilisateur connecté, et retourner `false` s'il ne dispose pas de la capacité, ou si aucun utilisateur n'est connecté. Très pratique.

Il y a des chances pour que vous ayez plus souvent recours à `current_user_can()` que `user_can()`.

Même si on utilise `current_user_can()` avec un seul paramètre le plus souvent, on peut aussi l'utiliser en passant une méta-capacité en premier paramètre et un deuxième paramètre correspondant à l'identifiant de l'objet sur lequel on veut vérifier que l'utilisateur ait les droits.

Pardon ? Par exemple, on peut utiliser `current_user_can( 'edit_post', $post_id )` pour vérifier que l'utilisateur courant a bien le droit d'éditer la publication ayant pour identifiant `$post_id`. Or, `edit_post` (au singulier) n'est pas une capacité listée dans celles des `editor` ou `author`.

C'est parce que c'est une méta-capacité. C'est une capacité dérivée de `edit_posts`, si vous préférez. Il existe un certain nombre de méta-capacités qui sont associées à des capacités primitives : `edit_post`, `delete_post`, `edit_user`, `delete_user`, `edit_post_meta`, `delete_post_meta`, etc. qui permettent donc d'avoir un regard et contrôle plus précis sur les droits des utilisateurs.

Retenez simplement que vous pouvez vérifier les capacités pour un élément précis (publication ou utilisateur) en passant son identifiant et la méta-capacité correspondante (en général, c'est la forme singulier de la capacité primitive).

Si l'on revient à notre tableau d'utilisateurs (si si, vous savez, notre code court du début du chapitre), on peut décider de ne le montrer qu'aux administrateurs.

```
/**  
 * User Table shortcode display callback  
 *  
 * @param array $atts Shortcode attributes  
 */  
function wpcookbook_roles( $atts ) {  
    $users = get_users();  
    ob_start();  
    if( current_user_can( 'administrator' ) ) :  
    ?>  
        <table style="table-layout:fixed; word-break:>  
            ...  
        </table>  
    <?php else : ?>  
        <p><?php _e( 'Sorry, you are not allowed to view this content.', '22-roles-  
capabilities' ); ?></p>  
        <?php  
        endif;  
        return ob_get_clean();  
    }  
}
```

On utilise `current_user_can()` pour vérifier que l'utilisateur courant a bien `administrator` dans la liste de ses capacités. Si non, on affiche un message d'erreur à la place du tableau des utilisateurs. Si on jette un œil au tableau, en étant connecté avec notre `chief-editor`, on peut voir qu'il n'a pas accès.

WPCookBook — Apprenez à développer pour WordPress

Page Article Catégorie Contact

## Roles

Désolé, vous n'avez pas les autorisations nécessaires pour voir ce contenu.

Modifier

Notre éditeur en chef n'a pas accès au tableau.

Dans la pratique, on va éviter de vérifier les rôles directement et plutôt vérifier des capacités plus précises

réservées à certains rôles. Ici, il vaut mieux donc utiliser `manage_options`. C'est une capacité que seuls les administrateurs et super-admins ont. Pour un éditeur, on pourra vérifier `edit_others_posts` par exemple.

Il est indispensable de vérifier les droits utilisateurs dans vos développements. `current_user_can()` est donc votre arme principale. Mais attention ! Cette fonction vérifie uniquement que votre utilisateur a les droits pour effectuer une action, mais pas qu'il l'a effectivement effectué ! Ça c'est le rôle des nonces (jetons de sécurité). Pas de panique, on verra tout ça plus en détails plus tard, dans le chapitre Comment créer une metabox avec des champs personnalisés.

## Capacités et multisite

Si vous devez vérifier les droits d'un utilisateur sur un autre site d'un réseau multisite, vous devez utiliser `current_user_can_for_blog( int $blog_id, string $capability )`, en passant l'identifiant du site pour lequel vous voulez vérifier les droits de l'utilisateur.

En effet, un même utilisateur peut être inscrit sur un site en tant qu'`editor`, et n'être que `contributor` sur un autre, par exemple.

## À retenir

- WordPress met 5 (ou 6) rôles de base à notre disposition : `subscriber` (abonné), `contributor` (contributeur), `author` (auteur), `editor` (éditeur), `administrator` (administrateur) et `super-admin` (seulement sur multisite)
- Chaque rôle a ses capacités, comme `read`, `edit_posts` ou `switch_themes`.
- Les rôles sont organisés de façon hiérarchique et chaque rôle hérite des capacités du rôle inférieur.
- On peut ajouter/supprimer de nouveaux rôles utilisateurs grâce aux fonctions `add_role()` et `remove_role()`. Les rôles sont inscrits en base de données, donc on effectue ces opérations qu'une seule fois, dans la mesure du possible.
- On peut ajouter/supprimer des rôles à un utilisateur en utilisant les méthodes `add_role()` et `remove_role()` de l'instance de `WP_User` correspondante.
- On peut ajouter/supprimer des capacités à un rôle ou à un utilisateur en utilisant les méthodes `add_cap()` et `remove_cap()` des instances `WP_Role` ou `WP_User` correspondantes.
- On vérifie les rôles des utilisateurs avec `user_can()` ou `current_user_can()` en passant la capacité que l'on veut vérifier.

Vérifier les droits des utilisateurs est essentiel. Vous en aurez toujours besoin, mais encore une fois, WordPress met tout à notre disposition pour le faire simplement !

# Comment créer un code court

Pour des raisons de sécurité, on ne peut pas insérer un bout de code PHP dans le contenu d'un article ou d'une page. Un **code court** (ou shortcode) est un moyen d'insérer du contenu dynamique dans le contenu statique de vos pages ou autres publications.

Un code court se présente sous la forme [mon\_super\_code\_court]. Un code court étant simplement une macro appelant une fonction PHP, vous pouvez faire un peu ce que vous voulez à l'intérieur. Par exemple, beaucoup d'extensions nécessitent la création de pages spéciales pour leurs fonctionnalités et souvent, il suffit de créer une nouvelle page avec un simple code court pour que la magie opère !

Prenez WooCommerce, par exemple. Les pages du panier et de validation de la commande sont entièrement gérées par de simples codes courts. On peut difficilement faire plus simple.

## Créer un code court

Créer un code court est ultra simple, et on l'a déjà fait plusieurs fois au long de ce guide. Il suffit d'utiliser la fonction `add_shortcode()`. On va juste aller plus loin dans ce chapitre et voir les codes courts plus en profondeur.

La fonction `add_shortcode()` est disponible très tôt lors du chargement de WordPress, tout comme `add_action()` ou `add_filter()`. Ce qui signifie que l'on peut parfaitement l'utiliser nue (c'est-à-dire hors d'une fonction de rappel hookée), comme on utilise `add_action()` ou `add_filter()` nues pour hooker nos fonctions.

Cela dit, de manière générale, que ce soit dans un thème ou une extension, on n'exécute pas de code directement mais on programme l'exécution de nos fonctions sur les hooks de WordPress. Les fonctions que l'on peut utiliser directement dans un thème ou une extension sont rares et `add_action()`, `add_filter()` et `add_shortcode()` en font partie.

Dites-vous juste que **par défaut**, on hooke tout. Toujours. Pas de code nu.

```

// In main plugin file
add_shortcode( 'wpcookbook_simple_shortcode', 'wpcookbook_simple_shortcode' );
/**
 * Our simple shortcode example callback function
 *
 * @param array $atts      Array of shortcode attributes
 * @param string $content Content between shortcode tags.
 * @param string $tag      Shortcode tag itself
 * @return string          String of html content
 */
function wpcookbook_simple_shortcode( $atts = array(), $content = '', $tag =
'wpcookbook_simple_shortcode' ){
    ob_start();
    ?>
        <strong><?php _e( 'Shortcode attributes :', '23-shortcodes' ); ?></strong>
        <pre><?php var_dump( $atts ); ?></pre>
        <strong><?php _e( 'Shortcode content :', '23-shortcodes' ); ?></strong>
        <pre><?php var_dump( $content ); ?></pre>
        <strong><?php _e( 'Shortcode tag :', '23-shortcodes' ); ?></strong>
        <pre><?php var_dump( $tag ); ?></pre>
    <?php
    return ob_get_clean();
}

```

add\_shortcode( string \$tag, callable \$callback )

La fonction add\_shortcode() est très simple. On lui passe le nom de notre code court \$tag et le nom de la fonction de rappel qui lui est associée. Donc dans notre exemple, on déclare un code court appelé wpcookbook\_simple\_shortcode, qui a pour fonction de rappel une fonction du même nom.

Quand WordPress tombera sur le nom de notre code court entre crochets ([wpcookbook\_simple\_shortcode]) dans le contenu d'une page ou autre publication, il exécutera notre fonction de rappel, et y collera ce que la fonction lui renvoie. C'est pourquoi il faut absolument que la fonction de rappel renvoie une chaîne ! Les codes courts fonctionnent un peu comme les filtres. Il est en quelque sorte remplacé par la valeur de retour de la fonction de rappel.

Jusqu'à maintenant, on a passé mais jamais utilisé le paramètre \$atts de la fonction de rappel. En réalité, la fonction de rappel prend trois paramètres.

- \$atts est un tableau des attributs passés au code court. Ce sont ses paramètres, en quelque sorte. Plus de détails sur ce point dans la section suivante.
- \$content est le contenu du code court, si le code court est de type balise. Pas de panique, on y vient.
- \$tag est le nom du code court lui-même. Facile.

Notre fonction de rappel wpcookbook\_simple\_shortcode() va pour le moment juste afficher les valeurs de ses trois paramètres. Pardon. Elle ne va rien afficher du tout. Elle va retourner une chaîne de caractères contenant l'HTML nécessaire pour afficher les valeurs de ses trois paramètres, grâce à ob\_start() et ob\_get\_clean().

Ces fonctions sont très pratiques. ob\_start() va permettre de ne rien afficher, mais de mettre tout ce qui devrait être affiché ensuite en mémoire tampon. Puis on retourne le contenu du tampon et on le vide avec ob\_get\_clean(). Personnellement, je les utilise beaucoup quand je dois afficher de l'HTML dans une

fonction PHP.

WPCookBook — Apprenez à développer pour WordPress

[Page](#) [Article](#) [Catégorie](#) [Contact](#)

---

# Shortcode

**Shortcode attributes :**

```
string(0) ""
```



**Shortcode content :**

```
string(0) ""
```

**Shortcode tag :**

```
string(27) "wpcookbook_simple_shortcode"
```

[Modifier](#)

Le code court affiche simplement les paramètres de sa fonction de rappel.

## Les attributs des codes courts

Pour personnaliser le contenu de notre code court, on peut passer des attributs dans le code court. On les passe simplement entre les crochets, sous la forme clé="valeur". Par exemple, si on retourne dans l'édition de notre page, et qu'on insère un attribut `title` à notre code court, comme ceci :

```
[wpcookbook_simple_shortcode title="Exemple de code court"]
```

On peut voir que sur le devant du site, notre attribut apparaît dans le tableau affiché. On peut ensuite récupérer ce titre dans la fonction de rappel du code court, pour l'afficher ensuite.

```
// in functions.php
function wpcookbook_simple_shortcode( $atts = array(), $content = '' , $tag =
'wpcookbook_simple_shortcode'){
    ob_start();
    ?>
    <?php if( ! empty( $atts['title'] ) ): ?>
        <h2><?php echo esc_html( $atts['title'] ); ?></h2>
    <?php endif; ?>
    ...
    <?php
    return ob_get_clean();
}
```

# Shortcode

## Exemple de code court

**Shortcode attributes :**

```
array(1) {  
    ["title"]=>  
    string(21) "Exemple de code court"  
}
```



**Shortcode content :**

```
string(0) ""
```

**Shortcode tag :**

```
string(27) "wpcookbook_simple_shortcode"
```

[Modifier](#)

Le code court affiche bien notre titre.

On peut passer autant d'attributs que nécessaire. Si vous voulez ajouter dix attributs, pas de souci. Ils se retrouveront tous dans le même tableau \$atts.

### Attributs par défaut

Dans l'exemple, j'ai été prudent et j'ai vérifié qu'il y avait bien un titre donné en attribut. Mais s'il n'y en a pas, on a peut-être besoin d'inclure un titre par défaut.

Pour ce faire, on va utiliser la fonction `shortcode_atts( array $pairs, array $atts, string $shortcode = '' ).`

Elle prend un tableau \$pairs de tous les attributs supportés par le shortcode et de leur valeur par défaut, les attributs passés dans le code court \$atts, et le nom de notre code court, qui peut être passé comme

contexte. En effet, le retour de la fonction est filtrable (via le hook `shortcode_atts_{$shortcode}`) donc cela peut être utile de passer le nom du code court aux fonctions hookées sur ce filtre.

La fonction va retourner un tableau en fusionnant nos attributs par défaut avec les attributs passés au code court. C'est très pratique.

```
function wpcookbook_simple_shortcode( $atts = array(), $content = '', $tag = 'wpcookbook_simple_shortcode' ){

    $defaults = array(
        'title' => __( 'Default title', '23-shortcodes' ),
    );
    $atts = shortcode_atts( $defaults, $atts, $tag );

    ob_start();
    ?>
        <h2><?php echo esc_html( $atts['title'] ); ?></h2>
        ...
    <?php
    return ob_get_clean();
}
```

On déclare des attributs et leur valeur par défaut, et on fusionne ce tableau avec `$atts`. Plus besoin de vérifier qu'on a bien un attribut `title` car on en aura un dans tous les cas. Easy.

Si on supprime notre attribut `title` dans le contenu de notre page, alors on a bien quand même un titre.

# Shortcode

## Default title

**Shortcode attributes :**

```
array(1) {  
    ["title"]=>  
        string(13) "Default title"  
}
```



**Shortcode content :**

```
string(0) ""
```

**Shortcode tag :**

```
string(27) "wpcookbook_simple_shortcode"
```

[Modifier](#)

Le titre par défaut s'affiche si on ne passe pas d'attributs.

## Codes courts doubles

On peut aussi créer des codes courts qui fonctionnent comme des balises HTML et qui vont donc encadrer du contenu. Dans notre éditeur de contenu, il suffit d'encadrer le contenu que l'on souhaite avec des codes courts [mon\_code\_court] et [/mon\_code\_court].

Par exemple, imaginons que nous voulions mettre en avant un paragraphe ou bloc de contenu, en le contenant dans une <div> avec des styles particuliers. Créons un nouveau code court.

```

// In main plugin file
add_shortcode( 'wpcookbook_enclosing_shortcode', 'wpcookbook_enclosing_shortcode' );
/** 
 * A simple enclosing shortcode example.
 *
 * @param array $atts      Array of shortcode attributes
 * @param string $content  Content between shortcode tags.
 * @param string $tag      Shortcode tag itself
 * @return string          String of html content
 */
function wpcookbook_enclosing_shortcode( $atts, $content, $tag ){

    if( empty( $content ) ){
        return __( 'You forgot to include content to emphasize!', '23-shortcodes' );
    }

    ob_start();
    ?>
        <div style="background-color: yellow; padding: 1.5rem; font-size: 1.125em;"><?php echo
$content; ?></div>
    <?php
    return ob_get_clean();
}

```

Le contenu entre nos codes courts balises est passé à la fonction de rappel via le paramètre \$content. On vérifie donc que notre contenu n'est pas vide. Si oui, on retourne un message d'erreur à afficher. Concrètement, il vaudrait peut-être mieux ne rien faire du tout, mais c'est pour l'exemple.

S'il y a du contenu, on l'encadre dans une magnifique div avec une couleur d'arrière plan jaune. Encore une fois, désolé pour le style en dur, mais pour l'exemple, cela suffira.

Testez le code court en incluant uniquement [wpcookbook\_enclosing\_shortcode]  
[/wpcookbook\_enclosing\_shortcode] dans l'éditeur de contenu, puis quelque chose comme  
[wpcookbook\_enclosing\_shortcode]Contenu à mettre en avant!   
[/wpcookbook\_enclosing\_shortcode]

WPCookBook — Apprenez à développer pour WordPress  
[Page](#) [Article](#) [Catégorie](#) [Contact](#)

## Shortcode

Contenu à mettre en avant !

Modifier



Code court balise affichant une boîte texte jaune. Magnifique.

## Codes courts imbriqués

Que se passe-t-il si on imbrique des codes courts ? Par exemple, si on utilise  
[wpcookbook\_enclosing\_shortcode][wpcookbook\_simple\_shortcode]  
[/wpcookbook\_enclosing\_shortcode] ?

Il n'est pas interprété, simplement. Il faut donc utiliser la fonction `do_shortcode()` sur la valeur de retour de notre code court balise pour que les codes courts internes soient interprétés. La fonction prend deux paramètres : le contenu dans lequel il faut appliquer les code courts (`$content`), et un booléen `$ignore_html` qui vaut `false` par défaut et qui permet d'ignorer les codes courts dans des balises HTML standards.

On va ajuster notre fonction de rappel comme suit:

```
function wpcookbook_enclosing_shortcode( $atts, $content, $tag ){
    ...
    ob_start();
    ...
    return do_shortcode( ob_get_clean() );
}
```

## Shortcode

### Default title

**Shortcode attributes :**

```
array(1) {  
    ["title"]=>  
        string(13) "Default title"  
}
```

**Shortcode content :**

```
string(0) ""
```

**Shortcode tag :**

```
string(27) "wpcookbook_simple_shortcode"
```

[Modifier](#)

Les codes courts imbriqués fonctionnent.

Et que se passe-t-il quand on imbrique les mêmes codes courts ? Par exemple

[wpcookbook\_enclosing\_shortcode] Ce texte est à mettre en avant ?

[wpcookbook\_enclosing\_shortcode] Ou celui-ci ?[/wpcookbook\_enclosing\_shortcode] ? Il lui manque une balise fermante, donc il considère le deuxième [wpcookbook\_enclosing\_shortcode] vide de contenu, et retourne le message d'erreur.

# Shortcode

Ce texte est à mettre en avant ?You forgot to include  
content to emphasize !Ou celui-ci ?

Modifier



Les mêmes codes imbriqués fonctionnent aussi, presque.

Attention cependant, on voit bien ici que WordPress considère qu'il n'y a qu'une seule paire de code balise. C'est-à-dire qu'il considère qu'il n'y a qu'un seul code court avec pour contenu "Ce texte est à mettre en avant ?[wpcookbook\_enclosing\_shortcode]Ou celui-ci ?", et pas un [wpcookbook\_enclosing\_shortcode] vide suivi du texte "Ce texte est à mettre en avant ?" puis un code court avec "Ou celui-ci ?" pour contenu.

Donc imbriquer des codes courts peut donner des résultats inattendus Si vous pouvez éviter, faites-le !

## Codes courts dans un modèle de page

Parfois, vous aurez besoin d'intégrer le contenu d'un code court dans un modèle de page directement. Certaines extensions permettent d'ajouter des fonctionnalités utiles, comme des filtres pour vos articles par exemple, mais le code court ne peut pas être mis dans le contenu de la page. Il faut ouvrir un modèle de votre thème enfant, et y insérer le code court.

En réalité, vous savez déjà comment faire ! On utilise simplement la fonction `do_shortcode()`, en lui passant tous les attributs dont vous avez besoin. Mais puisque `do_shortcode()` ne fait que filtrer son contenu et y interpréter les codes courts, il faut simplement l'afficher avec un echo.

```
<?php echo do_shortcode( '[wpcookbook_simple_shortcode title="Code hors contenu"]' ); ?>
```

Bon par contre, niveau performances un `do_shortcode()` est assez coûteux. Donc si vous connaissez le nom de la fonction de rappel, vous pouvez aussi l'utiliser directement.

```
<?php echo wpcookbook_simple_shortcode( array( 'title' => 'Code hors contenu' ) ); ?>
```

C'est parfaitement équivalent. Attention, j'ai omis les deuxième et troisième paramètres car la déclaration de ma fonction de rappel fournit une valeur par défaut pour ces derniers.

## Autres fonctions utiles

Si vous avez par hasard besoin de supprimer des codes courts enregistrés par une extension, vous pouvez le faire, en utilisant `remove_shortcode( $tag )` en lui passant le nom du code à supprimer. Il faudra mettre une priorité élevée sur votre fonction de rappel hookée pour le supprimer au bon moment, c'est-à-dire juste après qu'il soit déclaré avec `add_shortcode()`.

Vous pouvez vérifier qu'un code court existe avec `shortcode_exists( string $tag )`, qui vous renvoie un booléen selon si le code passé en paramètre existe.

Vous pouvez aussi vérifier qu'un certain contenu contient un code court en utilisant `has_shortcode( string $content, string $tag )`. La fonction prend en paramètre le contenu à vérifier et le code à chercher, et retourne un booléen selon si elle a trouvé le code dans le contenu. Simplement.

## Code court et sécurité

Un code court est un outil puissant qui permet, quand on y réfléchit bien, d'exécuter du code PHP dans le contenu de vos articles. C'est puissant, mais aussi dangereux. Quand vous déclarez un code court, il est disponible pour tout le monde, y compris le simple Contributeur qui écrit un article sur votre site. Certes, il n'a pas le droit de publier, mais il peut prévisualiser, donc exécuter votre code PHP.

Soyez donc prudent avec ce que vous permettez de faire avec votre code court. Un grand pouvoir implique une grande responsabilité.

## Ce qu'il faut retenir

- On déclare un code court avec `add_shortcode()`.
- La fonction de rappel d'un code court prend trois paramètres : `$atts`, `$content`, et `$tag`.
- Un code simple reçoit un `$content` vide. Un code de type balise reçoit le contenu entre ses balises dans `$content`.
- On peut passer n'importe quel nombre d'attributs dans les crochets du code court, sous la forme `clé="valeur"`. Un tableau avec ces attributs est passé à la fonction de rappel via `$atts`.
- On utilise `shortcode_atts()` pour déclarer un tableau d'attributs par défaut et les fusionner avec les attributs passés dans `$atts`.
- On peut imbriquer des codes courts mais il faut interpréter les codes "internes" en utilisant `do_shortcode()`. L'interprétation des codes courts n'est pas récursive par défaut.
- On peut interpréter et insérer le contenu d'un code court dans un modèle de page avec `echo do_shortcode()`.
- On peut aussi vérifier qu'un code court existe (`shortcode_exists()`), le supprimer (`remove_shortcode()`), ou scanner du contenu pour y déterminer si un certain code court y est présent (`has_shortcode()`).

Les codes courts sont des outils très puissants, grâce à leur flexibilité et leurs attributs. Créer un code court est bien plus simple que de développer un bloc dynamique pour le nouvel éditeur. Certes, c'est moins joli lors de l'édition du contenu, mais c'est tout aussi puissant. Ne vous privez pas !

# Comment créer un widget

Les widgets sont de petits blocs de contenu bien pratiques que vous pouvez ajouter aux zones de widgets prévues à cet effet dans votre thème. Si vous voulez apprendre à créer des zones de widgets (qui sont de la responsabilité des thèmes), rendez-vous au chapitre *Créer des zones de widgets* dans la section dédiée aux thèmes.

WordPress met à notre disposition plusieurs widgets utiles (Articles Récents, Catégories, etc.) dont le puissant widget HTML personnalisé qui permet de mettre directement du code HTML dans un widget. Il existe aussi beaucoup d'extensions qui proposent leurs widgets, mais parfois notre besoin n'est pas exactement comblé et il faut en créer un.

Créer un widget personnalisé est un peu plus complexe que créer un code court, mais pas de panique, car encore une fois WordPress met à notre disposition tous les outils dont nous avons besoin. Je sais, je dis ça à chaque chapitre.

## La classe `WP_Widget`

WordPress dispose d'une classe `WP_Widget` qu'il faudra étendre pour nos besoins. Le widget créé devant être disponible quelque soit le thème, on va créer une petite extension pour ce faire. Je pense que vous savez comment faire, maintenant.

Imaginons que vous ayez besoin de créer un widget affichant les derniers articles, mais de façon plus attractive que le widget fourni par WordPress. Il existe sûrement déjà un widget qui remplit ce besoin sur le répertoire, mais chut, on va faire semblant.

On va donc commencer par étendre la classe `WP_Widget` en créant une classe `WPCookbook_Recent_Posts_Widgets`. Créez un dossier `inc/` et un fichier `class-wpcookbook-recent-posts-widget.php` dans ce dossier. Puis, dans ce nouveau fichier, ajoutez :

```

// in class-wpcookbook-recent-posts-widget.php
/**
 * Class for our recent posts widget
 */
class WPCookbook_Recent_Posts_Widgets extends WP_Widget {
    /**
     * Constructor
     *
     * @param string $id_base      Optional Base ID for the widget, lowercase and unique.
     * If left empty,
     *                      a portion of the widget's class name will be used Has
     * to be unique.
     * @param string $name         Name for the widget displayed on the admin page.
     * @param array  $widget_options Optional. Widget options. See
     wp_register_sidebar_widget() for information
     *
     * @param array  $control_options Optional. Widget control options. See
     wp_register_widget_control() for
     *
     *                      information on accepted arguments. Default empty array.
     * @param array  $control_options Optional. Widget control options. See
     wp_register_widget_control() for
     *
     *                      information on accepted arguments. Default empty array.
     */
    public function __construct() {}

    /**
     * Outputs the widget on the front end
     *
     * @param array $args      Display arguments including 'before_title', 'after_title',
     *                      'before_widget', and 'after_widget'.
     * @param array $instance  The settings for the particular instance of the widget.
     */
    public function widget( $args, $instance ) {}

    /**
     * Outputs the settings form in the Widgets administration screen
     *
     * @param array $instance  Current settings.
     * @return string          Default return is 'noform'.
     */
    public function form( $instance ) {}

    /**
     * Saves our options
     *
     * @param array  $new_instance New settings for this instance.
     * @param array  $old_instance Old settings for this instance.
     * @return array            Settings to save or bool false to cancel saving
     */
    public function update( $new_instance, $old_instance ) {}
}

```

C'est toujours une bonne idée de créer un fichier par classe. Le nom du fichier est souvent préfixé par `class-`. C'est une convention utilisée dans WordPress, donc on va aussi l'utiliser.

La classe `WP_Widget` possède d'autres méthodes utiles et nous aurons besoin de certaines d'entre elles. Mais pour créer un widget, nous n'avons besoin de surcharger que `__construct()`, qui va être appelée à chaque création d'une instance du widget, et `widget()` qui est responsable de l'affichage du widget sur le devant du site.

Si le widget propose des réglages, alors il faut aussi surcharger les méthodes `form()`, qui affiche le formulaire de réglages dans l'administration de WordPress, et `update()` qui va s'occuper de sauvegarder nos réglages.

Notre classe `WPCookbook_Recent_Posts_Widgets` va hériter de toutes les propriétés et méthodes de son parent `WP_Widget`, sauf les 4 méthodes ci-dessus que nous allons surcharger.

## `__construct()`

`__construct()` est la méthode qui est appelée quand un widget est créé. Vous remarquerez qu'elle ne prend aucun paramètre, même si le commentaire de la fonction en présente 4. En fait, les paramètres présentés sont ceux de la classe parente `WP_Widget`, et notre `__construct()` va devoir appeler la méthode `__construct()` de son parent, en lui passant ce coup-ci des paramètres.

Parmi ces paramètres, on a :

- `$id_base` qui est l'identifiant unique du widget, auquel sera automatiquement ajouté un numéro pour identifier les différentes instances de notre widget,
- `$name` qui est le nom du widget affiché dans l'administration,
- `$widget_options` qui est un tableau d'informations complémentaires sur le widget,
- `$control_options` qui est un tableau d'options concernant l'interface du widget.

On va laisser de côté pour l'instant les paramètres `$widget_options` et `$control_options`, et déclarer un identifiant et un nom pour notre widget.

```
public function __construct() {
    parent::__construct( 'wpcookbook-recent-posts', __( 'WPCookBook Recent Posts', '24-widgets'
) );
}
```

On appelle donc le constructeur de notre parent, en lui passant un slug et une chaîne traduisible pour nom d'affichage.

## `widget()`

La méthode `widget()` est responsable de l'affichage de notre widget. Elle prend deux paramètres :

- `$args` est un tableau d'arguments. Ce sont les valeurs passées par la fonction `register_sidebar()` lors de la déclaration de la zone de widgets par le thème. En gros, ce sont les paramètres de la zone de widgets dans laquelle est inséré notre widget. Référez-vous au chapitre [Comment créer une zone de widgets](#) pour plus de détails.
- `$instance` contient tous les réglages de notre instance de widgets.

Pour faire simple, pour l'instant on va uniquement afficher un message :

```
public function widget( $args, $instance ) {
    echo $args['before_widget'];

    esc_html_e( 'This is our custom widget!', '24-widgets' );

    echo $args['after_widget'];
}
```

Pour que le balisage soit conforme à ce que le thème a déclaré à l'aide de `register_sidebar()`, on utilise `$args['before_widget']` et `$args['after_widget']` pour encadrer tout le contenu de notre widget, et on affiche simplement notre message pour le moment.

## register\_widget()

Notre widget est pour l'instant inutilisable, car il n'est pas enregistré. Tout ce qu'on a fait, c'est déclarer une classe.

On va donc se hooker sur `widgets_init` pour déclarer notre widget à l'aide de la fonction `register_widget()`. La fonction prend un seul paramètre, qui peut être le nom de la classe d'un widget, ou une instance de cette classe. On va donc lui passer `WPCookbook_Recent_Posts_Widgets`, en prenant soin de charger le fichier de classe avant. Dans notre bootstrap, ajoutez :

```
// in main plugin file
add_action( 'widgets_init', 'wpcookbook_register_widget' );
/**
 * Registers our new widget
 */
function wpcookbook_register_widget(){
    include 'inc/class-wpcookbook-recent-posts-widget.php';
    register_widget( 'WPCookbook_Recent_Posts_Widgets' );
};
```

Widgets Gérer avec l'aperçu en direct

**Widgets disponibles**

Pour activer un widget, glissez-le dans la colonne latérale ou cliquez dessus. Pour désactiver un widget et supprimer ses réglages, enlevez-le de la colonne latérale.

|                      |                         |
|----------------------|-------------------------|
| Archives             | Articles récents        |
| Calendrier           | Catégories              |
| Commentaires récents | Flux                    |
| Galerie              | HTML personnalisé       |
| Image                | Menu de navigation      |
| Méta                 | Nuage d'étiquettes      |
| Pages                | Rechercher              |
| Son                  | Texte                   |
| Vidéo                | WPCookBook Recent Posts |

**Pied de page**

Ajoutez ici des widgets qui apparaîtront dans votre pied de page.

|                         |
|-------------------------|
| Rechercher              |
| WPCookBook Recent Posts |

[Supprimer](#) | [Terminé](#) [Enregistrer](#)

Notre widget est disponible dans l'administration

WPCookBook — Apprenez à développer pour WordPress  
[Page](#) [Article](#) [Catégorie](#) [Contact](#)

## Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

vincent | septembre 12, 2019 | Uncategorized | Un commentaire | [Modifier](#)

This is our custom widget !

WPCookBook, Fièrement propulsé par WordPress.

Notre widget s'affiche correctement sur le devant du site

## form()

Notre widget fonctionne, mais il n'a même pas un titre paramétrable. On va donc lui ajouter un formulaire pour ses réglages. On fait ça en surchargeant la méthode `form()` de la classe `WP_Widget`.

```
// in class-wpcookbook-recent-posts-widget.php
public function form( $instance ) {
    $title = ! empty( $instance['title'] ) ? $instance['title'] : '';
    ?>
    <pre><?php var_dump( $instance ); ?></pre>
    <p>
        <label for="<?php echo esc_attr( $this->get_field_id( 'title' ) ); ?>">
            <?php esc_html_e( 'Title :', '24-widgets' ); ?>
        </label>
        <input
            type="text"
            class="widefat"
            id="<?php echo esc_attr( $this->get_field_id( 'title' ) ); ?>"
            name="<?php echo esc_attr( $this->get_field_name( 'title' ) ); ?>"
            value="<?php echo esc_attr( $title ); ?>">
        </p>
    <?php
}
```

La méthode `form()` prend `$instance` en paramètre, qui est un tableau contenant tous les réglages de l'instance courante, donc normalement vide lors de la création du widget. Pour que ce soit plus pratique, on va utiliser `var_dump()` pour inspecter la valeur de `$instance`, temporairement.

Dans la fonction, on utilise un simple `<p>` avec un champ de formulaire `<input>` et son `<label>`. On utilise juste la classe CSS `widefat` de la feuille de styles de l'administration de WordPress pour faire que notre champ occupe 100% de la largeur du conteneur. Rien d'extraordinaire, si ce n'est l'utilisation des méthodes `get_field_id()` et `get_field_name()`.

Ces deux méthodes sont très importantes. Dans le constructeur, on a passé un identifiant de base, auquel est préfixé un numéro en fonction de l'instance du widget. Ces deux méthodes permettent de récupérer les bonnes valeurs pour les attributs `name` et `id` en fonction de cet identifiant et du numéro de l'instance du widget.

En effet, si vous codez en dur ce `name`, vous pourrez vous retrouver avec plusieurs `<input>` avec des attributs `name` identiques si le widget est utilisé plusieurs fois, ce qui va poser des soucis lors de la sauvegarde.

Faites le test pour voir ! Codez un `name` en dur, ajoutez deux ou trois exemplaires de notre widget et inspectez les champs dans les outils de développement du navigateur. Ce n'est pas joli !

Aussi, on utilise les fonctions `esc_html()` et `esc_attr()` pour s'assurer que les chaînes passées s'affichent sans risque entre des balises HTML ou dans les attributs des balises.

## update()

Notre formulaire s'affiche, mais si on entre une valeur, elle n'est pas sauvegardée. C'est le rôle de notre méthode update().

La méthode prend deux paramètres : \$new\_instance qui est l'instance du widget avec ses nouveaux réglages (les valeurs soumises par l'utilisateur), et \$old\_instance qui est l'ancienne instance du widget (avant sauvegarde, donc).

Il convient de faire un minimum de travail sur les données fournies par l'utilisateur. On va donc au moins s'assurer que son texte est sain.

```
// in class-wpcookbook-recent-posts-widget.php
public function update( $new_instance, $old_instance ) {

    if( $new_instance === $old_instance ){
        return false;
    }

    $instance = array(
        'title' => ! empty( $new_instance['title'] ) ? sanitize_text_field(
            $new_instance['title'] ) : '',
    );

    return $instance;
}
```

Tout d'abord, si aucun réglage n'a été changé, on peut retourner `false` pour annuler la sauvegarde. S'il y a eu changement, alors on va recréer le tableau des réglages, en prenant la valeur du `title` de la `$new_instance`, en s'assurant qu'il ne soit pas vide puis en le nettoyant à l'aide de la fonction `sanitize_text_field()`.

La fonction `sanitize_text_field()` est extrêmement utile. C'est une fonction outil de WordPress qui va nettoyer une chaîne en vérifiant l'encodage UTF-8, changeant les < par leur entité, enlevant les balises HTML, etc...

WordPress met à notre disposition plusieurs fonctions de ce type pour nous aider à traiter les données utilisateur : `sanitize_email`, `sanitize_key`, `sanitize_textarea_field`, etc. On sera amené à en utiliser plusieurs dans les chapitres suivants.

Notez qu'au lieu d'écrire :

```
$instance = array(
    'title' => ! empty( $new_instance['title'] ) ? sanitize_text_field( $new_instance['title']
) : '',
);
```

on aurait pu aussi écrire :

```

if( ! empty( $new_instance['title'] ) ){
    $title = sanitize_text_field( $new_instance['title'] );
} else {
    $title = '';
}

$instance = array(
    'title' => $title,
);

```

L'opérateur ternaire `a ? b : c`; peut se lire `if a, then b, else c` en pseudo-code. La notation ternaire permet des raccourcis utiles. C'est tellement plus pratique.

Si vous faites le test dans l'admin, notre titre est bien nettoyé et sauvegardé. Cool. Maintenant, il nous reste à afficher notre titre, donc ajustons notre méthode `widget()`.

```

// in class-wpcookbook-recent-posts-widget.php
public function widget( $args, $instance ) {
    echo $args['before_widget'];

    if ( ! empty( $instance['title'] ) ) {
        echo $args['before_title']
            . apply_filters( 'widget_title', $instance['title'], $instance, $this->id_base )
            . $args['after_title'];
    }

    esc_html_e( 'This is our custom widget! ', '24-widgets' );

    echo $args['after_widget'];
}

```

On vérifie simplement que le titre de l'instance n'est pas vide, et on l'affiche entre les balises `before_title` et `after_title` fournies par `register_sidebar()`, tout comme on a fait pour le widget lui-même. Pour rappel, `$args` contient les arguments de la zone de widgets dans laquelle est affiché notre widget, et qui sont donnés par le thème quand il déclare ladite zone de widgets via `register_sidebar()`.

On va prendre le soin d'appliquer le filtre `widget_title` sur le titre de l'instance. Ce filtre est appliqué sur tous les widgets natifs de WordPress, il convient donc de permettre aux extensions utilisant ce filtre d'affecter également le titre de notre widget si c'est nécessaire. C'est la moindre des politesses.

Les utilisateurs du filtre vont avoir besoin du titre en question `$instance['title']`, des réglages de l'instance `$instance`, et de l'identifiant du widget comme contexte, `$this->id_base`. On les passe donc.

Si on revient sur le devant du site, pas de souci, notre titre s'affiche correctement.

# Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

 vincent  septembre 12, 2019  Uncategorized  Un commentaire  Modifier



WPCookBook, Fièrement propulsé par WordPress.

Notre widget affiche bien son titre. Ça avance.

## Retour sur `__construct()`

Si vous vous souvenez bien, on avait laissé de côté deux paramètres de notre constructeur : `$widget_options` et `$control_options`.

Pour trouver les valeurs possibles pour ces paramètres, il faut se référer aux fonctions `wp_register_sidebar_widget()` et `wp_register_widget_control()` respectivement, car ce sont elles qui sont appelées en coulisse pour enregistrer notre widget.

Pour `widget_options`, on peut passer un tableau contenant un nom de classe CSS (`classname`) qui sera utilisé sur le devant du site et une description détaillée qui apparaîtra dans l'administration (`description`). Si on omet la classe CSS, une classe basée sur l'identifiant du widget sera ajoutée automatiquement. Et c'est parfaitement acceptable.

Aussi, on peut ajouter '`customize_selective_refresh`' pour permettre à l'outil de personnalisation de WordPress de ne rafraîchir que le widget dans l'aperçu quand ses options sont modifiées.

Pour `control_options`, on peut passer un tableau contenant la hauteur du bloc widget dans l'administration (`height`), sa largeur (`width`), et on peut surcharger son identifiant en passant un autre `id_base`.

Concrètement, et c'est très bizarre pour le coup, `height` n'est jamais utilisé dans l'administration, et je ne vois pas pourquoi on surchargerait l'identifiant du widget qu'on a déclaré dans `__construct()`. Ou alors j'ai râté un truc. C'est possible.

Notre `__construct()` devient donc :

```
public function __construct() {
    parent::__construct(
        'wpcookbook-recent-posts', // $id_base
        __( 'WPCookBook Recent Posts', '24-widgets' ), // $display_name
        array(
            // 'classname'                => 'widget_wpcookbook-recent-posts', // Default
            'value, which is ok.'       => __( 'Displays attractive recent post cards', '24-
widgets' ), // Description in the admin area
            'customize_selective_refresh' => true,
        ),
        array(
            // 'height' => 500, // Default 200, but never used in the admin (?!)
            'width'  => 500, // Default 250
            // 'id_base' => 'wpcookbook-recent-posts', // Defaults to widget's id_base, which
is perfectly fine.
        ),
    );
}
```

The screenshot shows the WordPress dashboard with the 'Widgets' page open. On the left, there's a sidebar with various menu items like 'Tableau de bord', 'Articles', 'Médias', 'Pages', 'Commentaires', 'Apparence' (which is selected), 'Extensions', 'Utilisateurs', 'Outils', 'Règles', and 'Réduire le menu'. The main content area has a title 'Widgets' with a sub-link 'Gérer avec l'aperçu en direct'. Below it, there's a section titled 'Widgets disponibles' containing a grid of 15 widget options. One of these, 'Recent Posts', is currently selected and expanded. Its configuration panel shows the following settings:

- Title:** Articles récents
- Description:** Displays attractive recent post cards
- HTML:**

```
array(1) { ["title"]=> string(17) "Articles récents" }
```
- Buttons:** Supprimer | Terminé | Enregistrer

A yellow arrow points from the text 'Displays attractive recent post cards' back to the corresponding line in the code snippet above.

Notre widget est plus large (youpi...) et dispose maintenant d'une description (ça c'est cool).

Ce qui est rigolo, c'est que si l'on passe des valeurs plus grandes pour `width` et `height`, l'affichage est tout funky dans l'outil de personnalisation de WordPress. Si vous voulez mon avis, mis à part pour ajouter

une description, tous ces réglages ne servent à rien. Mais ce n'est que mon avis.

## **Vous savez tout !**

Si vous avez suivi jusqu'ici, vous savez tout. Du moins, vous avez tous les outils nécessaires pour commencer à développer vos widgets.

Mécaniquement parlant, **vous savez tout ce qu'il y a à savoir pour créer vos widgets**. Vous savez les déclarer dans WordPress et savez quelles méthodes surcharger pour créer des réglages, les sauvegarder et afficher votre widget.

À partir de maintenant, on va juste ajouter la fonctionnalité d'affichage des articles récents, avec un réglage ou deux pour s'entraîner.

## Peaufinons notre widget

Si vous vous souvenez, on avait décidé que ce widget afficherait des articles récents de façon plus attractive qu'une simple liste. On va les afficher simplement sous forme de petite carte. On va commencer par mettre en place notre contenu dans la méthode `widget()`.

```
public function widget( $args, $instance ) {
    echo $args['before_widget'];

    if ( ! empty( $instance['title'] ) ) {
        echo $args['before_title'] . apply_filters( 'widget_title', $instance['title'],
$instance, $this->id_base ) . $args['after_title'];
    }

    $query_args = array(
        'posts_per_page' => 3,
    );

    /**
     * Filters the wpcookbook_recent_post_widget's query arguments.
     *
     * @param array $query_args The arguments passed in to the new WP_Query
     * @param array $instance The widget settings
     * @return array The query arguments
     */
    $query_args = apply_filters( 'wpcookbook_recent_posts_widgets_query_args', $query_args,
$instance );

    $query = new WP_Query( $query_args );

    if( $query->have_posts() ){
        echo '<div class="recent-posts-widget-cards">';
        while ( $query->have_posts() ){
            $query->the_post();
            the_title();
            echo '<br>';
        }
        echo '</div>';
    }

    wp_reset_postdata();
}

echo $args['after_widget'];
}
```

On va créer une nouvelle `WP_Query` pour chercher nos 3 derniers articles et on va utiliser une boucle de WordPress, comme dans un modèle de page standard, pour les afficher. Cela nous permettra d'utiliser les templates tags natifs, comme `the_title()`.

`WP_Query` est une classe très puissante car elle donne énormément de contrôle sur le contenu que l'on va aller chercher. Si c'est la première fois que vous rencontrez `WP_Query` dans ce guide, je vous renvoie au chapitre [Comprendre la boucle de WordPress](#) pour plus de détails sur `WP_Query`.

Ici, on passe juste un tableau d'arguments contenant le nombre d'articles à aller chercher, soit trois. Et on

permet aux utilisateurs de notre widget qui veulent plus de contrôle de modifier les arguments de notre `WP_Query` avec un filtre, que l'on documente convenablement.

Puis on utilise une boucle classique. On vérifie qu'il y ait des articles à afficher avec `$query->have_posts()`, et si oui, on boucle dessus et on en affiche le titre seulement avec `the_title()`, pour le moment.

Enfin, on restore la globale `$post` à sa valeur courante dans la boucle principale avec `reset_postdata()`.

Si tout cela n'est pas très clair, je vous recommande vraiment de lire le chapitre *Comprendre la boucle de WordPress* dans lequel vous allez vraiment comprendre comment fonctionne la boucle et comment en créer de nouvelles.

Notez aussi que les arguments de la `WP_Query` sont appelés `$query_args` et pas simplement `$args`. Tout simplement parce que sinon, vous allez surcharger les arguments `$args` du widget ! Ca paraît bête, mais j'ai tellement l'habitude d'appeler ce tableau `$args`, que j'ai fait l'erreur pendant l'écriture de ce chapitre ! Donc je me suis dit que le dire pouvait être utile. Ahem.

---

## Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

 vincent  septembre 12, 2019  Uncategorized  Un commentaire  Modifier



A screenshot of a WordPress dashboard. At the top left is a search bar with the placeholder "Recherche...". Below it is a blue button labeled "Rechercher". To the right is a sidebar with a yellow border containing a section titled "Articles récents" with three items: "Hello Galaxy !", "Hello Universe !", and "Hello world!". At the bottom of the sidebar is a footer note: "WPCookBook, Fièrement propulsé par WordPress."

WPCookBook, Fièrement propulsé par WordPress.

Notre widget affiche bien les titres de nos articles.

## Un peu de nettoyage

Pour plus de clarté, on va inclure dans notre bootstrap la définition de constantes utiles que l'on a déjà vues au chapitre "Comment ajouter des hooks dans vos développements", ainsi que de la fonction `wpcbook_locate_template()` qui va nous permettre de charger notre modèle partiel pour nos

articles, en allant le chercher dans notre thème enfant en priorité, puis dans le parent, sinon dans le dossier templates de notre extension... que nous devons encore créer.

```
define( 'WIDGETS_PATH', plugin_dir_path( __FILE__ ) );
define( 'WIDGETS_URL', plugin_dir_url( __FILE__ ) );
...

/**
 * Returns the path to a template file.
 * Looks first if the file exists in the `wpcookbook/` folder in the child theme,
 * then in the parent's theme `wpcookbook/` folder,
 * finally in the default plugin's template directory
 *
 * @param string $template_name The template we're looking for
 * @return string $located      The path to the template file if found.
 */
function wpcookbook_locate_template( $template_name ) {
    $located = '';

    if ( file_exists( STYLESHEETPATH . '/wpcookbook/' . $template_name ) ) {
        $located = STYLESHEETPATH . '/wpcookbook/' . $template_name;
    } elseif ( file_exists( TEMPLATEPATH . '/wpcookbook/' . $template_name ) ) {
        $located = TEMPLATEPATH . '/wpcookbook/' . $template_name;
    } elseif ( file_exists( WIDGETS_PATH . '/templates/' . $template_name ) ) {
        $located = WIDGETS_PATH . '/templates/' . $template_name;
    }

    return $located;
}
```

Dans la méthode `widget()`, on va charger le modèle des articles :

```
public function widget( $args, $instance ) {
    echo $args['before_widget'];
    ...
    while ( $query->have_posts() ){
        $query->the_post();
        include wpcookbook_locate_template( 'recent-posts-widget/post-card.php' );
    }
    ...
    wp_reset_postdata();
    echo $args['after_widget'];
}
```

## Un modèle pour nos articles

Et nous allons finalement créer un modèle par défaut `post-card.php` dans un dossier `templates/recent-post-widget` dans notre extension.

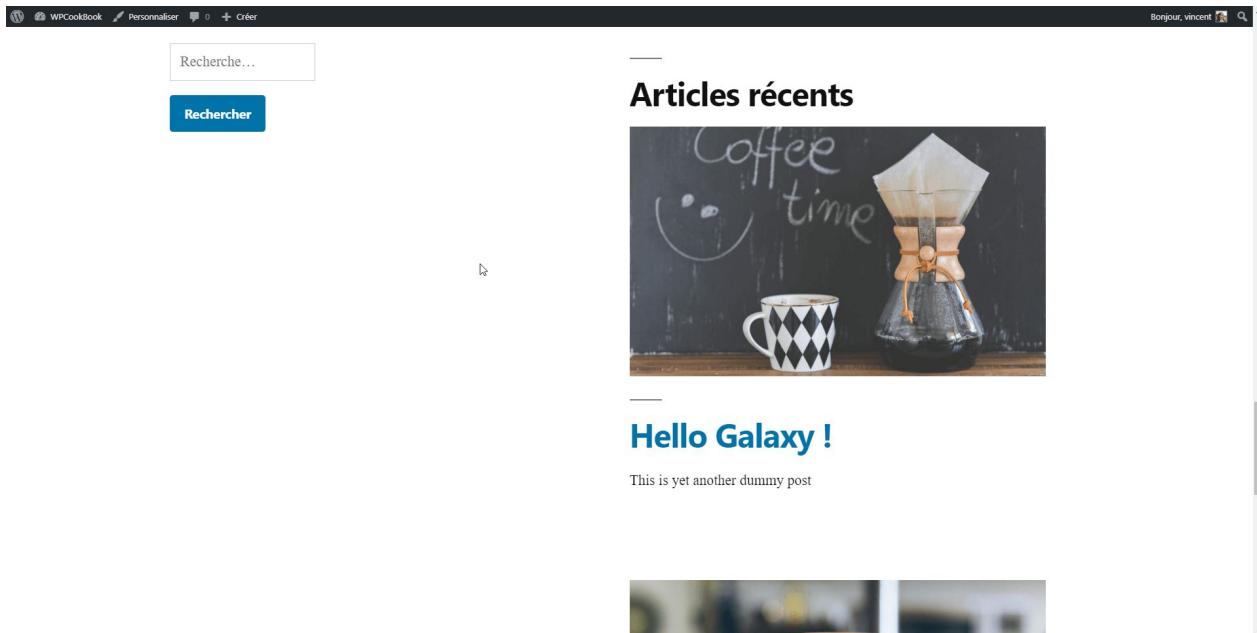
```

<?php
/**
 * The template for our posts in our recent posts widgets
 */
?>
<div <?php post_class( 'recent-posts-widget-card' );?>>
    <?php if ( has_post_thumbnail() ) : ?>
        <a href="<?php the_permalink(); ?>">
            <?php the_post_thumbnail( 'medium-large' ); ?>
        </a>
    <?php endif; ?>
    <?php the_title( sprintf( '<h2 class="recent-posts-widget-card-title"><a href="%s"
rel="bookmark">', esc_url( get_permalink() ) ), '</a></h2>' ); ?>
    <div class="recent-posts-widget-card-content"><?php the_excerpt(); ?></div>
</div>

```

On utilise simplement les templates tags basiques de WordPress pour afficher une image cliquable ( avec `has_post_thumbnail()`, `the_post_thumbnail()` et `the_permalink()`), un titre cliquable (avec `the_title()` et `get_permalink()`) et l'extrait de l'article avec `the_excerpt()`.

On prend le soin d'afficher toutes les classes utiles avec `post_class()` et on y ajoute la nôtre (`recent-posts-widget-card`) en la passant en paramètre.



Le template pour nos articles est fait.

Je vous laisse le soin d'ajouter un peu de CSS pour rendre ça plus joli, et le charger correctement.

## Ajouter des réglages

Pour ajouter des réglages, il suffit simplement d'ajouter les champs dans la méthode `form()` et de traiter les données soumises dans la méthode `update()`.

Ajoutons un champ pour le nombre d'articles à afficher :

```
public function form( $instance ) {
    $title      = ! empty( $instance['title'] ) ? $instance['title'] : '';
    $number_posts = ! empty( $instance['number_posts'] ) ? (int) $instance['title'] : 3;
    ?>

    <p>
        <label for=<?php echo esc_attr( $this->get_field_id( 'title' ) ); ?>>
            <?php esc_html_e( 'Title :', '24-widgets' ); ?>
        </label>
        <input
            type="text"
            class="widefat"
            id=<?php echo esc_attr( $this->get_field_id( 'title' ) ); ?>
            name=<?php echo esc_attr( $this->get_field_name( 'title' ) ); ?>
            value=<?php echo esc_attr( $title ); ?>
        >
    </p>
    <p>
        <label for=<?php echo esc_attr( $this->get_field_id( 'number_posts' ) ); ?>>
            <?php esc_html_e( 'Number of posts to display:', '24-widgets' ); ?>
        </label>
        <input
            type="number"
            class="widefat"
            id=<?php echo esc_attr( $this->get_field_id( 'number_posts' ) ); ?>
            name=<?php echo esc_attr( $this->get_field_name( 'number_posts' ) ); ?>
            value=<?php echo esc_attr( $number_posts ); ?>
        >
    </p>
<?php
}
```

On vérifie qu'un réglage est bien présent, sinon on déclare une valeur par défaut avec les opérateurs ternaires, puis on affiche un simple `<input type="number">` sous notre champ titre. Attention à bien utiliser `$this->get_field_id()` et `$this->get_field_name()` pour avoir les bonnes valeurs des attributs !

**Note :** Ici, j'utilise `esc_attr()` sur `$number_posts`, mais en réalité ce n'est pas nécessaire, car je me suis assuré juste avant que la variable contient bien un entier. Mais bon, c'est un bon réflexe.

Sauvegardons la données dans la méthode update() :

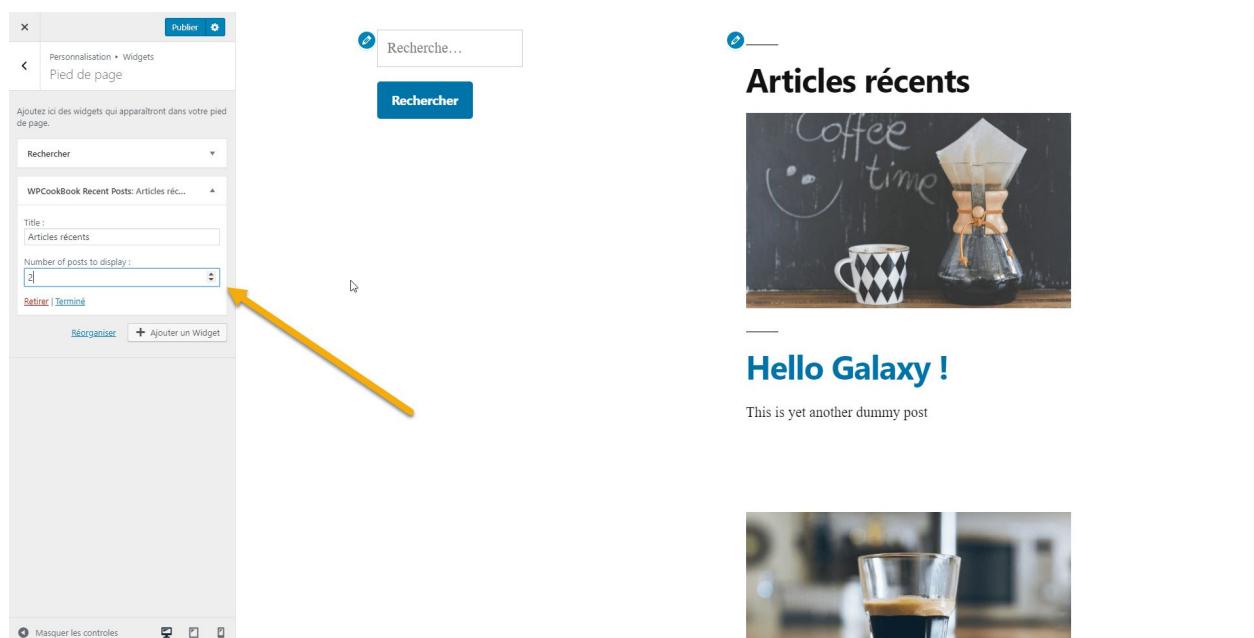
```
public function update( $new_instance, $old_instance ) {  
  
    if( $new_instance === $old_instance ){  
        return false;  
    }  
  
    $instance = array(  
        'title'      => ! empty( $new_instance['title'] ) ? sanitize_text_field($new_instance['title']) : '',  
        'number_posts' => ! empty( $new_instance['number_posts'] ) ? (int)$new_instance['number_posts'] : 3,  
    );  
  
    return $instance;  
}
```

On s'assure simplement que le nombre d'articles est bien un entier. Et voilà ! Notre nouveau réglage fonctionne !

On peut maintenant ajuster notre méthode widget() pour prendre en compte ce réglage dans les arguments de notre WP\_Query.

```
$query_args = array(  
    'posts_per_page' => ! empty( $instance['number_posts'] ) ? (int) $instance['number_posts']  
    : 3,  
);
```

Encore une fois, on fait attention à utiliser une valeur par défaut si l'utilisateur n'a rien indiqué.



Notre nouveau réglage fonctionne.

## À retenir

- Pour créer un nouveau type de widgets, on surcharge la classe `WP_Widget`.
- Dans notre classe, on surcharge le constructeur `__construct()` en appelant celui du parent, et en lui passant les informations nécessaires sur le widget. Au minimum un identifiant et un nom d'affichage.
- On doit surcharger la méthode `widget()` qui est responsable de l'affichage du contenu de notre widget sur le devant du site.
- On peut aussi surcharger les méthodes `form()` (contenu du formulaire en administration), `update()` (qui nettoie et sauvegarde nos données), si notre widget inclut des réglages.
- La méthode `widget( $args, $instance )` reçoit les arguments généraux `$args` de la zone de widgets, passés par le thème dans `register_sidebar()`, et les réglages de l'instance du widget `$instance`.
- Le méthode `form( $instance )` reçoit les réglages actuels de l'instance. Attention à bien utiliser `$this->get_field_id()` et `$this->get_field_name()` pour les attributs des champs ! Cela permet d'éviter d'avoir des attributs de même valeur si notre widget est utilisé plusieurs fois.
- Le méthode `update( $new_instance, $old_instance )` reçoit les réglages soumis par l'utilisateur `$new_instance` et les réglages actuels de l'instance `$old_instance` (avant sauvegarde). Elle doit retourner un tableau avec les nouveaux réglages, nettoyés.
- On enregistre un nouveau widget avec la fonction `register_widget()` hookée sur `widgets_init`. La fonction prend le nom de la classe du widget à enregistrer.

Et c'est tout ! C'est déjà pas mal, je vous l'avoue, mais si on prend un peu de recul, il suffit de surcharger une classe et quatre méthodes, en faisant attention d'utiliser les `$args` de la zone de widgets du thème et `get_field_name()` et `get_field_id()` pour les champs du formulaire.

Allez, on continue !

# Comment créer un type de contenu personnalisé

Les types de contenu personnalisé, ou Custom Post Types (CPT), sont un des outils les plus puissants de WordPress. Cette fonctionnalité a fait passer WordPress d'une plateforme de blogging en CMS (Content Management System ou système de gestion de contenu) robuste, et a ouvert un champ de possibilités énormes.

Quasiment tous les sites WordPress déclarent un CPT, que ce soit pour des produits, des pièces de portfolio, des témoignages clients, les membres de l'équipe, etc. Ils permettent de mieux classer votre contenu et de le hiérarchiser, et donne plus de contrôle sur son affichage.

## Déclarer un type de contenu personnalisé

WordPress utilise par défaut plusieurs types de contenu. Certains sont administrables par l'utilisateur, d'autres sont moins directement accessibles. Parmi les contenus administrables, on a les articles, les pages, les médias, les révisions, les menus. Parmi ceux moins évidents, on a le CSS personnalisé (eh ouais), les changements dans l'outil de personnalisation et les blocs de l'éditeur. C'est-à-dire qu'un type de contenu peut ne pas être public, et être utilisé uniquement en coulisses, ce qui est très pratique et offre des possibilités d'administration intéressantes.

Imaginons que nous voulions créer un type de contenu Snippets pour y ranger des bouts de code. Créons et activons une petite extension pour ce faire. Vous allez voir, il est très simple de déclarer de nouveaux types de contenu. Pour cela, on utilise la fonction `register_post_type()`.

```
register_post_type( string $post_type, array|string $args = array() )
```

La fonction se hooke sur `init`. Elle prend un identifiant `$post_type` en premier paramètre et un tableau d'arguments `$args` en second. Il y a de nombreux arguments, et pour l'instant on va les lister avec leur valeur par défaut (sauf le label et la description). Prêts ? Accrochez-vous !

Dans le fichier bootstrap de votre extension, ajoutez le snippet suivant :

```

add_action( 'init', 'wpcookbook_register_post_types' );
/**
 * Registers our 'Snippets' custom post type
 */
function wpcookbook_register_post_types(){
    register_post_type( 'wpcookbook-snippets', array(
        'label'                  => __x( 'Snippets', 'post type general name', '25-cpt' ),
        'labels'                 => array(),
        'description'            => __( 'Code snippets to help you develop.', '25-cpt' ),
        'hierarchical'           => false,
        'public'                 => false,
        'exclude_from_search'   => true, // Default to opposite value of 'public'
        'publicly_queryable'    => false, // Default to the value of 'public'
        'show_ui'                => false, // Default to the value of 'public'
        'show_in_menu'            => false, // Default to the value of 'show_ui'
        'show_in_nav_menus'      => false, // Default to the value of 'public'
        'show_in_admin_bar'      => false, // Default to the value of 'show_in_menu'
        'show_in_rest'            => false,
        'rest_base'              => 'wpcookbook-snippets',
        'rest_controller_class'  => 'WP_REST_Posts_Controller',
        'menu_position'          => null,
        'menu_icon'               => 'dashicons-admin-post',
        'capability_type'        => 'post',
        'capabilities'            => array(), // Based on capability_type if not provided
        'map_meta_cap'            => null,
        'supports'                => array( 'title', 'editor' ),
        'register_meta_box_cb'    => null,
        'taxonomies'              => array(),
        'has_archive'             => false,
        'rewrite'                 => true, // uses slug of post type by default
        'query_var'               => 'wpcookbook-snippets', // Defaults to slug
        'can_export'              => true,
        'delete_with_user'        => null,
    ) );
}

```

- **label** est un nom d'affichage pour notre type de contenu. En général, on fournit ici la forme pluriel.  
Attention à rendre la chaîne fournie traduisible.
- **labels** est un tableau de tous les labels utilisés dans l'administration et permet donc d'ajuster (presque) tous les messages dans l'administration. Nous verrons tous les labels possibles un peu plus loin.
- **description** est une description qui peut apparaître sur les archives, selon le thème.
- **hierarchical** détermine si le contenu est hiérachique et accepte des enfants, comme les pages par exemple.
- **public** est un booléen qui indique si oui ou non le type de contenu est public et apparaîtra dans l'administration et sur le devant du site.
- **exclude\_from\_search** est un booléen qui indique s'il faut exclure ce type de contenu des résultats de recherche.
- **publicly\_queryable** indique si ce type de contenu peut faire l'objet d'une requête sur le devant du site.
- **show\_ui** détermine s'il faut afficher ou non une interface en administration
- **show\_in\_menu** détermine s'il faut afficher le type de contenu dans le menu d'administration. Si on passe `true` alors un nouvel élément de menu principal est créé. On peut aussi passer un menu

existant pour le faire apparaître dans un sous-menu (par exemple `post.php`). C'est bien pratique si votre extension déclare plusieurs types de contenus personnalisés, mais que vous souhaitez les regrouper, par exemple.

- `show_in_nav_menus` détermine s'il faut créer une interface pour notre type de contenu dans l'administration des menus.
- `show_in_admin_bar` détermine s'il faut créer les raccourcis dans la barre d'administration
- `show_in_rest` détermine s'il faut rendre le type de contenu disponible dans l'API REST de WordPress. Le nouvel éditeur utilisant l'API REST de WordPress, il faut passer ce paramètre à `true` si vous voulez profiter du nouvel éditeur. Sinon ce sera l'éditeur classique.
- `rest_base` permet de modifier la route de base pour le type de contenu dans l'API REST.
- `rest_controller_class` permet de passer une classe de contrôleur personnalisé pour les routes de l'API REST. C'est plutôt avancé, et on ne traitera pas de cela dans ce guide.
- `menu_position` est un entier correspondant à la position du menu dans le menu principal de l'administration. Utile uniquement si `show_in_menu` vaut `true`. Par défaut, le type de contenu apparaît en bas de la première section, sous Commentaires.
- `menu_icon` permet d'utiliser une Dashicon de votre choix pour le menu. On peut lui passer un slug d'icône ou directement un SVG en utilisant une data URI. Vous pouvez trouver les Dashicons sur <https://developer.wordpress.org/resource/dashicons/>
- `capability_type` est une chaîne qui va servir de base pour déterminer les capacités nécessaires pour notre type de contenu. Par défaut, cela vaut `post`. Du coup, les capacités associées vont être celles des posts par défaut (`edit_posts`, `delete_posts`, etc.)
- `capabilities` est un tableau de capacités associées. Il n'y en a généralement pas besoin si on passe `capability_type`.
- `map_meta_cap` est un booléen permettant de déterminer si on utilise les meta-capacités. Vous trouverez plus de détails dans le chapitre Comprendre les rôles et capacités de WordPress.
- `supports` contient un tableau de fonctionnalités à activer pour notre type de contenu. Par défaut, dans l'administration, on n'aura accès qu'au titre et au contenu quand on veut créer un nouvel élément de contenu. On peut y ajouter `editor`, `comments`, `revisions`, `trackbacks`, `author`, `excerpt`, `page-attributes`, `thumbnail`, `custom-fields`, et `post-formats`.
- `register_meta_box_cb` est une fonction de rappel qui peut être appelée automatiquement pour déclarer les metaboxes associées à notre type de contenu. Une sorte de raccourci contenant les appels à `add_meta_box()`. On parlera plus en détails des metaboxes dans un prochain chapitre.
- `taxonomies` est un tableau d'identifiants de taxonomies à utiliser pour notre type de contenu. Pour l'instant, nous utiliserons uniquement une taxonomie déjà existante. Créer des taxonomies est le sujet du chapitre suivant. Notez qu'il faut que la taxonomie soit déclarée avant le type de contenu pour que ce paramètre fonctionne.
- `has_archive` est un booléen indiquant s'il faut créer des archives pour le type de contenu. On peut aussi passer une chaîne de caractères à utiliser comme `slug` pour les archives.
- `rewrite` permet de définir des règles de réécriture d'URL pour les permaliens. Si vous passez `false`, les jolis permaliens ne seront pas utilisés et les URLs contiendront à la place des paramètres comme <https://wpcookbook.local/?wpcookbook-snippets=mon-snippet>. Si vous passez `true`, l'URL ressemblera plutôt à <https://wpcookbook.local/wpcookbook-snippets/mon-snippet/>. Par défaut, le `slug` du type de contenu est utilisé.
- `query_var` permet de déterminer la clé de la variable d'URL utilisé pour notre type de contenu. Par défaut, cela vaut le `slug` du type de contenu.
- `can_export` autorise ou non l'export XML par défaut de WordPress pour ce type de contenu.

- `delete_with_user` détermine s'il faut supprimer les publications de ce type de contenu d'un utilisateur quand ce dernier est supprimé. Par défaut, les publications sont supprimées si le type de contenu a `author` dans ses supports.

La fonction retourne l'objet `WP_Post_Type` correspondant ou une `WP_Error` selon si l'opération a réussi ou non. Les paramètres offrent énormément de flexibilité, mais pour beaucoup de besoins, la plupart des réglages par défaut sont tout à fait acceptables. Donc pour notre exemple, on va procéder pas à pas et ajouter nos paramètres un par un.

## Déclarer un type de contenu basique

Pour le moment, dans notre fonction de rappel, utilisons ceci :

```
// in main plugin file
add_action( 'init', 'wpcookbook_register_post_types' );
/**
 * Registers our 'Snippets' custom post type
 */
function wpcookbook_register_post_types(){
    register_post_type( 'wpcookbook-snippets', array(
        'label'          => __( 'Snippets', '25-cpt' ),
        'description'   => __( 'Code snippets to help you develop.', '25-cpt' ),
        'public'         => true,
    ) );
}
```

On peut difficilement faire plus simple. On utilise juste un identifiant, un label unique, une description et on déclare le type de contenu public, ce qui signifie qu'il va apparaître dans l'administration, qu'on pourra ajouter du contenu de la même façon que pour un article et qu'il sera accessible sur le devant du site.

**Attention cependant à votre identifiant !** Il doit faire maximum 20 caractères, utilisant uniquement minuscules, tirets et underscores. En plus pour éviter toute collision de nom, il est recommandé de le préfixer, ce qui complique la tâche ! Ici, on a eu chaud car il fait 19 caractères !

Notre type de contenu est accessible. On peut déjà créer du contenu.

Voilà, notre type de contenu fonctionne. Presque. Parce que si vous essayez d'aller sur le devant du site pour voir votre nouveau snippet, il y a de grandes chances que vous rencontriez une erreur 404 !

Quand on déclare un type de contenu, les éléments de contenu doivent être disponibles sur le devant du site via une certaine URL. Selon les paramètres utilisés, on déclare volontairement ou non des règles de réécriture pour ces URLs, et WordPress stocke ces règles en base de données. Il faut donc simplement les mettre à jour.

On peut faire ceci de deux façons différentes : soit en visitant simplement la page de réglages des permaliens (Magic !) ou alors en appelant la fonction `flush_rewrite_rules()`. (En coulisses, la page de réglages des permaliens les rafraîchit juste avant de s'afficher)

Un bon moment pour appeler cette fonction est juste après avoir déclaré le type de contenu pour la première fois. Le souci c'est que notre fonction est hookée sur `init`, donc va s'exécuter à chaque chargement de page. Or rafraîchir les permaliens est assez coûteux donc on va éviter de le faire sur chaque page. Une solution est de déclarer notre type de contenu à l'activation de l'extension et d'utiliser `flush_rewrite_rules()` juste après.

Dans le fichier bootstrap de l'extension, ajoutez :

```
register_activation_hook( __FILE__, 'wpcookbook_activation' );
/**
 * Declare our Custom Post Type then flush rewrite rules on activation
 */
function wpcookbook_activation(){
    wpcookbook_register_post_types();
    flush_rewrite_rules();
}
```

Dès l'activation, on utilise notre fonction `wpcookbook_register_post_types()` suivi de `flush_rewrite_rules()` pour prévenir les erreurs 404. En désactivant puis réactivant l'extension, notre premier snippet est accessible sur le devant du site.



WPCookBook — Apprenez à développer pour WordPress  
[Page](#) [Article](#) [Catégorie](#) [Contact](#)

## Déclarer un type de contenu

vincent · novembre 20, 2019 · [Modifier](#)

Pour déclarer un type de contenu, on utilise la fonction `register_post_type()`, hookée sur `init`.

[Modifier](#)

Notre snippet est accessible sur le devant du site.

Vous remarquerez que l'URL utilise l'identifiant de notre type de contenu. Cela peut-être désirable ou non et c'est paramétrable dans la déclaration du type de contenu assez simplement.

## Améliorer l'interface d'administration

Pour le moment, j'aimerais revenir dans l'administration. Vous avez sûrement remarqué que les labels faisaient tous référence à des articles (Modifier l'article, par exemple), que l'on utilisait l'éditeur classique pour nos snippets, et que l'écran d'édition était un peu vide.

J'aimerais corriger les labels, utiliser le nouvel éditeur de WordPress et ajouter la possibilité d'associer une image mis en avant pour les snippets. J'aimerais aussi utiliser les étiquettes par défaut de WordPress pour classer ces snippets. Concrètement, il vaut mieux déclarer une taxonomie séparée pour notre type de contenu, mais c'est l'objet du chapitre suivant. Pas tout en même temps. Donc pour le moment, les étiquettes standards des articles feront l'affaire.

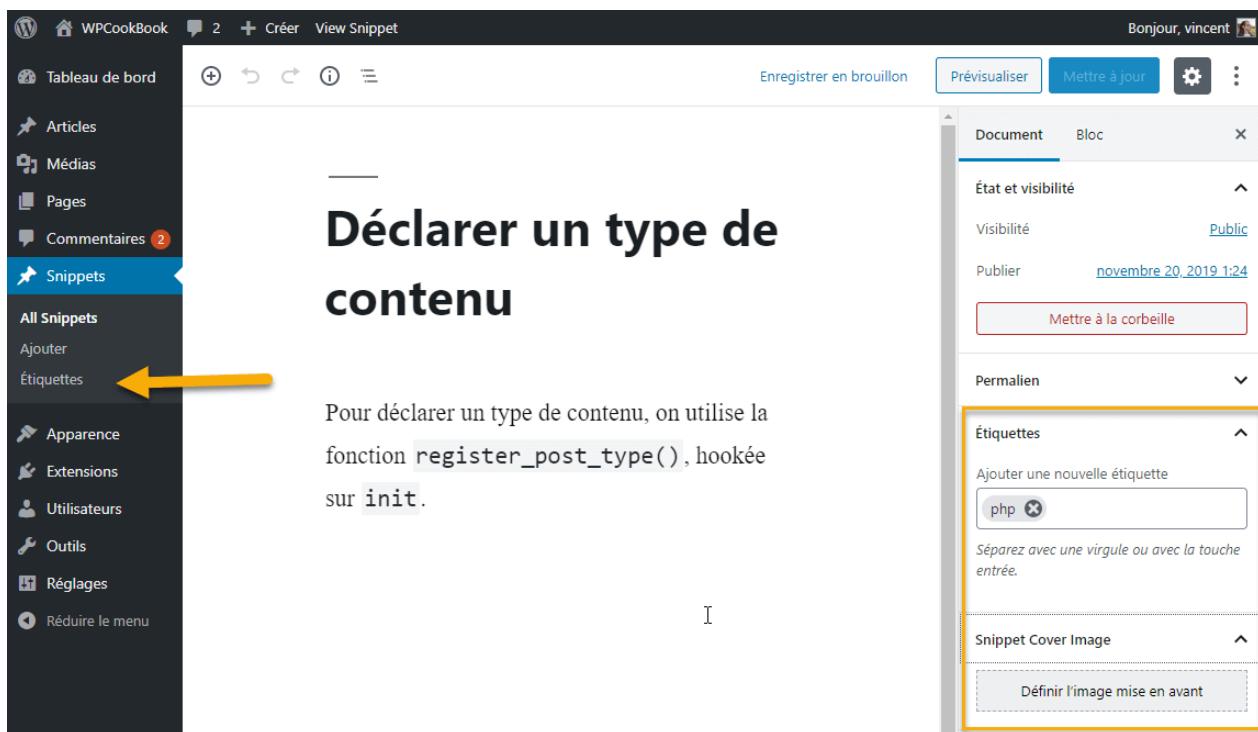
```
function wpcookbook_register_post_types(){
    $labels = array(
        'name'                  => _x( 'Snippets', 'Post type general name', '25-cpt' ),
        'singular_name'         => _x( 'Snippet', 'Post type singular name', '25-cpt' ),
        'menu_name'              => _x( 'Snippets', 'Admin Menu text', '25-cpt' ),
        'name_admin_bar'        => _x( 'Snippet', 'Add New on Toolbar', '25-cpt' ),
        // 'add_new'               => __( 'Add New' ), // On va utiliser la traduction de
    WordPress
        'add_new_item'           => __( 'Add New Snippet', '25-cpt' ),
        'new_item'                => __( 'New Snippet', '25-cpt' ),
        'edit_item'                => __( 'Edit Snippet', '25-cpt' ),
        'view_item'                => __( 'View Snippet', '25-cpt' ),
        'all_items'                => __( 'All Snippets', '25-cpt' ),
        'search_items'           => __( 'Search Snippets', '25-cpt' ),
        'parent_item_colon'      => __( 'Parent Snippets:', '25-cpt' ),
        'not_found'                => __( 'No snippets found.', '25-cpt' ),
        'not_found_in_trash'     => __( 'No snippets found in Trash.', '25-cpt' ),
        'featured_image'          => _x( 'Snippet Cover Image', 'Overrides the "Featured Image" phrase for this post type. Added in 4.3', '25-cpt' ),
        // 'set_featured_image'    => _x( 'Set cover image', 'Overrides the "Set featured image" phrase for this post type. Added in 4.3', '25-cpt' ),
        // 'remove_featured_image' => _x( 'Remove cover image', 'Overrides the "Remove featured image" phrase for this post type. Added in 4.3', '25-cpt' ),
        // 'use_featured_image'     => _x( 'Use as cover image', 'Overrides the "Use as featured image" phrase for this post type. Added in 4.3', '25-cpt' ),
        'archives'                 => _x( 'Snippet archives', 'The post type archive label used in nav menus. Default "Post Archives". Added in 4.4', '25-cpt' ),
        'insert_into_item'         => _x( 'Insert into snippet', 'Overrides the "Insert into post"/"Insert into page" phrase (used when inserting media into a post). Added in 4.4', '25-cpt' ),
        'uploaded_to_this_item'    => _x( 'Uploaded to this snippet', 'Overrides the "Uploaded to this post"/"Uploaded to this page" phrase (used when viewing media attached to a post). Added in 4.4', '25-cpt' ),
        'filter_items_list'        => _x( 'Filter snippets list', 'Screen reader text for the filter links heading on the post type listing screen. Default "Filter posts list"/"Filter pages list". Added in 4.4', '25-cpt' ),
        'items_list_navigation'    => _x( 'Snippets list navigation', 'Screen reader text for the pagination heading on the post type listing screen. Default "Posts list navigation"/"Pages list navigation". Added in 4.4', '25-cpt' ),
        'items_list'                 => _x( 'Snippets list', 'Screen reader text for the items list heading on the post type listing screen. Default "Posts list"/"Pages list". Added in 4.4', '25-cpt' ),
    );
}
```

```

register_post_type( 'wpcookbook-snippets', array(
    'labels'      => $labels,
    'description' => __( 'Code snippets to help you develop.', '25-cpt' ),
    'public'       => true,
    'show_in_rest' => true,
    'supports'     => array( 'title', 'editor', 'thumbnail' ),
    'taxonomies'   => array( 'post_tag' ),
) );
}

```

Ici, on a ajouté `show_in_rest` pour déclarer notre type de contenu dans l’API REST et pouvoir activer le nouvel éditeur de WordPress. Aussi, on a déclaré qu’il utilisait la taxonomie `post_tag` et on a ajouté le support pour les images mises en avant en ajoutant `thumbnail` dans le tableau par défaut des supports. Enfin, on a ajouté un gros tableau contenant nos labels personnalisés pour l’administration, en s’assurant que tout est traduisible.



Maintenant, on peut utiliser le nouvel éditeur, ajouter des étiquettes et une image mise en avant.

C'est beaucoup mieux. Si ça ne marche pas, c'est qu'il faut rafraîchir encore les permaliens, simplement. Visitez la page de réglages des permaliens et le tour est joué .

On n'a aucune raison de modifier les paramètres de route pour la REST API, donc on va complètement ignorer les paramètres `rest_base` et `rest_controller_class`. Aussi, on n'a aucun besoin d'exclure nos snippets des résultats de recherche, ni de ne pas les afficher dans les différentes sections d'administration (barre d'admin, menus, etc.). Donc tout hériter de la valeur de `public` est tout à fait acceptable. Sachez juste que vous avez le contrôle sur l'accès à votre type de contenu dans l'administration, que ce soit via les paramètres d'affichage ou via les capacités.

## Améliorer l'accès sur le devant du site

Mais j'aimerais ajouter un lien dans mon menu vers tous les snippets, c'est-à-dire vers l'archive du type de contenu. Or, si je vais dans l'administration des menus, mes snippets apparaissent bien mais je ne trouve que mon unique snippet ! Pas de lien vers les archives !

The screenshot shows the 'Ajouter des éléments de menu' (Add menu items) dialog box. At the top, 'Snippets' is selected from a dropdown menu. Below this, there's a search bar with 'Les plus récentes' and 'Afficher tout' buttons. There are two checkboxes: one for 'Déclarer un type de contenu' and another for 'Tout sélectionner'. At the bottom is a large blue 'Ajouter au menu' button. Other categories like 'Pages', 'Articles', 'Liens personnalisés', and 'Catégories' are listed with dropdown arrows.

Le lien vers les archives n'est pas disponible.

Aussi, si j'essaie de tricher en tapant l'URL <https://wpcookbook.local/wpcookbook-snippets/> censée mener aux archives de notre type de contenu par défaut, j'ai une belle 404 ! Pour corriger cela, on va ajouter quelques paramètres.

```
// in wpcookbook_register_post_types()
register_post_type( 'wpcookbook-snippets', array(
    'labels'          => $labels,
    'description'    => __( 'Code snippets to help you develop.', '25-cpt' ),
    'public'          => true,
    'menu_icon'       => 'dashicons-editor-code',
    'show_in_rest'   => true,
    'supports'        => array( 'title', 'editor', 'thumbnail' ),
    'taxonomies'      => array( 'post_tag' ),
    'has_archive'     => true,
) );
```

J'en ai profité pour ajouter une nouvelle icône, en passant un slug d'une des dashicons utilisées dans l'administration de WordPress. Mais ce qui est important ici, c'est qu'on a ajouté le paramètre `has_archive`. Maintenant, notre lien vers les archives des snippets apparaît dans l'administration des

menus, et la page à l'URL <https://wpcookbook.local/wpcookbook-snippets/> nous liste bien tous nos snippets.

The screenshot shows a dark-themed WordPress dashboard header. From left to right, it includes: a user icon, a gear icon, the text "WPCookBook", a pen icon, "Personnaliser", a speech bubble icon with "3", and a plus sign icon with "Créer". Below the header, the main navigation bar has five items: "Page", "Article", "Catégorie", "Contact", and "Snippets", with "Snippets" being the active one and highlighted with a yellow border. The main content area has a yellow-bordered box containing the text "Archives du type de publication : Snippets.". A cursor arrow is visible on the right side of the page.

## Déclarer un type de contenu

Pour déclarer un type de contenu, on utilise la fonction `register_post_type()`, hookée sur `init`.

Les archives sont maintenant accessibles.

## Peaufinons nos URL

Actuellement, nos URL contiennent le préfixe de notre identifiant (`wpcookbook-`). C'est assez moyen. Mais la fonction `register_post_type()` fournit des paramètres pour réécrire ces URL.

```
// in wpcookbook_register_post_types()
register_post_type( 'wpcookbook-snippets', array(
    ...
    'has_archive' => true,
    'rewrite'      => array(
        'slug' => 'snippet'
    ),
) );
```

Le paramètre `rewrite` prend un tableau associatif de paramètres. En spécifiant le `slug` à utiliser pour notre type de contenu, les URL sont bien plus propres et explicites. Nos archives vivent à <https://wpcookbook.local/snippet/>, et notre snippet exemple vit à

<https://wpcookbook.local/snippet/declarer-un-type-de-contenu/>.

Cela dit, utiliser un pluriel pour les archives (/snippets), ce serait pas mal non plus. Il suffit d'ajuster le paramètre `has_archive`.

```
// in wpcookbook_register_post_types()
register_post_type( 'wpcookbook-snippets', array(
    ...
    'has_archive' => 'snippets',
    'rewrite'      => array(
        'slug' => 'snippet'
    ),
) );
```

Là c'est mieux ! Nos archives vivent à <https://wpcookbook.local/snippets/>.

Mais on a un souci. Notre structure de permaliens utilise simplement le nom de la publication, mais que se passe-t-il si on ajoute `blog/` devant le nom des publications ?

The screenshot shows the WordPress admin dashboard with the 'Réglages' (Settings) menu selected. On the left, a sidebar lists various settings categories. The 'Permalien' (Permalink) option is highlighted. The main content area is titled 'Réglages les plus courants' (Common settings). It displays five preset structures and a 'Structure personnalisée' (Custom structure) field. The 'Structure personnalisée' field contains the value `https://wpcookbook.local /blog/%postname%`, which is highlighted with a yellow border. Below this field is a list of available shortcodes: %year%, %monthnum%, %day%, %hour%, %minute%, %second%, %post\_id%, %postname%, and %category%.

La structure par défaut est personnalisable.

Cela marche bien pour nos articles. Par exemple, l'article Hello World se trouve à <https://wpcookbook.local/blog/hello-world>, mais par contre, nos archives de snippets se trouvent à <https://wpcookbook.local/blog/snippets> et notre snippet simple à <https://wpcookbook.local/blog/snippet/declarer-un-type-de-contenu>.

Il faudrait donc enlever le préfixe, si possible.

```
// in wpcookbook_register_post_types()
register_post_type( 'wpcookbook-snippets', array(
    ...
    'has_archive' => 'snippets',
    'rewrite'      => array(
        'slug'       => 'snippet',
        'with_front' => false,
    ),
) );
```

Le paramètre `with_front` permet d'ajouter ou d'ignorer le préfixe personnalisé des articles, simplement. En le mettant sur `false`, retour à la normale. On retrouve les URL souhaitées pour nos snippets, indépendamment du réglage du préfixe pour les articles.

## Types de contenu personnalisé et modèles de page

On a vu que tout s'affiche correctement sur le devant du site. Si vous avez lu attentivement le chapitre Comprendre son thème, vous devriez savoir quel modèle a été utilisé et pourquoi. Si non, je vous invite à le lire, et à revenir ici. Je vous attends.

De retour ? Nos archives de snippets utilisent le modèle de page des archives par défaut, et ce modèle dans notre thème `twentynineteen` est `archive.php`. Les snippets seuls utilisent le même modèle que les articles, soit `single.php`.

A partir de là, on peut dupliquer ces deux fichiers dans notre thème enfant et les modifier pour prendre en compte notre type de contenu `wpcookbook-snippets` ou alors on peut les renommer en `archive-wpcookbook-snippets.php` et `single-wpcookbook-snippets.php`. Ces deux fichiers sont plus spécifiques et seront donc utilisés en priorité pour notre type de contenu. Je vous laisse personnaliser les modèles comme vous le souhaitez.

Mais nos snippets peuvent apparaître aussi sur les pages d'archives des étiquettes, car nous lui avons déclaré cette taxonomie. Pour tester, j'ai ajouté une étiquette PHP à l'article `Hello World` et à notre snippet exemple. En théorie, ces deux éléments de contenu devraient apparaître sur la page des archives de l'étiquette.

Archives de l'étiquette :  
**php.**

---

## Hello world!

Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

👤 vincent ⏰ septembre 12, 2019 📁 Uncategorized, WordPress 💬 php  
💬 Un commentaire ✎ Modifier

Notre snippet est absent. Aïe.

Malheureusement notre snippet n'apparaît pas, simplement parce que la page d'archives pour une étiquette cherche dans le type de contenu `post` par défaut. Il faut donc lui dire de chercher aussi dans nos snippets.

```

// in main plugin file
add_action( 'pre_get_posts', 'wpcookbook_tag_archive_query' );
/**
 * Adjust main query to include our wpcookbook-snippets post type
 *
 * @param WP_Query $query The main query object
 */
function wpcookbook_tag_archive_query( $query ) {
    if ( $query->is_tag() && $query->is_main_query() ) {
        $query->set( 'post_type', array( 'post', 'wpcookbook-snippets' ) );
    }
}

```

Pour inclure le type de contenu wpcookbook-snippets, il faut utiliser ce qu'on a vu dans le chapitre Comprendre la boucle de WordPress, et se hooker sur `pre_get_posts` pour modifier la requête. On vérifie simplement qu'on est bien sur une page d'archives d'étiquette avec `is_tag()`, qu'on est bien sur la boucle principale avec `is_main_query()` et si c'est le cas, on ajoute wpcookbook-snippets aux types de contenu à inclure.

## Ce qu'il faut retenir

- On déclare un type de contenu personnalisé à l'aide de la fonction `register_post_type()` hookée sur `init`. On lui passe un slug et un tableau d'arguments.
- N'oubliez pas de déclarer votre CPT et d'utiliser `flush_rewrite_rule()` à l'activation pour éviter les erreurs 404.
- Vous pouvez personnaliser complètement l'administration de votre type de contenu, en rendant disponible ou non les raccourcis, les éléments de menu, en activant ou désactivant certaines fonctionnalités de l'éditeur, etc.
- Vous pouvez aussi personnaliser vos URL à l'aide des paramètres `rewrite`, `has_archive`, ou `query_vars`.
- Vous pouvez contrôler l'accès au contenu avec des capacités.
- Vous pouvez aussi associer n'importe quelle taxonomie à votre type de contenu, y compris des taxonomies personnalisées.

On n'a pas joué avec tous les paramètres, pour la bonne raison que les valeurs par défaut sont très bien et ce que vous avez vu jusqu'ici va couvrir 95% de vos besoins. Par contre, on va parler plus en détails des taxonomies dans le chapitre suivant. Ça, vous en aurez sûrement besoin !

# Comment déclarer une taxonomie personnalisée

Tout comme les types de contenu personnalisés, les taxonomies personnalisées sont un outil très puissant. On n'est plus limité aux catégories et étiquettes pour classer notre contenu et on peut créer autant de taxonomies que l'on souhaite pour tout contenu, qu'il soit d'un type natif comme les articles, ou personnalisé comme les snippets déclarés au chapitre précédent.

Et WordPress met à notre disposition tous les outils nécessaires pour gérer nos taxonomies dans l'administration. C'est très simple.

## Déclarer une taxonomie

Comme pour les types de contenu personnalisé, il suffit d'appeler une seule fonction : `register_taxonomy()`. Cette fonction se hooke sur `init`, exactement comme `register_post_type()`. Dans une nouvelle extension, ajoutez le code suivant :

```
// in main plugin file
add_action( 'init', 'wpcookbook_register_taxonomies' );
/**
 * Registers a 'Programming Languages' taxonomy for posts and snippets
 */
function wpcookbook_register_taxonomies(){
    register_taxonomy( 'wpcookbook-technologies', 'post', array() );
}
```

```
register_taxonomy( string $taxonomy, array|string $object_type, array|string
$args = array() )
```

La fonction prend trois paramètres : un identifiant/slug, une chaîne ou un tableau spécifiant le(s) type(s) de contenu au(x)quel(s) notre taxonomie est associée et un tableau d'arguments pour la paramétriser.

Pour l'exemple, nous allons créer une taxonomie `wpcookbook-technologies` pour classer nos articles. Vous pouvez associer la taxonomie `wpcookbook-technologies` au type de contenu `snippets` que nous avons créé au chapitre précédent, si vous le souhaitez.

Voici les arguments avec leur valeur par défaut :

```
add_action( 'init', 'wpcookbook_register_taxonomies' );
/**
 * Registers a 'Programming Languages' taxonomy for posts and snippets
 */
function wpcookbook_register_taxonomies(){
    register_taxonomy( 'wpcookbook-technologies', 'post', array(
        'label'              => __( 'Technologies', '26-taxonomies' ),
        // 'labels'           => array(),
        'description'        => __( 'Technologies discussed in the post', '26-taxonomies' ),
        'hierarchical'       => false,
        'public'             => true,
        'publicly_queryable' => true, // Default value inherited from 'public'
        'show_ui'            => true, // Default value inherited from 'public'
        'show_in_menu'        => true, // Default value inherited from 'show_ui'
        'show_in_nav_menus'   => true, // Default value inherited from 'public'
        'show_in_rest'        => true, // Default false
        'rest_base'          => 'wpcookbook-technologies', // Defaults to taxonomy slug
        'rest_controller_class' => 'WP_REST_Terms_Controller',
        'show_tagcloud'       => true,
        'show_in_quick_edit'  => true, // Default value inherited from 'show_ui'
        'show_admin_column'   => false,
        'meta_box_cb'         => null, // Uses standard categories or tags metabox depending
        on value of 'hierarchical'
        'meta_box_sanitize_cb' => null,
        'capabilities'        => array(), // Default to standard capabilities associated with
        categories or tags, depending on the value of 'hierarchical'
        'rewrite'             => true,
        'query_var'           => 'wpcookbook-technologies',
        'update_count_callback' => '',
    ) );
}
```

Si vous avez lu le chapitre précédent, alors vous reconnaîtrez beaucoup de paramètres. Sinon, voici une description plus détaillée :

- `label` est un nom d'affichage pour la taxonomie. En général, on utilise une forme pluriel.
- `labels` est un tableau des labels utilisés dans l'administration. Par défaut, les labels utilisés sont ceux associés aux catégories ou étiquettes, selon si la taxonomie est hiérarchique ou non.
- `description` est une description qui peut apparaître sur les archives selon le thème.
- `hierarchical` détermine si la taxonomie est hiérarchique (c'est-à-dire si les termes acceptent des enfants). Les catégories des articles sont un exemple de taxonomie hiérarchique et les étiquettes un exemple de taxonomie non-hiéarchique.
- `public` est un booléen qui indique si oui ou non la taxonomie est publique et apparaitra dans l'administration et sur le devant du site.
- `publicly_queryable` indique si la taxonomie peut faire l'objet d'une requête sur le devant du site.
- `show_ui` détermine s'il faut afficher ou non une interface en administration.
- `show_in_menu` détermine s'il faut afficher la taxonomie dans le sous-menu du type de contenu associé. Le paramètre hérite sa valeur de `public` par défaut.
- `show_in_nav_menus` détermine s'il faut créer une interface pour la taxonomie dans l'administration des menus.

- `show_in_rest` détermine s'il faut rendre la taxonomie disponible dans l'API REST de WordPress. `true` rend disponible la taxonomie dans le nouvel éditeur de WordPress.
- `rest_base` permet de modifier la route de base pour la taxonomie dans l'API REST. Par défaut, la route utilise le slug de la taxonomie.
- `rest_controller_class` permet de passer une classe de contrôleur personnalisée pour les routes de l'API REST.
- `show_tagcloud` indique s'il faut montrer les termes de la taxonomie dans le widgetnuage d'étiquettes de WordPress. lol
- `show_in_quick_edit` indique s'il faut montrer le formulaire pour éditer nos termes dans l'interface de modifications rapides.
- `show_admin_column` indique s'il faut ajouter une colonne pour notre taxonomie dans la liste des contenus associés.
- `meta_box_cb` est la fonction de rappel à utiliser pour l'administration de nos termes de taxonomie. Par défaut, la même metabox que pour les catégories ou pour les étiquettes est utilisée selon si la catégorie est hiérarchique ou non.
- `meta_box_sanitize_cb` est une fonction de rappel à utiliser pour nettoyer les données de la taxonomie avant sauvegarde. Si aucune n'est fournie, la fonction par défaut dépend de `meta_box_cb`.
- `capabilities` est un tableau de capacités associées nécessaires pour gérer la taxonomie. Par défaut, il faut les mêmes capacités que pour gérer les catégories ou les étiquettes.
- `rewrite` permet de définir des règles de réécriture d'URL pour les permaliens. Par défaut, le paramètre vaut `true` et va réécrire vos URL en utilisant le slug de la taxonomie. Si vous passez `false`, les URL contiendront à la place des paramètres comme `https://wpcookbook.local/?wpcookbook-technologies=php`.
- `query_var` permet de déterminer la clé de la variable d'URL utilisé pour la taxonomie. Par défaut, cela vaut le slug du type de contenu.
- `update_count_callback` est une fonction de rappel à utiliser qui va compter le nombre de termes.

Comme pour les types de contenu personnalisé, il y a beaucoup de paramètres. Ce qui veut dire qu'on a beaucoup de contrôle, mais la plupart des paramètres ont une valeur par défaut tout à fait acceptable. On va donc travailler pas à pas et faire évoluer ses paramètres au fil de nos besoins.

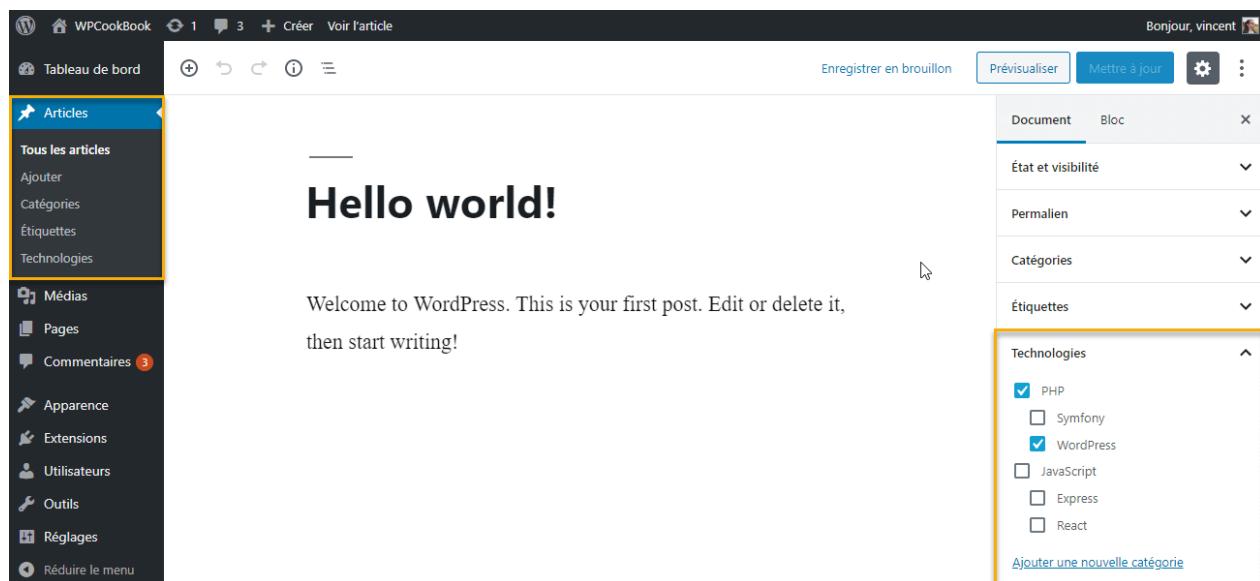
```
// in main plugin file
add_action( 'init', 'wpcookbook_register_taxonomies' );
/**
 * Registers a 'Technology' taxonomy for posts and snippets
 */
function wpcookbook_register_taxonomies(){
    register_taxonomy( 'wpcookbook-technologies', 'post', array(
        'label'          => __( 'Technologies', '26-taxonomies' ),
        'description'   => __( 'Technologies discussed in the post', '26-taxonomies' ),
        'hierarchical'  => true,
        'public'         => true,
        'show_in_rest'   => true, // Default false
    ) );
}
```

Pour commencer, on va déclarer notre taxonomie `wpcookbook-technologies` comme disponible

uniquement pour les articles simples (`post`). Le slug de notre taxonomie doit faire moins de 32 caractères et n'utiliser que des minuscules, tirets et underscores. Par précaution, je le préfixe d'un `wpcookbook-`.

Parmi les arguments, on lui passe simplement un label, une description et on la déclare hiérarchique. Pour ce paramètre, la question est simple : est-ce qu'on veut que notre taxonomie se comporte plutôt comme une catégorie d'articles (les sous-catégories sont possibles), ou plutôt comme une étiquette ? Je souhaite pouvoir mettre des sous-technologies pour représenter des spécialités, outils ou frameworks comme Symfony, WordPress ou React, donc on la déclare hiérarchique.

Aussi, on la déclare publique pour qu'une interface en administration soit disponible et qu'on puisse aussi en afficher des archives sur le devant du site. Enfin, `show_in_rest` va permettre de faire apparaître la taxonomie sur la page d'édition d'un article dans le nouvel éditeur.



Tout fonctionne correctement.

Voilà, vous savez déclarer une taxonomie ! C'était pas trop compliqué. Maintenant, si vous avez déclaré un type de contenu personnalisé, il suffit de changer le deuxième paramètre de la fonction `register_taxonomy()` pour le remplacer par le slug de votre type de contenu.

## Utiliser notre taxonomie sur le devant du site

On peut ajouter un lien vers les archives des termes de notre taxonomie de la même façon qu'on peut ajouter un lien vers une catégorie dans le menu. Si vous ajoutez un élément de menu menant vers les articles ayant pour language PHP, et que vous visitez ce lien sur le devant du site, vous risquez d'avoir un message d'erreur ou la liste de vos articles, sans prise en compte de votre requête pour le langage PHP.

La raison est exactement la même que pour les types de contenu. En déclarant une taxonomie, il faut rafraîchir les permaliens pour prendre en compte nos nouvelles règles de réécriture d'URL. On peut enregistrer notre taxonomie dès l'activation et utiliser la fonction `flush_rewrite_rules()` pour ce faire. Je vous renvoie au chapitre précédent pour plus de détails, le processus étant exactement le même.

Visitez la page des réglages des permaliens puis revenez sur le devant du site, et vous verrez que les archives de notre terme de taxonomie fonctionnent correctement.

Par contre, ce serait bien de pouvoir afficher un lien vers les archives du langage dans le pied de l'article, exactement comme on affiche un lien vers les catégories de l'article.

---

# Hello Galaxy !

This is yet another dummy post

 vincent  novembre 1, 2019  Test, WordPress  Laisser un commentaire  Modifier

Un petit lien ici serait sympa.

Ce petit pied d'article est affiché par la fonction `twentynineteen_entry_footer()` du thème parent `twentynineteen`, mais cette fonction est précédé d'un `if` vérifiant si la fonction n'existe pas déjà avant de la déclarer. Ce qui veut dire qu'on peut la déclarer dans notre thème enfant sans provoquer d'erreur.

Le fichier `functions.php` de notre thème enfant étant chargé avant le parent, on va surcharger cette fonction dans notre enfant en la copiant-collant et en ajoutant simplement un lien vers nos langages.

```

if ( ! function_exists( 'twentynineteen_entry_footer' ) ) :
    /**
     * Prints HTML with meta information for the categories, tags and comments.
     */
    function twenty_nineteen_entry_footer() {

        // Hide author, post date, category and tag text for pages.
        if ( 'post' === get_post_type() ) {

            ...

            /* translators: used between list items, there is a space after the comma. */
            $tags_list = get_the_tag_list( '', __( ', ', 'twentynineteen' ) );
            if ( $tags_list ) {
                ...

                the_terms(
                    get_the_ID(),
                    'wpcookbook-technologies',
                    sprintf( '<span class="tech-list">%1$s<span class="screen-reader-text">%2$s</span>',
                        twenty_nineteen_get_icon_svg( 'archive', 16 ),
                        __( 'Technologies: ', 'twentynineteen-child' )
                    ),
                    '',
                    '</span>'
                );
            }
            ...
        }
    }
endif;

```

On affiche les termes de notre taxonomie en utilisant la fonction `the_terms()`.

```
the_terms( int $id, string $taxonomy, string $before = '', string $sep = ', ',
string $after = '' )
```

Elle accepte 5 paramètres : l'identifiant de la publication, le slug de la taxonomie, une chaîne affichée avant la liste (en général un peu de code HTML), une chaîne servant de séparateur pour les éléments de la liste, et une chaîne affichée après la liste (on ferme nos balises HTML).

On lui passe donc `get_the_ID()` qui va récupérer l'identifiant de la publication courante et l'identifiant de la taxonomie `wpcookbook-technologies`. C'est notre `$before` qui est un peu moins évident, car on affiche ici un conteneur `<span class="tech-list">`, une icône et un texte pour les lecteurs d'écran, tout cela en utilisant la fonction `sprintf()`. Notre séparateur est une simple virgule et enfin on ferme notre conteneur `</span>`.

Ici, j'essaie simplement de reproduire le même balisage et format que pour les liens vers les catégories et étiquettes voisines. Malheureusement, le thème ne fournit pas d'icône plus adaptée pouvant représenter un langage de programmation. Je vous laisse chercher un meilleur svg.

# Hello world!



Welcome to WordPress. This is your first post. Edit or delete it, then start writing!

vincent · septembre 12, 2019 · Uncategorized, WordPress · Un commentaire  
PHP, WordPress · Modifier

Les liens fonctionnent correctement.

Aussi, on peut personnaliser ce modèle de page. C'est une archive de termes de taxonomie, mais le thème parent ne propose pas de modèle spécifique pour les taxonomies ou termes. C'est donc `archive.php` qui est utilisé. Dans votre thème enfant, vous pouvez donc créer un fichier plus spécifique comme `taxonomy-wpcookbook-technologies.php` pour prendre le contrôle sur ces archives.

## Améliorons nos URL

Comme pour la déclaration d'un type de contenu, on peut personnaliser la structure des URL des pages d'archives des termes. Pour le moment, nos URL sont du type `https://wpcookbook.local/wpcookbook-technologies/wordpress/`, c'est-à-dire qu'elles utilisent le slug de la taxonomie par défaut.

```
// main plugin file
function wpcookbook_register_taxonomies(){
    register_taxonomy( 'wpcookbook-technologies', 'post', array(
        'label'          => __( 'Technologies', '26-taxonomies' ),
        'description'   => __( 'Technologies discussed in the post', '26-taxonomies' ),
        'hierarchical'  => true,
        'public'         => true,
        'show_in_rest'  => true, // Default false
        'rewrite'        => array(
            'slug' => 'technologies',
        ),
    )));
}
```

En passant un `slug` personnalisé dans le paramètre `rewrite`, nos permaliens sont plus jolis : `https://wpcookbook.local/technologies/php/`.

Aussi, par défaut, le préfixe ajouté pour les articles dans la page des réglages des permaliens est aussi pris en compte pour nos archives de termes.

## Réglages les plus courants

The screenshot shows the 'Permalinks' settings page in WordPress. It lists several options for generating URLs:

- Simple: <https://wpcookbook.local/?p=123>
- Date et titre: <https://wpcookbook.local/2019/11/21/exemple-article/>
- Mois et titre: <https://wpcookbook.local/2019/11/exemple-article/>
- Numérique: <https://wpcookbook.local/archives/123>
- Titre de la publication: <https://wpcookbook.local/exemple-article/>
- Structure personnalisée: <https://wpcookbook.local /blog/%postname%/>

A yellow box highlights the 'Structure personnalisée' section. Below it, a list of available labels is shown:

- %year%
- %monthnum%
- %day%
- %hour%
- %minute%
- %second%
- %post\_id%
- %postname%
- %category%
- %author%

Un préfixe personnalisé va affecter nos archives.

```
// main plugin file
function wpcookbook_register_taxonomies(){
    register_taxonomy( 'wpcookbook-technologies', 'post', array(
        'label'      => __( 'Technologies', '26-taxonomies' ),
        'description' => __( 'Technologies discussed in the post', '26-taxonomies' ),
        'hierarchical' => true,
        'public'      => true,
        'show_in_rest' => true, // Default false
        'rewrite'     => array(
            'slug'      => 'technologies',
            'with_front' => false,
        ),
    ) );
}
```

A l'aide de `with_front` sur `false`, nos URL restent du type  
<https://wpcookbook.local/technologies/php/> et non  
<https://wpcookbook.local/blog/technologies/php/>.

Aussi, si vous le souhaitez vous pouvez refléter la hiérarchie de votre taxonomie dans les URL.

```
// main plugin file
function wpcookbook_register_taxonomies(){
    register_taxonomy( 'wpcookbook-technologies', 'post', array(
        ...
        'rewrite'     => array(
            'slug'      => 'technologies',
            'with_front' => false,
            'hierarchical' => true,
        ),
    ) );
}
```

En passant `hierarchical` à `true` dans le tableau des `rewrite`, vos URL affichent clairement les sous-éléments. Par exemple, l'URL du terme enfant WordPress (le parent étant PHP) devient `https://wpcookbook.local/technologies/php/wordpress/`. Ce n'est bien évidemment pas obligatoire, mais c'est une possibilité.

## Améliorons l'interface d'administration

Pour améliorer un peu l'administration, nous pouvons d'abord passer un tableau de `labels`. Vous avez peut-être remarqué que les labels utilisés étaient ceux des catégories. C'est simplement parce que notre taxonomie est hiérarchique. Donc elle se comporte comme une catégorie et affiche les chaînes par défaut des catégories.

L'ensemble des labels possibles se trouve ici :

[https://developer.wordpress.org/reference/functions/get\\_taxonomy\\_labels/](https://developer.wordpress.org/reference/functions/get_taxonomy_labels/).

```
// main plugin file
function wpcookbook_register_taxonomies(){

    $labels = array(
        'name'          => _x( 'Technologies', 'taxonomy general name', '26-taxonomies' ),
        'singular_name' => _x( 'Technology', 'taxonomy singular name', '26-taxonomies' ),
        'search_items'  => __( 'Search Technologies', '26-taxonomies' ),
        'all_items'     => __( 'All Technologies', '26-taxonomies' ),
        'parent_item'   => __( 'Parent Technology', '26-taxonomies' ),
        'parent_item_colon' => __( 'Parent Technology:', '26-taxonomies' ),
        'edit_item'     => __( 'Edit Technology', '26-taxonomies' ),
        'update_item'   => __( 'Update Technology', '26-taxonomies' ),
        'add_new_item'  => __( 'Add New Technology', '26-taxonomies' ),
        'new_item_name' => __( 'New Technology Name', '26-taxonomies' ),
        'menu_name'     => __( 'Technologies', '26-taxonomies' ),
    );

    register_taxonomy( 'wpcookbook-technologies', 'post', array(
        'labels'          => $labels,
        'description'    => __( 'Technologies discussed in the post', '26-taxonomies' ),
        'hierarchical'   => true,
        'public'          => true,
        'show_in_rest'   => true, // Default false
        'show_admin_column' => true,
        'rewrite'         => array(
            'slug'      => 'technologies',
            'with_front' => false,
        ),
    )));
}

}
```

Aussi, ajouter `show_admin_column` permet d'afficher la taxonomie dans le tableau listant les articles. La taxonomie est aussi directement éditabile dans le formulaire des modifications rapides.

The screenshot shows the WordPress admin dashboard under the 'Articles' section. A taxonomy selector for 'Technologies' is highlighted with a yellow box. Another yellow box highlights the 'Technologies' section in the sidebar where 'PHP' and 'WordPress' are selected.

La taxonomie Technologies est bien intégrée à l'interface.

## Ce qu'il faut retenir

- On déclare une taxonomie à l'aide de la fonction `register_taxonomy()` que l'on hooke sur `init`.
- N'oubliez pas d'utiliser `flush_rewrite_rule()` pour mettre à jour vos permaliens.
- Comme pour les types de contenu personnalisé, vous pouvez personnaliser l'administration de votre type de contenu, en la faisant apparaître dans le tableau des publications, dans la boîte des modifications rapides, dans les menus, etc.
- Vous pouvez personnaliser vos URL à l'aide du paramètre `rewrite`.
- Vous pouvez aussi contrôler l'accès au contenu avec des capacités.
- Vous pouvez associer votre taxonomie à n'importe quel type de contenu.

Comme pour les types de contenu personnalisé, on n'a pas exploré tous les paramètres car il n'y a pas vraiment de bonne raison de les changer pour une utilisation courante. Mais sachez que vous pouvez personnaliser les routes de l'API REST, le contrôleur REST, modifier les variables de requêtes, etc.

Si vous avez bien saisi comment utiliser les taxonomies et les types de contenu personnalisés, alors vous avez tout ce qu'il faut pour gérer tout type de contenu !

# Comment créer une page de réglages pour une extension

Il n'y a que très peu d'extensions qui ne proposent aucun réglage. La plupart du temps, elles ajoutent leur propre élément de menu dans le menu principal de l'administration, avec plusieurs sous-menus.

Et il y a de grandes chances (environ 100%) que vous ayez besoin de savoir faire de même pour votre extension. C'est ce que l'on va voir dans ce chapitre.

Pour créer des pages de réglages et pour utiliser ces réglages, WordPress met à notre disposition deux APIs: la Settings API et la Options API. Comme pour les réglages du thème dans l'outil de personnalisation, on va passer en revue les fonctions principales de ces API, avec leurs différents paramètres. C'est parti.

## Créer une page dédiée pour nos réglages

Avant de pouvoir créer nos réglages et nos champs, on va mettre en place une page dédiée pour nos réglages. Pour cela, WordPress met à notre disposition tous les outils dont nous avons besoin. Créez donc une petite extension pour faire nos tests.

On va utiliser les fonctions `add_menu_page()` et `add_submenu_page()`. La différence est très simple. `add_menu_page()` va permettre de déclarer un élément de menu principal, qui va donc venir encombrer votre menu principal, alors qu'`add_submenu_page()` va créer une page dans un menu déjà existant comme Réglages, par exemple.

```
add_menu_page( string $page_title, string $menu_title, string $capability, string $menu_slug, callable $function = '', string $icon_url = '', int $position = null )
```

La fonction `add_menu_page()` prend 7 paramètres :

- `$page_title` est un titre de votre page, qui va apparaître dans la balise `<title>` entre autres.
- `$menu_title` est le titre à afficher dans le menu d'administration de WordPress
- `$capability` est la capacité nécessaire pour afficher l'élément de menu et le contenu de la page. `manage_options` est une bonne option, car en général, seuls les administrateurs ont le droit de toucher aux réglages.
- `$menu_slug` est un identifiant/slug pour la page. Donc composé de minuscules, tirets, et underscores.
- `$function` est la fonction de rappel à utiliser pour afficher le contenu. Cette fonction n'est pas appelée si l'utilisateur n'a pas les capacités requises.
- `$icon_url` est l'URL de l'icône à utiliser pour le menu. On peut passer une data URL ou alors un slug de Dashicons.
- `$position` est un entier représentant la position du menu dans la liste.

Dans notre extension, ajoutez ceci :

```
// in main plugin file
add_action( 'admin_menu', 'wpcookbook_settings_menu' );
/**
 * Registers our new menu item
 */
function wpcookbook_settings_menu() {
    $hookname = add_menu_page(
        __( 'WPCookBook', '27-settings' ), // Page title
        __( 'WPCookBook', '27-settings' ), // Menu title
        'manage_options', // Capabilities
        'wpcookbook-page', // Slug
        'wpcookbook_menu_page_callback', // Display callback
        'dashicons-carrot', // Icon
        66 // Priority/position. Just after 'Plugins'
    );
}

/**
 * Displays our new settings page content
 */
function wpcookbook_menu_page_callback(){
    ?>
    <div class="wrap">
        <h1><?php echo esc_html( get_admin_page_title() ); ?></h1>
    </div>
    <?php
}
```

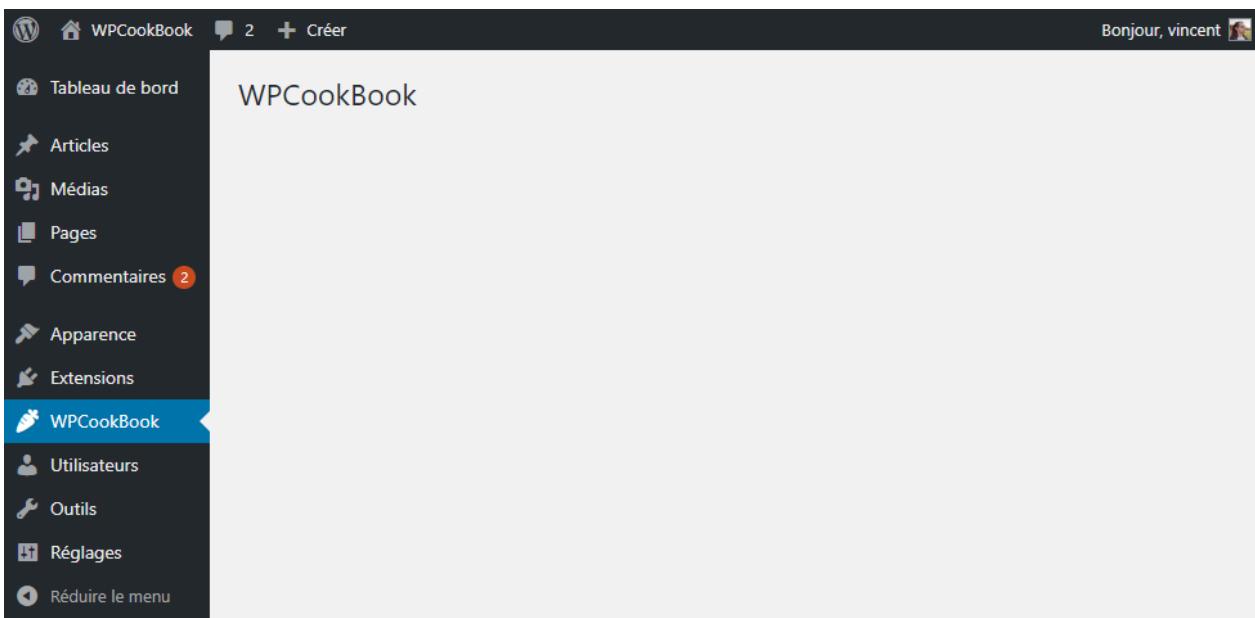
Pour déclarer un menu, on se hooke sur `admin_menu`. Puis on utilise la fonction `add_menu_page()` en passant tous les paramètres nécessaires. Seul les quatres premiers sont requis mais en général ils sont tous utiles. Si vous voulez jouer sur la position du menu, voilà un tableau décrivant les positions de chaque menu par défaut :

| Position | Menu            |
|----------|-----------------|
| 2        | Tableau de bord |
| 4        | Séparation      |
| 5        | Articles        |
| 10       | Médias          |
| 20       | Pages           |
| 25       | Commentaires    |
| 59       | Séparation      |
| 60       | Apparence       |

| Position | Menu         |
|----------|--------------|
| 65       | Extensions   |
| 70       | Utilisateurs |
| 75       | Outils       |
| 80       | Réglages     |
| 99       | Séparation   |

La fonction renvoie un nom de hook, sur lequel sera hookée notre fonction de rappel. On n'en a pas encore besoin pour l'instant, mais c'est toujours utile de le savoir.

Puis, on déclare une fonction de rappel responsable de l'affichage du contenu de notre page. Pour l'instant, elle utilise simplement `get_admin_page_title()` pour récupérer le titre de notre page et l'afficher dans un `<h1>`. Aussi, il est recommandé de mettre notre contenu dans une `<div>` avec la classe CSS `wrap`, utilisée un peu partout par WordPress.



Notre page est prête à accueillir ses réglages.

Si vous préférez glisser votre page dans un sous-menu, comme Réglages par exemple, rien de plus simple. Il faut utiliser `add_submenu_page()` à la place.

```
add_submenu_page( string $parent_slug, string $page_title, string $menu_title,
string $capability, string $menu_slug, callable $function = '', int $position =
null )
```

La fonction prend les mêmes paramètres qu'`add_menu_page()`, à deux exceptions près : on doit lui passer le slug d'une page parente, et on ne peut pas lui passer d'icône. On peut donc ajuster notre

fonction de rappel hookée sur `admin_init`.

```
// in main plugin file
function wpcookbook_settings_menu() {
    $hookname = add_submenu_page(
        'options-general.php',           // Parent slug
        __( 'WPCookBook', '27-settings' ), // Page title
        __( 'WPCookBook', '27-settings' ), // Menu title
        'manage_options',               // Capabilities
        'wpcookbook-page',              // Slug
        'wpcookbook_menu_page_callback', // Display callback
        10                             // Priority/position.
    );
}
```



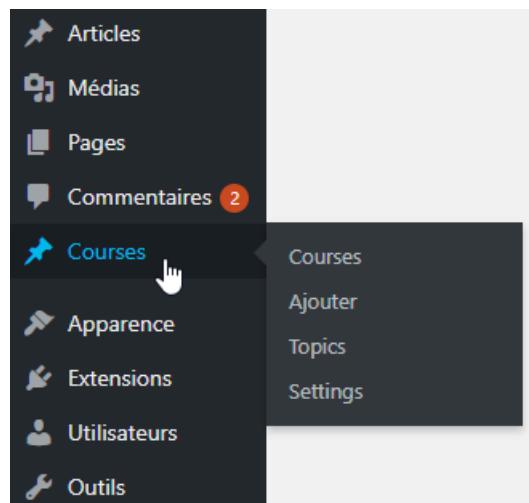
Dans un sous-menu, c'est mieux.

Les priorités vont varier d'un sous-menu à l'autre, donc pour ça il faudra tâtonner un peu, désolé. Par contre, voici les slugs des pages parentes :

| Page                           | Slug                              |
|--------------------------------|-----------------------------------|
| Tableau de bord                | index.php                         |
| Articles                       | edit.php                          |
| Médias                         | upload.php                        |
| Pages                          | edit.php?post_type=page           |
| Commentaires                   | edit-comments.php                 |
| Types de contenu personnalisés | edit.php?post_type=your_post_type |
| Apparence                      | themes.php                        |

| Page               | Slug                |
|--------------------|---------------------|
| Extensions         | plugins.php         |
| Utilisateurs       | users.php           |
| Outils             | tools.php           |
| Réglages           | options-general.php |
| Réglages du réseau | settings.php        |

Ce qui est vraiment sympa, c'est qu'on peut aussi passer le slug d'un type de contenu personnalisé. Ce qui veut dire que par exemple, on peut créer une extension qui va enregistrer un type de contenu et ranger les réglages relatifs à ce type de contenu dans son sous-menu.



Vous pouvez placer vos pages de réglages où vous voulez !

Si vous souhaitez ranger votre page sous un menu par défaut de WordPress, il existe aussi des fonctions raccourci qui vont appeler directement `add_submenu_page()` avec le slug qui convient : `add_dashboard_page()` pour faire apparaître votre sous-menu sous Tableau de bord, `add_posts_page()`, `add_media_page()`, `add_comments_page()`, `add_pages_page()`, `add_theme_page()`, `add_plugins_page()`, `add_management_page()` pour les Outils, et `add_options_page()` pour les Réglages.

Il est recommandé de ne pas polluer l'administration avec des éléments de menus principaux supplémentaires. Autant que faire se peut, on va ranger nos réglages dans les pages par défaut de WordPress. Pour la suite, on va garder notre page sous Réglages, ce qui est tout indiqué.

## Créer des réglages

Votre page est prête. Il faut maintenant y ajouter un formulaire avec nos champs. On pourrait, dans notre fonction de rappel affichant le contenu de la page, insérer une balise <form>, nos champs <input> et traiter le formulaire sur cette même page, en hookant sur `load-$hookname`, où `$hookname` est le nom du hook retourné par `add_submenu_page()`.

```
// in main plugin file
function wpcookbook_settings_menu() {
    $hookname = add_submenu_page(
        'options-general.php',           // Parent slug
        __( 'WPCookBook', '27-settings' ), // Page title
        __( 'WPCookBook', '27-settings' ), // Menu title
        'manage_options',               // Capabilities
        'wpcookbook-page',              // Slug
        'wpcookbook_menu_page_callback', // Display callback
        10                             // Priority/position.
    );

    add_action( 'load-' . $hookname, 'wpcookbook_handle_settings' );
}
```

Mais non, on ne va pas faire comme ça. On est là pour apprendre à faire les choses correctement, the WordPress Way. Et dans notre cas, la WordPress Way nous recommande d'utiliser la Settings API.

Normalement, on crée une page de réglages pour paramétrier certaines fonctionnalités d'une extension. Dans notre cas, on n'a pas vraiment d'extension à développer à proprement parler. Les exemples qui vont suivre sont donc juste pour le sport, l'objectif étant de vous montrer les mécaniques propres à WordPress et la Settings API, ok ?

### Enregistrer un réglage

Dans notre page WPCookBook créée plus tôt, on va créer une série de réglages de types différents. Pour l'instant, mettons en place notre premier champ texte. Créer une page de réglages est en fait assez simple. Les trois fonctions principales à connaître sont : `register_setting()`, `add_settings_section()` et `add_settings_field()`.

Avant de pouvoir utiliser un réglage, il faut le déclarer. On fait cela avec la fonction `register_setting()`, que l'on va hooker sur `admin_init`.

```

// in main plugin file
add_action( 'admin_init', 'wpcookbook_register_settings' );
/**
 * Registers our new settings
 */
function wpcookbook_register_settings(){
    register_setting(
        'wpcookbook',           // Group Name
        'wpcookbook_text_field', // Setting Name
        array(
            'type'          => 'string',           // Value type
            'description'   => __( 'Simple text field', '27-settings' ), // Description
            'sanitize_callback' => 'sanitize_text_field', // Sanitize
            'callback'       => null,
            'show_in_rest'   => false,             // Whether to make available in REST API
            'default'        => '',                // Default
            'value'          => null
        )
    );
}

```

```

register_setting( string $option_group, string $option_name, array $args = array() )

```

La fonction prend **trois paramètres** : le nom du groupe de réglages, le nom du réglage et un tableau d'arguments. Le nom du groupe `$option_group` doit correspondre au groupe de réglages auquel vous ajoutez le vôtre. Pour notre page de réglages, nous créons notre propre groupe, mais il est possible d'enregistrer un réglage pour l'afficher sur une page et dans un groupe déjà existant.

Le deuxième paramètre `$name` doit être un nom unique pour votre réglage. Il convient de le préfixer pour éviter tout conflit.

Le troisième paramètre est un tableau, et peut prendre les valeurs suivantes :

- `type` est le type correspondant à la valeur du champ : `string`, `boolean`, `integer`, `number`, `array`, ou `object`.
- `description` est utile uniquement pour l'API REST.
- `sanitize_callback` est une fonction de rappel utilisée pour nettoyer la valeur de notre réglage.
- `show_in_rest` détermine si le réglage va être disponible dans l'API REST.
- `default` est une valeur par défaut pour le réglage.

Les paramètres `type`, `description` et `show_in_rest` ne sont utiles que si vous souhaitez rendre le réglage disponible dans l'API REST. Vous pouvez les omettre sans souci. Aussi, `default` est optionnel. Le paramètre le plus important est la `sanitize_callback` !

Si vous n'avez pas besoin de l'API REST, vous pouvez même déclarer votre réglage en passant simplement une `sanitize_callback` en troisième paramètre, comme ceci :

```

register_setting(
    'wpcookbook',           // Group name
    'wpcookbook_text_field', // Setting name
    'sanitize_text_field',   // Sanitize callback
);

```

## Ajouter une section

Notre réglage est simplement déclaré. Il faut maintenant afficher le champ. Pour ce faire, on va d'abord créer une **section** pour l'accueillir. On crée une section avec `add_settings_section()`, qui prend 4 paramètres.

```
add_settings_section( string $id, string $title, callable $callback, string $page )
```

`$id` est l'identifiant de la section à ajouter, `$title` est un titre traduisible, `$callback` est une fonction de rappel pour en afficher le contenu et `$page` est le slug de la page sur laquelle la section doit apparaître.

On peut donc ajuster notre fonction de rappel comme ceci :

```

// In main plugin file
function wpcookbook_register_settings(){
    register_setting(
        'wpcookbook',           // Group name
        'wpcookbook_text_field', // Setting name
        'sanitize_text_field',   // Sanitize callback
    );
    add_settings_section(
        'wpcookbook-first-section',           // Section ID
        __( 'First section', '27-settings' ), // Title
        'wpcookbook_first_section_display',   // Callback
        'wpcookbook-page'                   // Page
    );
}

```

Et on va déclarer la fonction de rappel censée afficher notre section. Pas besoin d'y afficher notre champ pour le moment. Vous allez comprendre, pas de panique. Aussi, la fonction de rappel prend comme paramètre les arguments passés à `add_settings_section()`, ce qui est pratique.

```

// In main plugin file
/**
 * Displays the content of the WPCookBook settings page first section
 *
 * @param array $args Arguments passed to the add_settings_section() function call
 */
function wpcookbook_first_section_display( $args ){
    printf( '<p><strong>%s</strong></p>', esc_html( $args['title'] ) );
}

```

Pour l'instant, on affiche juste le titre de la section. Notez qu'il est tout à fait possible d'omettre cette fonction de rappel si l'on n'a rien de spécial à afficher mis à part nos champs. Il suffit de passer une chaîne vide pour troisième paramètre à add\_settings\_section().

## Ajouter un champ

Maintenant, ajoutons-lui un champ avec add\_settings\_field().

```
add_settings_field( string $id, string $title, callable $callback, string $page,
string $section = 'default', array $args = array() )
```

Les paramètres sont assez explicites : \$id est un identifiant, \$title un titre traduisible, \$callback la fonction de rappel qui va afficher notre champ, \$page et \$section sont la page et la section où afficher notre champ, et \$args un tableau d'arguments optionnels dans lequel on peut passer une class CSS et label\_for qui permet d'utiliser le \$title dans un <label>.

```
// In main plugin file
function wpcookbook_register_settings(){
    register_setting( ... );
    add_settings_section( ... );
    add_settings_field(
        'wpcookbook_text_field', // Field ID
        __( 'Simple text field', '27-settings' ), // Title
        'wpcookbook_text_field_display', // Callback
        'wpcookbook-page', // Page
        'wpcookbook-first-section', // Section
        array(
            'label_for' => 'wpcookbook_text_field', // Label
            'class'      => 'wpcookbook-text-field', // CSS Classname
        )
    );
}
```

Et notre fonction de rappel prend aussi les arguments \$args passés à add\_settings\_field() en paramètre :

```
/**
 * Displays our simple text field setting
 *
 * @param array $args Arguments passed to corresponding add_settings_field() call
 */
function wpcookbook_text_field_display( $args ){
    $value = get_option( 'wpcookbook_text_field' ) ?: '';
    ?>
        <input id=<?php echo esc_attr( $args['label_for'] ); ?>" type="text"
name="wpcookbook_text_field" value=<?php echo esc_attr( $value ); ?>>
    <?php
}
```

On utilise get\_option() pour récupérer la valeur du réglage, vérifier si elle existe et l'assigner à \$value,

`get_option()` renvoyant `false` si l'option est introuvable. Puis, on affiche un simple champ `<input>`, dont l'attribut `id` correspond à celui déclaré. Il est possible d'avoir un attribut `id` différent, mais l'attribut `name` doit correspondre au nom du réglage passé dans `register_setting()`.

On utilise ici l'opérateur ternaire `? :`. Cela signifie que la valeur de `get_option()` est assignée à `$value` si elle existe, sinon ce sera une chaîne vide qui sera assignée. C'est le raccourci d'un opérateur ternaire, qui lui-même est un raccourci pour un `if / else`. Les trois expressions suivantes sont donc équivalentes.

```
$value = get_option( 'wpcookbook_text_field' ) ?: '' ;
$value = get_option( 'wpcookbook_text_field' ) ? get_option( 'wpcookbook_text_field' ) : '';
;

if( get_option( 'wpcookbook_text_field' ) ){
    $value = get_option( 'wpcookbook_text_field' );
} else {
    $value = '';
}
```

## Retour sur notre page WPCookBook

On a créé un réglage, ajouté une section et ajouté un champ. Maintenant, si on va sur notre page WPCookBook, rien ne s'affiche. C'est simplement parce qu'on n'a pas indiqué à WordPress d'afficher les sections et réglages correspondants à notre page. On va donc modifier notre fonction de rappel affichant le contenu de la page :

```
/**
 * Displays our new settings page content
 */
function wpcookbook_menu_page_callback(){
    ?>
        <div class="wrap">
            <h1><?php echo esc_html( get_admin_page_title() ); ?></h1>
            <form action="options.php" method="POST">
                <?php
                    settings_fields( 'wpcookbook' );
                    do_settings_sections( 'wpcookbook-page' );
                    submit_button();
                ?>
            </form>
        </div>
    <?php
}
```

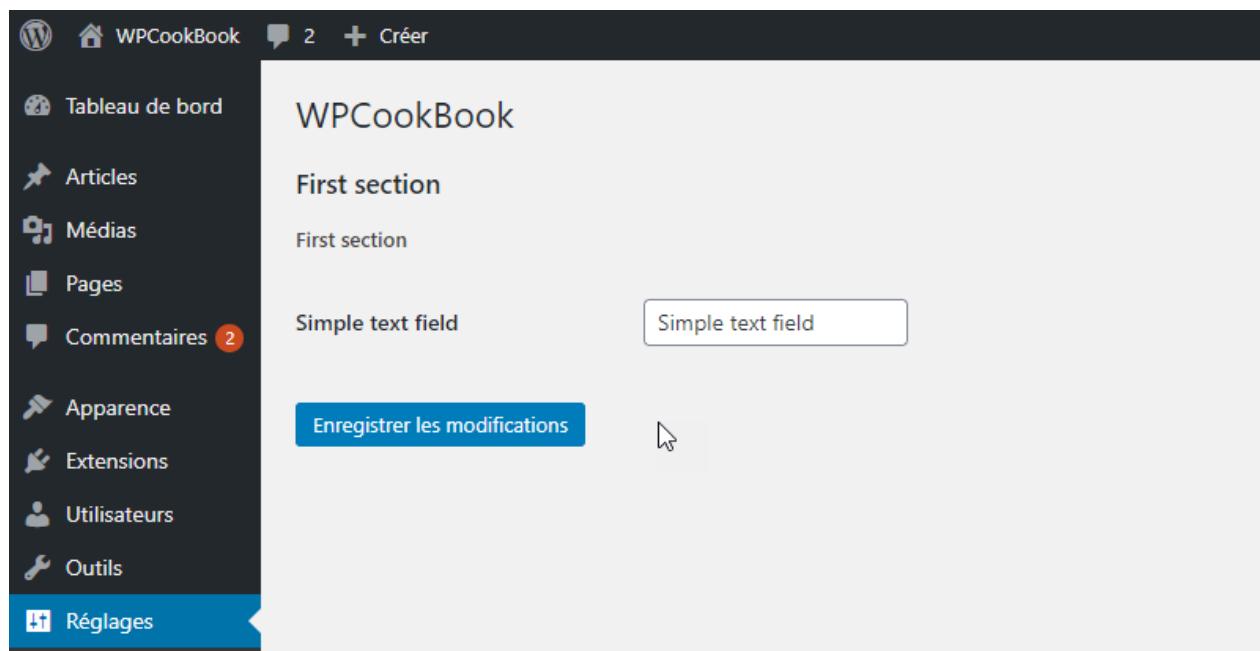
Il suffit d'une balise `<form>` et de trois fonctions de WordPress pour que notre page de réglages fonctionne. Le formulaire doit pointer vers `options.php` en envoyant ses données en `POST`. Puis on utilise `settings_fields()` pour afficher tous les champs cachés nécessaires au bon traitement des données du formulaire, `do_settings_sections()` pour afficher toutes les sections et les champs associés à la page passée en paramètre, et enfin un `submit_button()` pour envoyer le formulaire.

`settings_fields( string $option_group )` prend un seul paramètre qui correspond au nom du

groupe de réglages à afficher sur la page, et affiche automatiquement les champs nonce, action, referer et option\_page qui vont servir pour traiter automatiquement toute la sécurité du formulaire.

do\_settings\_sections( string \$page ) prend uniquement un nom de page et va afficher toutes les sections et champs associés à cette page.

submit\_button( string \$text = null, string \$type = 'primary', string \$name = 'submit', bool \$wrap = true, array|string \$other\_attributes = null ) affiche un bouton de soumission du formulaire. On peut personnaliser le texte du bouton \$text, le type de bouton \$type (primary par défaut, mais on peut passer small, large ou une classe CSS personnalisée), ou l'attribut name. On peut aussi le placer ou non dans un paragraphe (\$wrap) et lui ajouter des attributs supplémentaires (\$other\_attributes).



Notre page fonctionne enfin !

WordPress s'occupe de générer tout le balisage nécessaire autour de notre champ. À la sauvegarde des modifications, les champs de sécurité générés par settings\_fields() sont automatiquement vérifiés, notre sanitize\_callback est automatiquement appelée et nos réglages sauvegardés dans la table wp\_options. Magic.

Si vous souhaitez plus de contrôle sur le code HTML, c'est possible, car il existe aussi la fonction do\_settings\_fields() qui va vous permettre d'afficher uniquement les champs associés à une section. Vous pouvez donc l'utiliser dans les fonctions de rappel de vos sections. Entre les fonctions de rappel des sections, des champs, de la page, et ces deux fonctions do\_settings\_fields() et do\_settings\_sections(), vous devriez pouvoir vous en sortir.

## Petit point intermédiaire

Avant d'aller plus loin, faisons un petit point intermédiaire.

Pour créer un élément de menu et la page associée :

- On crée un élément de menu de premier niveau dans le menu d'administration de WordPress avec `add_menu_page()`, hooké sur `admin_menu`.
- On crée un élément d'un sous-menu avec `add_submenu_page()`. On peut aussi utiliser les fonctions `add_dashboard_page()`, `add_theme_page()`, `add_options_page()` pour les créer directement sous le bon élément principal.

Pour créer un réglage :

- On déclare un réglage en utilisant `register_setting()`, hooké sur `admin_init`. N'oubliez pas la `sanitize_callback` ! Aussi, utilisez le nom du groupe correspondant à celui auquel vous voulez ajouter votre réglage.
- On déclare une section pour nos réglages avec `add_settings_section()`.
- On déclare nos champs avec `add_settings_fields()`. Par contre, il faut le baliser manuellement dans la fonction de rappel.
- On utilise `get_option()` pour récupérer la valeur d'un réglage ou `false` s'il n'existe pas.

Pour afficher nos réglages :

- On crée un `<form>` envoyant ses données vers `options.php`, en POST.
- On utilise `settings_fields()` pour ajouter les champs de sécurité cachés nécessaires.
- `do_settings_sections()` va afficher nos sections et les champs associés.
- `submit_button()` va créer le bouton pour envoyer le formulaire.

## Réglages et erreurs

Pour l'instant nous disposons uniquement d'un simple champ texte, qui est passé dans la fonction `sanitize_text_field()`. Donc si on tente d'ajouter des balises HTML dans le champ texte, elles sont purement et simplement supprimées sans feedback. Ca serait sympa d'afficher un petit message pour prévenir quand même, non ?

On peut aussi utiliser notre propre `sanitize_callback`, ce qui nous permet d'afficher des notices utiles quand les réglages de la page sont sauvegardés. Ajustez l'appel à `register_setting()` en passant une fonction personnalisée :

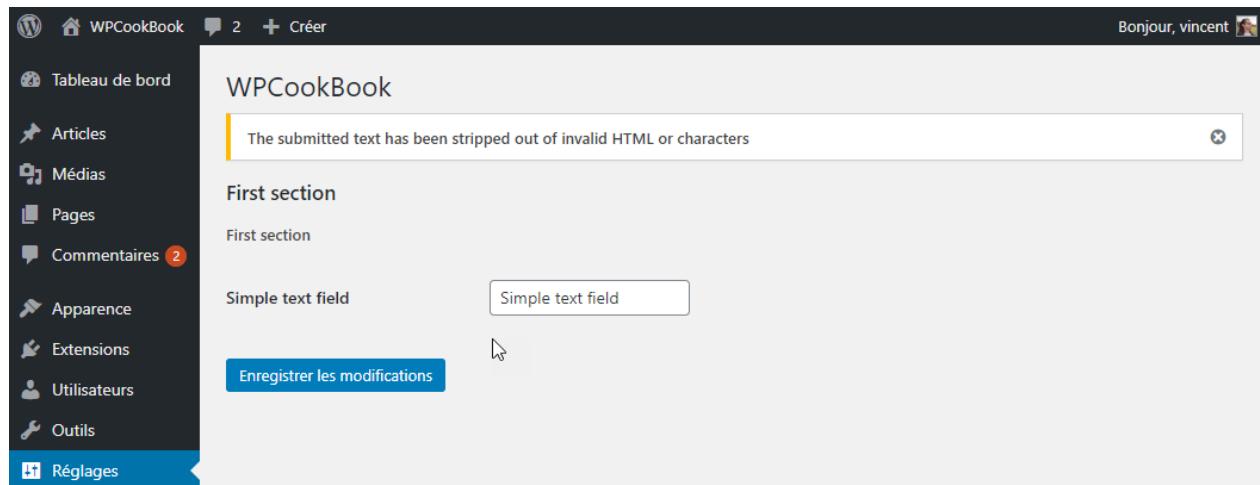
```
// in main plugin file, in the callback hooked on 'admin_init'  
register_setting( 'wpcookbook', 'wpcookbook_text_field', 'wpcookbook_sanitize_text_field' );
```

Puis créons cette fonction personnalisée. Elle doit recevoir la valeur du réglage et la retourner nettoyée.

```
// In main plugin file
/**
 * Sanitizes our simple text field
 *
 * @param string $value The value of our text field
 * @return string $value
 */
function wpcookbook_sanitize_text_field( $value ){
    $clean = sanitize_text_field( $value );
    if( $clean !== $value ){
        add_settings_error( 'wpcookbook_text_field', 'invalid', __( 'The submitted text has been stripped out of invalid HTML or characters', '27-settings' ), 'warning' );
    }
    return $clean;
}
```

`add_settings_error( string $setting, string $code, string $message, string $type = 'error' )` permet de déclarer des erreurs qui seront affichées automatiquement sous forme de notices natives de WordPress. `$setting` doit correspondre au réglage pour lequel on déclare l'erreur, `$code` est un identifiant qui apparaîtra dans l'attribut `id` de la notice, `$message` est le message à afficher et `$type` est le type de notice à afficher. Par défaut, c'est une `error`, mais on peut aussi afficher des `warning`, `success` ou `info`.

Dans notre exemple, je vérifie si du code HTML ou des caractères invalides ont été nettoyés par `sanitize_text_field()`, mais puisque je traite le réglage et le sauvegarde quand même, j'affiche juste un avertissement.



Les notices sont affichées automatiquement.

## Un second réglage

Ajoutons un second réglage, d'un type différent.

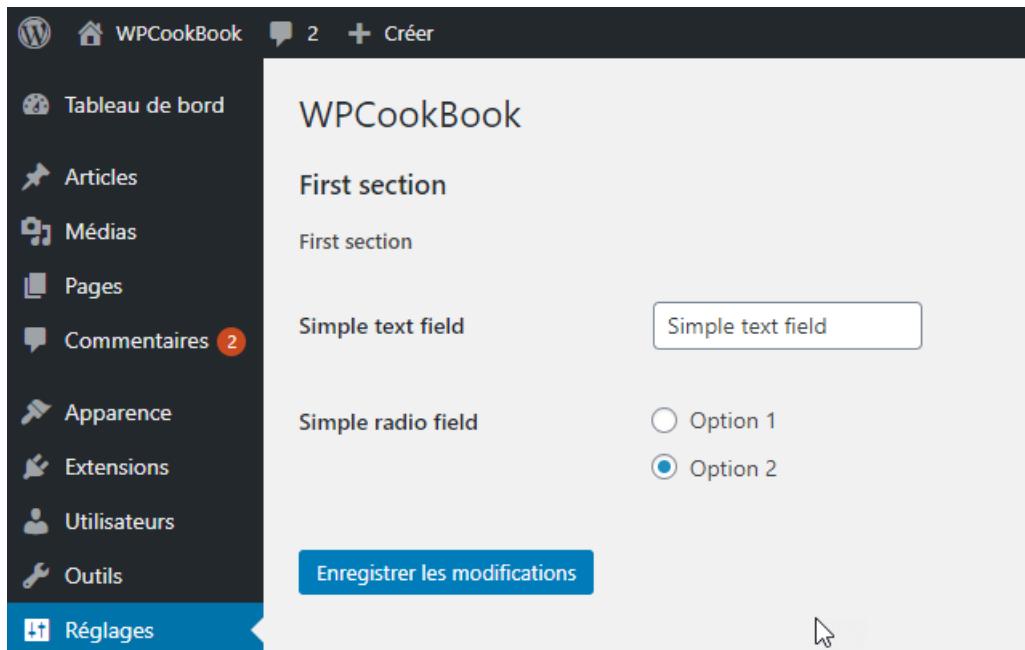
```
// In main plugin file
add_action( 'admin_init', 'wpcookbook_register_settings' );
/**
 * Registers our new settings
 */
function wpcookbook_register_settings(){
    register_setting( 'wpcookbook', 'wpcookbook_text_field', 'wpcookbook_sanitize_text_field'
);
    add_settings_section( ... );
    add_settings_field( ... );

    register_setting( 'wpcookbook', 'wpcookbook_radio_field', 'sanitize_key' );
    add_settings_field(
        'wpcookbook_radio_field', // Field ID
        __( 'Simple radio field', '27-settings' ), // Title
        'wpcookbook_radio_field_display', // Callback
        'wpcookbook-page', // Page
        'wpcookbook-first-section', // Section
    );
}
```

Et la fonction de rappel pour notre champ.

```
// In main plugin file
/**
 * Displays our simple radio field setting
 *
 * @param array $args Arguments passed to corresponding add_settings_field() call
 */
function wpcookbook_radio_field_display( $args ){
    $value = get_option( 'wpcookbook_radio_field' ) ?: 'option1' ;
    ?>
    <fieldset>
        <label for="option1">
            <input id="option1" type="radio" name="wpcookbook_radio_field" value="option1"
<?php checked( $value, 'option1' ); ?> />
            <span><?php esc_html_e( 'Option 1', '27-settings' ); ?></span>
        </label><br />
        <label for="option2">
            <input id="option2" type="radio" name="wpcookbook_radio_field" value="option2"
<?php checked( $value, 'option2' ); ?> />
            <span><?php esc_html_e( 'Option 2', '27-settings' ); ?></span>
        </label>
    </fieldset>
<?php
}
```

On utilise la fonction `checked()` pour automatiquement afficher l'attribut `checked` si les deux valeurs passées sont identiques. La fonction est très pratique pour gérer les champs de type checkbox et radio.



Le champs s'affichent automatiquement sans souci.

Le champ fonctionne et sauvegarde notre valeur sans problème. Il passe la valeur du champ dans la fonction `sanitize_key` qui va nettoyer la chaîne de façon à ce qu'elle puisse être utilisée comme identifiant. Elle ne contiendra donc que des minuscules, chiffres, tirets, et underscores.

Par contre, le code HTML peut être amélioré. La fonction de rappel prend en paramètres les arguments passés à `add_settings_field()`. Mais ce qui est bien pratique, c'est que l'on peut ajouter autant d'arguments personnalisés que l'on souhaite. On peut donc passer un tableau des choix possibles et utiliser une boucle dans notre fonction de rappel. Comme ceci :

```
// in wpcookbook_register_settings() function
add_settings_field(
    'wpcookbook_radio_field', // Field ID
    __( 'Simple radio field', '27-settings' ), // Title
    'wpcookbook_radio_field_display', // Callback
    'wpcookbook-page', // Page
    'wpcookbook-first-section', // Section
    array(
        'options' => array(
            'option1' => __( 'Option 1', '27-settings' ),
            'option2' => __( 'Option 2', '27-settings' ),
        )
    ),
);
```

```

// in main plugin file
/**
 * Displays our simple radio field setting
 *
 * @param array $args Arguments passed to corresponding add_settings_field() call
 */
function wpcookbook_radio_field_display( $args ){
    $value = get_option( 'wpcookbook_radio_field' ) ?: 'option1' ;
    ?>
    <fieldset>
        <?php foreach ( $args['options'] as $option => $label ) : ?>
            <label for="<?php echo esc_attr( $option ); ?>">
                <input
                    id="<?php echo esc_attr( $option ); ?>"
                    type="radio"
                    name="wpcookbook_radio_field"
                    value="<?php echo esc_attr( $option ); ?>"
                    <?php checked( $value, $option ); ?>
                />
                <span><?php echo esc_html( $label ); ?></span>
            </label>
            <br />
        <?php endforeach; ?>
    </fieldset>
    <?php
}

```

Le champ fonctionne exactement de la même façon, mais si vous souhaitez ajouter un choix pour le champ, il suffit de déclarer la nouvelle valeur et label dans `add_settings_field()`.

## Organisation et optimisations

Jusqu'ici tout va bien, mais vous pouvez remarquer que l'on a quand même un certain nombre de fonctions à appeler et à déclarer : pour les sections, pour la page, pour chaque champ, pour le nettoyage des données, etc. Cela peut vite devenir très fouilli, notamment au niveau des noms des identifiants et fonctions, et créer des fichiers de taille conséquente. Aussi, chaque réglage implique une entrée dans la table `wp_options`.

Pour pallier à ça et faciliter les manipulations, une solution consiste à stocker les valeurs de vos réglages dans des tableaux. Vous pouvez par exemple créer un tableau par section, voire par page. Cela réduit les appels à `register_setting()` et permet de rester organisé.

Pour cela, on va ajuster notre appel à `register_setting()` :

```
register_setting( 'wpcookbook', 'wpcookbook_example_fields',
  'wpcookbook_sanitize_example_fields' );
// register_setting( 'wpcookbook', 'wpcookbook_radio_field', 'sanitize_key' );
```

On n'a plus besoin d'enregistrer un réglage pour nos boutons radios, puisqu'on va tout enregistrer dans un seul tableau. Aussi, on va appeler une seule fonction qui sera responsable du nettoyage de tous nos réglages.

On ajuste aussi nos fonctions affichant nos champs :

```
// in main plugin file
/**
 * Displays our simple text field setting
 *
 * @param array $args Arguments passed to corresponding add_settings_field() call
 */
function wpcookbook_text_field_display( $args ){
    $settings = get_option( 'wpcookbook_example_fields' );
    $value    = ! empty( $settings['text'] ) ? $settings['text'] : '';
    ?>
        <input id=<?php echo esc_attr( $args['label_for'] ); ?>" type="text"
name="wpcookbook_example_fields[text]" value=<?php echo esc_attr( $value ); ?>">
<?php
}

/**
 * Displays our simple radio field setting
 *
 * @param array $args Arguments passed to corresponding add_settings_field() call
 */
function wpcookbook_radio_field_display( $args ){
    $settings = get_option( 'wpcookbook_example_fields' );
    $value    = ! empty( $settings['radio'] ) ? $settings['radio'] : 'option1';

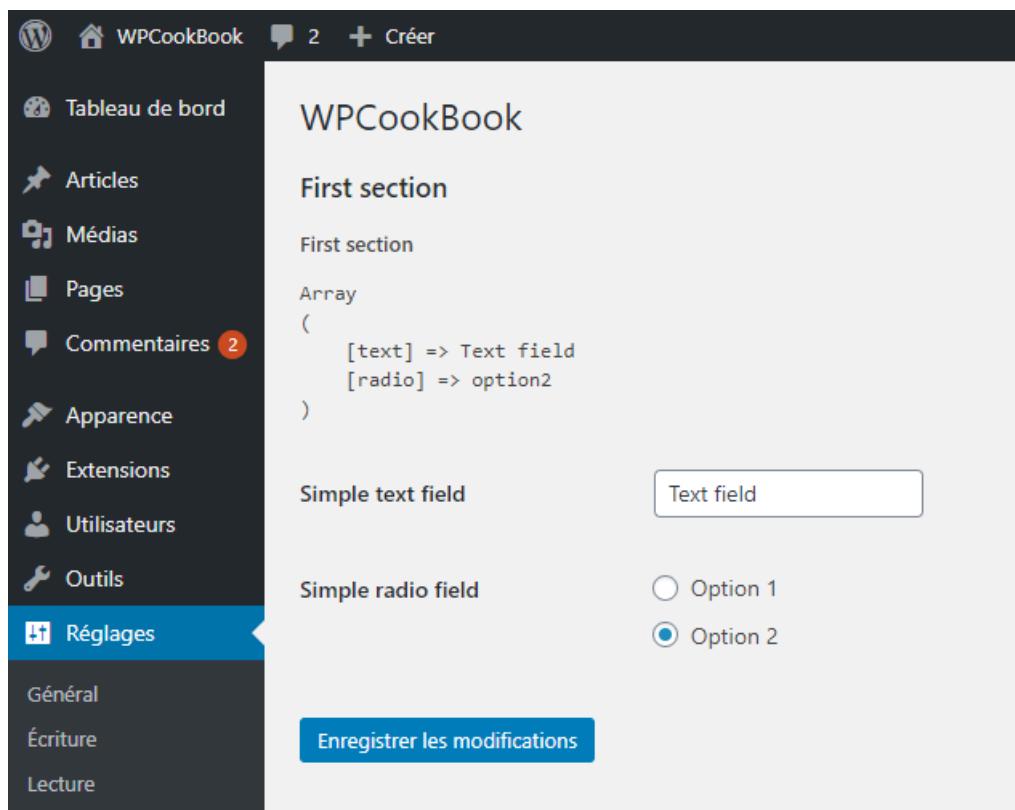
    ?>
        <fieldset>
            <?php foreach ( $args['options'] as $option => $label ) : ?>
                <label for=<?php echo esc_attr( $option ); ?>">
                    <input
                        id=<?php echo esc_attr( $option ); ?>"
                        type="radio"
                        name="wpcookbook_example_fields[radio]"
                        value=<?php echo esc_attr( $option ); ?>">
                        <?php checked( $value, $option ); ?>
                    />
                    <span><?php echo esc_html( $label ); ?></span>
                </label>
                <br />
            <?php endforeach; ?>
        </fieldset>
    <?php
}
```

Nos réglages sont dans un tableau, donc on récupère l'ensemble de nos réglages via `get_option()`. Puis on lit le tableau pour y chercher notre valeur. Les attributs `value` des champs sont donc à ajuster. Les valeurs des attributs `name` prennent un tableau au nom de notre réglage, en lui précisant la clé pour créer un tableau associatif.

Enfin, on crée notre seule et unique fonction de nettoyage :

```
/**  
 * Sanitizes our settings  
 *  
 * @param array $settings Array of values  
 * @return array $settings  
 */  
function wpcookbook_sanitize_example_fields( $settings ) {  
  
    if( ! empty( $settings['text'] ) ){  
        $text = $settings['text'];  
        $clean = sanitize_text_field( $text );  
        if( $clean !== $text ){  
            add_settings_error( 'wpcookbook_text_field', 'invalid', __( 'The submitted text has  
been stripped out of invalid HTML or characters', '27-settings' ), 'warning' );  
        }  
        $settings['text'] = $clean;  
    }  
  
    $settings['radio'] = ! empty( $settings['radio'] ) ? sanitize_key( $settings['radio'] ) :  
'option1';  
    return $settings;  
}
```

On utilise exactement la même technique que pour nos réglages simples. Il faut juste lire les valeurs dans un tableau et retourner le tableau entier, avec toutes ses valeurs nettoyées.



WPCookBook

First section

First section

Array

(

[text] => Text field

[radio] => option2

)

Simple text field

Text field

Simple radio field

○ Option 1

○ Option 2

Enregistrer les modifications

Notre groupe de champs fonctionne même si les réglages sont dans un seul tableau.

Pour vous aider à visualiser, j'ai ajusté la fonction de rappel affichant la section, en ajoutant une ligne pour afficher le contenu de notre tableau de réglages.

Vous pourrez vous rendre compte que même si WordPress nous simplifie la tâche, il reste beaucoup de répétitions dans ce code. Si vous ajoutez plusieurs champs texte simples par exemple, comment allez-vous faire ? Copier-coller une fonction de rappel, en changeant le nom des champs ? C'est possible, mais cela va rendre votre fichier illisible.

Une solution est de créer des fonctions outils qui vont générer des champs pour vous. Tous les champs texte peuvent par exemple prendre la même fonction de rappel, et vous pouvez passer ce dont vous avez besoin dans les arguments supplémentaires.

Organisez-vous comme vous le souhaitez, mais vu que déclarer des réglages est assez répétitif, cette tâche se prête très bien aux boucles. Profitez-en !

## Quelques classes CSS

Puisque vous contrôlez l'affichage de vos champs, vous pouvez très bien utiliser les classes CSS utilisées par WordPress dans l'administration pour améliorer un peu l'apparence de vos champs. La classe `regular-text` sur les champs texte permet d'avoir un champ de longueur un peu plus confortable, et on peut aussi passer une ou plusieurs classes CSS à notre bouton de sauvegarde.

Vous pourrez trouver une référence concernant le balisage et les classes CSS utilisés dans l'administration sur <https://wpadmin.bracketspace.com/>.

## Ajouter une section et un champ sur une page existante

On a vu comment créer une page et ses réglages de toute pièce, mais on peut aussi ajouter des sections et des réglages à des pages déjà existantes. Pour cela, il suffit de passer les bons slugs de section ou de page à nos fonctions. Pour s'entraîner, ajoutons une nouvelle section à notre page de réglages généraux.

```

// in main plugin file
add_action( 'admin_init', 'wpcookbook_register_additional_settings' );
function wpcookbook_register_additional_settings(){
    register_setting(
        'general',                                     // Option group
        'wpcookbook_additional_general_fields',         // Option name
        'wpcookbook_sanitize_additional_general_fields' // Sanitize callback
    );
    add_settings_section(
        'wpcookbook-additional-general-section',        // Section ID
        __( 'Additional fields', '27-settings' ),        // Title
        'wpcookbook_additional_general_section_display', // Callback
        'general'                                         // Page
    );
    add_settings_field(
        'wpcookbook_additional_text_field',               // Field ID
        __( 'Additional text field', '27-settings' ),   // Title
        'wpcookbook_additional_text_field_display',       // Callback
        'general',                                         // Page
        'wpcookbook-additional-general-section',          // Section
    );
}

```

Il suffit simplement de passer le bon groupe d'options dans `register_setting()`, et le bon slug de page dans `add_settings_section()` et `add_settings_field()`. Parmi les groupes et slugs par défaut, on trouve `general`, `writing`, `reading`, `discussion`, `media`, et `permalink`. Malheureusement, il est impossible d'ajouter des réglages à la page de confidentialité.

Je ne vais pas vous infliger de nouveau le code des différentes fonctions de rappel pour afficher et nettoyer un simple champ texte. Je vous laisse le soin de tester.

Réglages

Aperçu : novembre 26, 2019

Général

Écriture

Lecture

Discussion

Médias

Permaliens

Confidentialité

WPCookBook

Réduire le menu

Format d'heure

13 h 00 min

1:00

13:00

Personnalisé :

g:i A

H:i

g:i a

Aperçu : 1:00

[Documentation sur le formatage de la date et de l'heure.](#)

La semaine débute le

Lundi

Additional fields

Additional fields

Additional text field

Example additional text field

Enregistrer les modifications

Merci de faire de [WordPress](#) votre outil de création.

Ajouter une nouvelle section est assez simple.

## Utiliser nos réglages

Pour utiliser nos réglages sur le devant du site, ou dans notre administration, on va utiliser les fonctions de l'Options API. On a déjà utilisé la fonction `get_option()` plusieurs fois, mais ça vaut le coup d'y revenir un peu.

```
get_option( string $option, mixed $default = false )
```

La fonction prend un nom de réglage en paramètre et renvoie sa valeur, simplement. On peut lui passer une valeur par défaut à renvoyer. Sinon, elle renvoie `false`. C'est la fonction de l'Options API que vous utiliserez le plus.

```
add_option( string $option, mixed $value = '', string $deprecated = '',
string|bool $autoload = 'yes' )
```

Elle permet d'ajouter une entrée dans la table des options. Il suffit de lui passer un nom de réglage. Tous les autres paramètres sont optionnels. Même pas besoin de lui passer une valeur. Vous pouvez la mettre à jour plus tard. Le paramètre `$autoload` indique s'il faut charger automatiquement l'option quand WordPress démarre.

```
update_option( string $option, mixed $value, string|bool $autoload = null )
```

Elle permet de mettre à jour un réglage. On lui passe simplement le nom du réglage et la nouvelle valeur. On a rarement besoin de mettre à jour la valeur `$autoload`. Si la valeur n'existe pas, alors elle est ajoutée, avec `autoload` sur `yes`. Donc on n'a pas forcément besoin d'utiliser `add_option()` pour créer un

réglage. Ça c'est bien cool.

`delete_option( string $option )` supprime un réglage. Bim.

Vous remarquerez que même si on dispose de ces fonctions, on n'a pas une seule fois eu besoin d'utiliser `update_option()`. Aussi, on n'a pas eu besoin de gérer les droits utilisateur, ni de créer et vérifier un jeton de sécurité, ni vérifier d'où venaient les données à sauvegarder. Enfin, remarquez qu'on a stocké des tableaux de valeurs en base de données sans s'occuper de sérialiser nos données ou les dé-sérialiser pour les lire ! (Sérialiser, c'est transformer un tableau en une chaîne de caractères avec un format spécial pour pouvoir le stocker en base de données)

Tout a été géré automatiquement par WordPress ! On a créé notre jeton de sécurité et les autres champs cachés nécessaires grâce à une seule fonction `settings_fields()` et on a aussi affiché nos champs grâce à une seule fonction `do_settings_sections()` !

C'est le bénéfice d'utiliser la Settings API de WordPress. Tout ce qu'on a eu à faire, c'est utiliser les fonctions `register_setting()` et `add_settings_section()` et `add_settings_field()` pour tout déclarer, ainsi que quelques fonctions de rappel pour afficher l'HTML de nos champs.

Utiliser la Settings API est un gain de temps énorme car on n'a pas besoin de s'occuper ni de la sécurité, ni de la sauvegarde de nos données. On a juste besoin de `get_option()` pour récupérer la valeur de nos réglages, et les utiliser dans notre extension. C'est tout !

## Ce qu'il faut retenir

- Donnez du feedback à l'utilisateur en utilisant `add_settings_error()`, si besoin est.
- Passez les données dont vous avez besoin dans les fonctions de rappel de vos champs via les arguments optionnels de `add_settings_field()`
- Créez des fonctions génériques/outils qui vont vous aider à générer le balisage des champs. Vous pourrez ainsi réutiliser les mêmes fonctions de rappel pour les champs du même type.
- On peut stocker nos réglages dans des tableaux. C'est même mieux, car cela évite les multiples appels à `register_setting()` et les multiples fonctions de nettoyage. Un tableau de réglages par section ou page, c'est bien.
- On peut ajouter des réglages à des pages existantes très simplement. Il suffit de passer les bons slugs de pages et noms des groupes quand vous déclarez vos réglages, sections et champs.
- La Settings API s'occupe de tout ! Pas besoin de sérialiser votre tableau, pas besoin de vérifier les droits, ni les jetons de sécurité, etc.
- L'Option API est très pratique et simple d'utilisation, mais en utilisant la Setting API tout ce dont vous avez besoin, c'est `get_option()` pour lire la valeur de vos réglages.

Maintenant que vous savez créer vos réglages, on va étudier la création de metabox pour la gestion de champs personnalisés pour les publications. WordPress met des outils à notre disposition, mais cette fois-ci, on va devoir s'occuper de la sécurité ! C'est parti !

# Comment créer une metabox avec des champs personnalisés

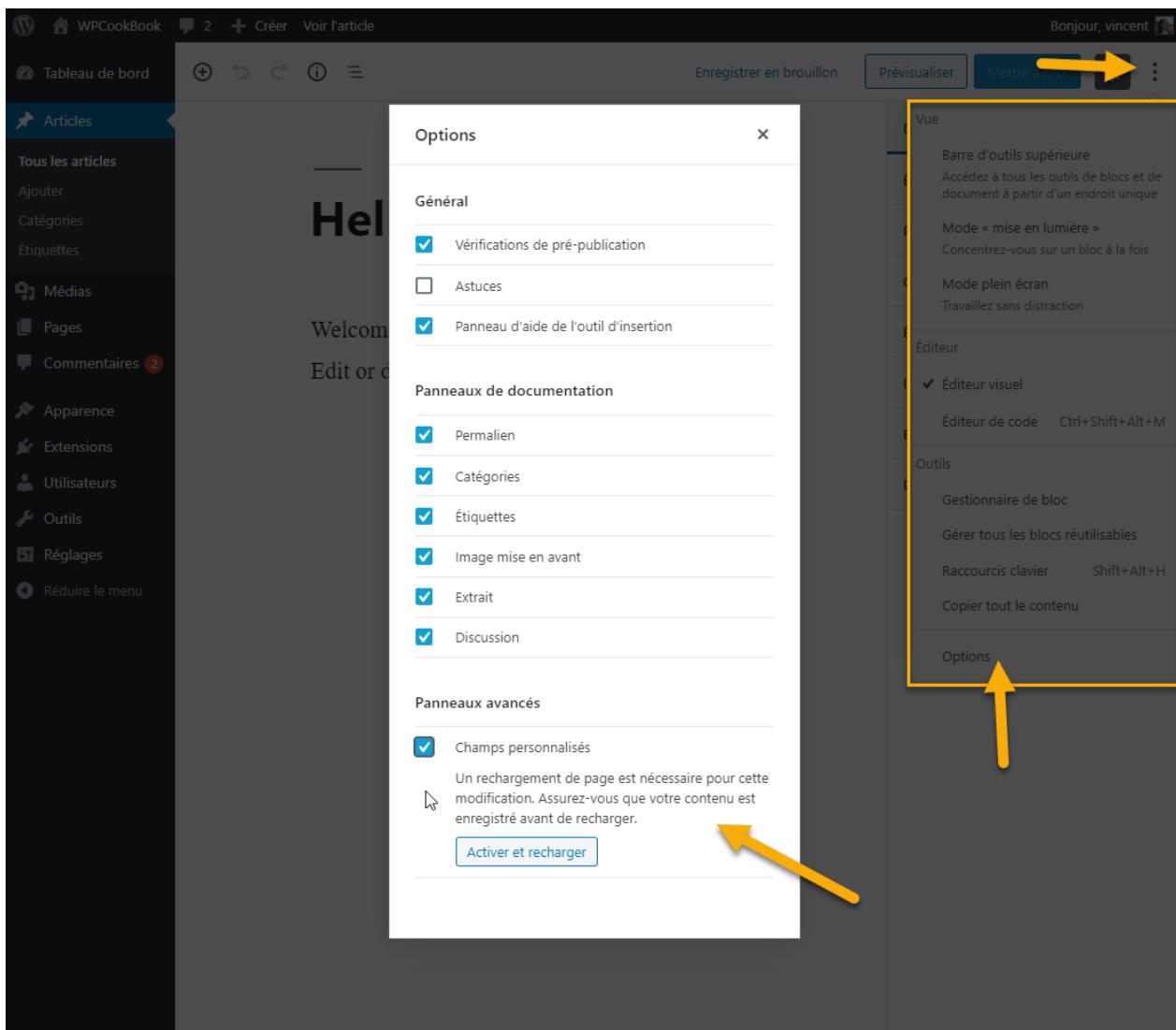
Les metaboxes sont des boites de réglages qui peuvent être ajoutées à vos pages d'édition et qui permettent d'ajouter des champs personnalisés pour vos publications. C'est un outil très puissant, qui permet d'ajouter de multiples champs et données à vos articles, pages ou tout autre type de contenu.

Vous utilisez sûrement déjà une extension comme Advanced Custom Fields pour ajouter des champs à vos pages. Mais si vous créez une extension et que vous avez besoin d'ajouter un champ personnalisé, **vous n'allez pas demander à vos utilisateurs d'installer ACF et de le configurer à votre manière !** Non, vous allez créer vous-même une metabox, gérer ses contrôles et la sauvegarde de ses données. C'est ce qu'on va voir dans ce chapitre.

## La fonctionnalité native de WordPress

WordPress propose une fonctionnalité native permettant de créer des champs personnalisés. On y a accès dans l'éditeur classique en cochant la case correspondante dans les Options de l'écran, quand vous vous situez sur une page d'édition d'une publication.

Si vous utilisez le nouvel éditeur, cette fonctionnalité est désactivée par défaut. Il faudra aller dans les options de l'éditeur en cliquant sur les trois points verticaux en haut à droite, puis dans les Options, et activer les Champs personnalisés dans les Panneaux avancés.



Pour accéder aux champs personnalisés, il faudra les activer d'abord

Une fois activés, une metabox apparaît sous notre contenu avec une interface nous permettant de sauvegarder des champs personnalisés de façon très simple sous la forme de paire clé / valeur.

The screenshot shows the 'Champs personnalisés' metabox with a table for adding new custom fields. The table has columns for 'Nom' and 'Valeur'. A row is selected with the 'Nom' field set to 'my\_custom\_field' and the 'Valeur' field set to 'Value'. A cursor is hovering over the 'Value' input field. Below the table, there is a note: 'Saisissez-en un nouveau' and a blue button labeled 'Ajouter un champ personnalisé'.

| Nom             | Valeur |
|-----------------|--------|
| my_custom_field | Value  |

Saisissez-en un nouveau

Ajouter un champ personnalisé

Les champs personnalisés peuvent être utilisés pour ajouter des métadonnées supplémentaires à une publication, que vous pouvez ensuite [utiliser dans votre thème](#).

L'interface est simple.

Ça peut être très pratique, mais c'est surtout imprévisible ! Vu que l'utilisateur à le contrôle sur le nom de la clé, vous ne pouvez pas vous baser sur cette interface pour une extension que vous créez. Vous vous imaginez documenter votre extension en expliquant que pour que ça marche, il faut donner à votre publication le champ personnalisé `toto` avec pour valeur 1 ? Impossible !

Cette interface est utile pour votre site personnel, si vous ne voulez pas vous embêter à coder une metabox, ou si vous voulez juste consulter le contenu d'un champ personnalisé existant. Mais dans 99% des cas, une metabox personnalisée avec des champs permettant des valeurs précises et contrôlées, c'est bien mieux !

## Déclarer une metabox

Pour créer une metabox, on dispose de la fonction `add_meta_box()`.

```
add_meta_box( string $id, string $title, callable $callback,  
string|array|WP_Screen $screen = null, string $context = 'advanced', string  
$priority = 'default', array $callback_args = null )
```

- `$id` est l'identifiant/slug de la metabox.
- `$title` est un titre traduisible
- `$callback` est une fonction de rappel utilisée pour afficher son contenu. Elle reçoit automatiquement l'objet `WP_Post` de la page en cours, ainsi qu'un tableau des paramètres de la metabox.
- `$screen` est l'écran sur lequel vous voulez afficher la metabox. Vous pouvez passer l'identifiant d'un écran, un tableau d'identifiants ou encore une instance de `WP_Screen`. Vous pouvez aussi lui passer l'identifiant d'un type de contenu personnalisé. Par exemple, `array( 'post', 'recipe' )` va ajouter la metabox aux pages d'éditions d'articles et de recettes (en supposant qu'un type de contenu personnalisé `recipe` soit déclaré, évidemment).
- `$context` indique la position de la metabox. Vous pouvez passer `'normal'`, `'side'` ou `'advanced'`. `'normal'` va placer la metabox sous l'éditeur de contenu, `'advanced'` va la placer sous les metaboxes placées en `'normal'` et `'side'` va placer la metabox dans la colonne latérale, avec les autres metaboxes déjà ajoutées par WordPress. Attention, si vous passez `'comment'` comme valeur pour `$screen` pour ajouter une metabox sur la page d'édition d'un commentaire, il faut obligatoirement passer `'normal'` pour `$context`. C'est un des petits pièges de WordPress.
- `$priority` va permettre d'ajuster la position de notre metabox dans son contexte. On peut passer `'high'`, `'low'`, ou `'default'`. Les metaboxes sur une même priorité sont affichées dans l'ordre dans lequel elles sont déclarées.
- `$callback_args` est un tableau d'arguments supplémentaires qui vont être passés à la fonction de rappel affichant la metabox. Cela peut être très pratique selon le besoin.

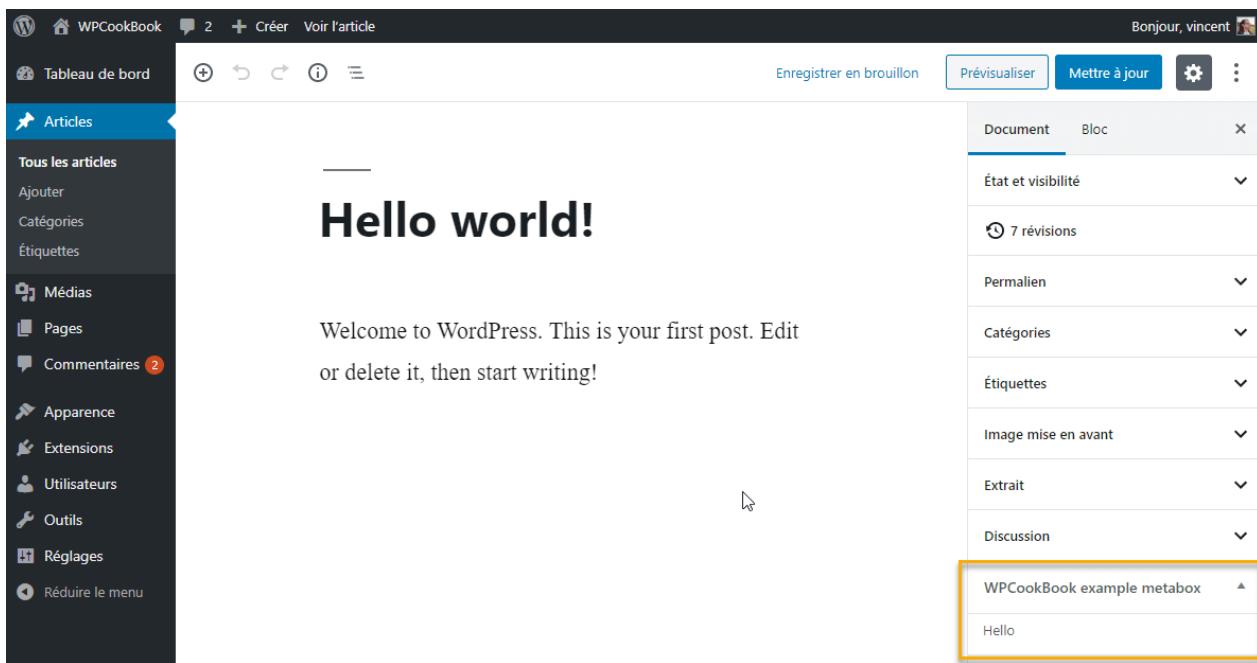
Créez une petite extension, et ajoutez-y le code suivant :

```
// in main plugin file
add_action( 'add_meta_boxes', 'wpcookbook_add_metabox' );
/**
 * Registers our new metabox
 */
function wpcookbook_add_metabox(){
    add_meta_box(
        'wpcookbook-metabox', // ID
        __( 'WPCookBook example metabox', '28-metabox' ), // Title
        'wpcookbook_metabox_display', // Callback
        'post', // Screen. Default null.
        'side', // Context. Default 'advanced'
        'low', // Priority. Default 'default'
        array( // Additional data passed to the
            'callback' // Default null
            'description' => __( 'Type in your awesome tagline!', '28-metabox' ),
        )
    );
}

/**
 * Displays our metabox content
 *
 * @param WP_Post $post The post object being edited
 * @param array $args The parameters passed to add_meta_box() call
 */
function wpcookbook_metabox_display( $post, $args ){
    echo 'Hello';
}
```

On se hooke sur `add_meta_boxes` pour utiliser `add_meta_box()`. Jusque là rien de vraiment surprenant. Mais sachez qu'on peut aussi se hooker sur `add_meta_boxes_{post_type}` pour ajouter nos metaboxes uniquement pour un certain type de contenu, et sur `add_meta_box_comment` pour les pages d'édition des commentaires.

Dans notre appel à `add_meta_box()`, on passe un identifiant, un titre, une fonction de rappel (qui n'affiche que `Hello` pour l'instant), et on spécifie que l'on souhaite rendre la metabox disponible uniquement pour les articles ('`post`'), sur le côté avec les autres metaboxes ('`side`'), et tout en bas. On passe aussi un tableau d'arguments supplémentaire pour l'exemple.



La metabox s'affiche bien en bas de la colonne.

## Ajouter du contenu

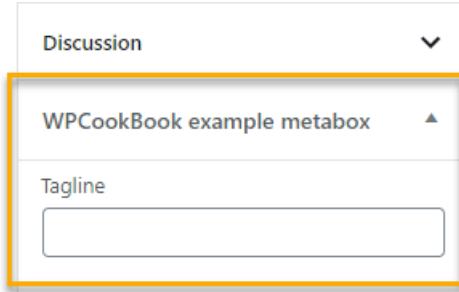
Maintenant, il faut ajouter les champs dont nous avons besoin. Supposons que nous voulions créer un champ *Sous-titre* ou *Slogan* permettant d'insérer un court texte ou une tagline et de l'afficher avec les titres des articles. Un simple champ texte fera l'affaire.

On va donc mettre à jour notre fonction de rappel.

```
// in main plugin file
function wpcookbook_metabox_display( $post, $args ){
    ?>
    <div class="components-base-control">
        <div class="components-base-control__field">
            <label for="wpcookbook-post-tagline" class="components-base-control__label">
<?php esc_html_e( 'Tagline', '28-metabox' ); ?></label>
            <input id="wpcookbook-post-tagline" name="wpcookbook-post-tagline" type="text"
class="components-text-control__input" />
        </div>
    </div>
    <?php
}
```

Le champ est un simple `<input type="text">` accompagné de son `<label>`. J'ai juste ajouté les classes CSS et le balisage utilisé par le nouvel éditeur, que vous pouvez découvrir en inspectant les champs des autres metaboxes. Libre à vous d'utiliser vos propres styles.

Aussi, il n'y a pas besoin d'ajouter de formulaire ou de bouton pour le soumettre. Les metaboxes sont déjà dans un formulaire, et les valeurs des champs seront envoyées normalement avec toutes les autres données de l'éditeur.



Notre champ s'affiche correctement.

Le champ s'affiche mais la donnée n'est pas sauvegardée. Si vous mettez à jour la valeur du champ puis l'article, vous n'aurez aucun souci mais en réalité rien ne se passe. Il faut que nous intervenions pour enregistrer notre valeur.

## Enregistrer notre valeur

Pour enregistrer notre valeur, il faut se hooker au moment où WordPress enregistre les données de l'article. Ce hook c'est `save_post`.

Il existe d'autres hooks qui sont déclenchés selon la situation et sur lesquels on peut se greffer : `wp_insert_post`, `edit_post` ou `post_updated` quand une publication est mise à jour (et selon si vous avez besoin de l'ancienne version de la publication), `edit_attachment`, `attachment_updated` ou `add_attachment` pour les téléchargements. Il y a un hook dynamique particulièrement intéressant pour nous, c'est `save_post_${post_type}` qui ne va se déclencher que pour les publications d'un certain type.

```
// in main plugin file
add_action( 'save_post_post', 'wpcookbook_metabox_save', 10, 3 );
/**
 * Saves our metabox fields
 *
 * @param int      $post_ID  Post ID.
 * @param WP_Post  $post      Post object.
 * @param bool     $update    Whether this is an existing post being updated or not.
 */
function wpcookbook_metabox_save( $post_ID, $post, $update ){
    if( isset( $_POST['wpcookbook-post-tagline'] ) ){
        update_post_meta( $post_ID, 'wpcookbook_post_tagline', sanitize_text_field(
            $_POST['wpcookbook-post-tagline'] ) );
    }
}
```

On se hooke donc sur `save_post_post` pour déclencher la fonction de rappel uniquement pour les articles. Le hook lui passe trois paramètres : l'identifiant de la publication dont il est question, l'objet `WP_Post` complet et un booléen indiquant si on est en train de créer ou de mettre à jour le contenu.

Dans notre fonction de rappel, on a accès à la super globale `$_POST` contenant toutes les informations

dont nous avons besoin. Donc on va vérifier si on a bien une valeur pour notre champ et si c'est le cas, on l'enregistre dans les métadonnées de l'article à l'aide de la fonction `update_post_meta()`.

## Manipuler les métadonnées

Chaque publication dispose de données (titre, contenu, date de publication, ...) et de métadonnées qui sont des données supplémentaires qui lui sont associées. Il convient de stocker tous les réglages ou contenus supplémentaires dans les métadonnées de la publication. Le contenu de la publication est stocké dans la table `wp_posts` et les métadonnées sont stockées dans la table `wp_postmeta`.

Et pour y avoir accès, pas besoin d'écrire une requête SQL ! WordPress met à notre disposition des fonctions outils : `add_post_meta()`, `update_post_meta()`, `get_post_meta()` et `delete_post_meta()`.

`add_post_meta( int $post_id, string $meta_key, mixed $meta_value, bool $unique = false )` permet d'ajouter une métadonnée. Il faut lui passer l'identifiant `$post_id` de la publication pour laquelle on ajoute une donnée, une clé `$meta_key` pour notre donnée et sa valeur `$meta_value`.

On peut aussi lui passer un booléen `$unique` indiquant si la clé est unique et ne peut donc pas être réutilisée. Par défaut, cela vaut `false`, ce qui implique que vous pouvez très bien utiliser `add_post_meta()` plusieurs fois pour la même publication et la même clé. Pour chaque valeur, cela va créer une métadonnée avec la même clé mais un identifiant différent. Cela peut être très pratique. La fonction renvoie l'identifiant de la métadonnée ou `false` si l'opération échoue.

`update_post_meta( int $post_id, string $meta_key, mixed $meta_value, mixed $prev_value = '' )` prend les mêmes premiers paramètres et va mettre à jour une métadonnée existante ou la créer si elle n'existe pas. Le dernier paramètre permet de cibler la valeur à modifier si plusieurs valeurs sont associées à la même clé. Si ce paramètre est omis, toutes les valeurs associées à la clé sont modifiées. Elle retourne l'identifiant de la métadonnée si celle-ci vient d'être créée, ou un booléen selon si l'opération a réussi ou pas.

`get_post_meta( int $post_id, string $key = '', bool $single = false )` permet de récupérer une valeur associée à une publication et une clé. Omettre la clé permet de récupérer toutes les métadonnées associées à la publication passée en premier paramètre. Le troisième paramètre `$single` est un booléen indiquant s'il faut récupérer une seule valeur ou toutes les valeurs associées à la clé. Par défaut, toutes les valeurs sont retournées dans un tableau. La fonction renvoie `false` si elle n'a rien trouvé.

`delete_post_meta( int $post_id, string $meta_key, mixed $meta_value = '' )` permet de supprimer une métadonnée complètement, ou seulement une de ses valeurs, si vous lui passez le troisième paramètre `$meta_value`.

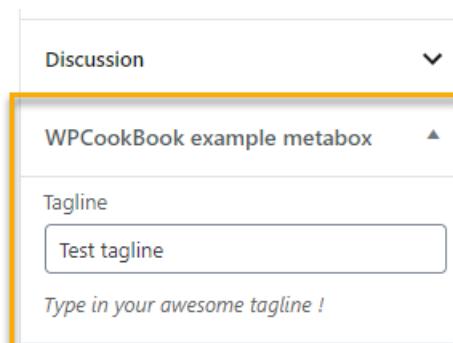
Dans notre fonction de rappel hookée sur `save_post`, on utilise `update_post_meta()` pour créer ou remplacer la valeur de la métadonnée ayant pour clé `wpcookbook_post_tagline`. Pour cela, on lui passe l'identifiant récupéré dans les paramètres de la fonction, le nom de notre clé et la valeur récupérée dans les données du formulaire que l'on a pris soin de nettoyer.

Il faut maintenant afficher la valeur dans le champ au chargement de la page.

```
// in main plugin file
function wpcookbook_metabox_display( $post, $args ){
    $value = get_post_meta( $post->ID, 'wpcookbook_post_tagline', true ) ?: '';
    ?>
    <div class="components-base-control">
        <div class="components-base-control__field">
            <label for="wpcookbook-post-tagline" class="components-base-control__label">
<?php esc_html_e( 'Tagline', '28-metabox' ); ?></label>
            <input
                type="text"
                class="components-text-control__input"
                id="wpcookbook-post-tagline"
                name="wpcookbook-post-tagline"
                value="<?php echo esc_attr( $value ); ?>">
            />
        </div>
    </div>
    <p class="description"><?php echo esc_html( $args['args']['description'] ); ?></p>
<?php
}
```

On passe `true` à la fonction `get_post_meta()` en troisième paramètre pour ne pas récupérer un tableau mais seulement la valeur associée à la clé. Aussi la fonction renvoyant `false` si elle ne trouve rien, on peut utiliser un opérateur ternaire pour affecter sa valeur à une variable. Puis on échappe sa valeur dans l'attribut `value` du champ texte.

Aussi, j'en profite pour ajouter un petit paragraphe avec notre description, passée à notre fonction de rappel par `add_meta_box()`. Le tableau `$args` contient tous les paramètres passés à `add_meta_box()`. Donc il faudra creuser un peu dans ce tableau pour y trouver notre description.



Notre champ est complètement fonctionnel.

Tout fonctionne correctement. La metabox s'affiche uniquement sur les articles, la métadonnée est bien sauvegardée en base et s'affiche au chargement de la page. Tout fonctionne, mais on n'a pas vraiment fini.

## Nonces et permissions

Dans le traitement de notre champ hooké sur `save_post`, vous avez pu remarquer qu'on manipule la super globale `$_POST`. Cela implique qu'une requête POST a été envoyée avec les données de notre formulaire. Un formulaire est un moyen d'envoyer une requête HTTP en utilisant la méthode POST vers une URL, mais il en existe bien d'autres ! Ce qui signifie que l'envoi de données vers cette même URL peut être falsifié, par un script malveillant par exemple.

Avant de traiter les données de notre metabox, il faut donc nous assurer que la requête est authentique et vienne d'un vrai utilisateur connecté, qui a affiché la page sur son navigateur et qui a soumis le formulaire manuellement.

Pour cela, on utilise les nonces. Littéralement, une nonce est un *number used once*, c'est-à-dire un jeton de sécurité unique pour identifier une requête unique. Les nonces de WordPress ne sont pas des nonces au sens strict du terme, dans le sens où ils sont composés de chiffres et de lettres et sont générés pour un utilisateur pour 24h maximum, mais ils permettent tout de même de se protéger des failles de sécurité de type CSRF (Cross Site Request Forgery).

Il faut donc créer un nonce et l'envoyer via le formulaire avec la valeur de notre champ. Notre fonction hookée sur `save_post` va devoir intercepter le nonce et vérifier sa validité avant de traiter la valeur de notre champ.

Pour créer et manipuler les nonces, WordPress met à notre disposition plusieurs fonctions: `wp_create_nonce()`, `wp_nonce_field()`, `wp_nonce_url()`, et `wp_verify_nonce()`.

`wp_create_nonce( string|int $action = -1 )` va créer un nonce en utilisant le paramètre `$action`, l'identifiant de l'utilisateur et sa session. Chaque nonce est donc lié à un seul utilisateur et une seule action.

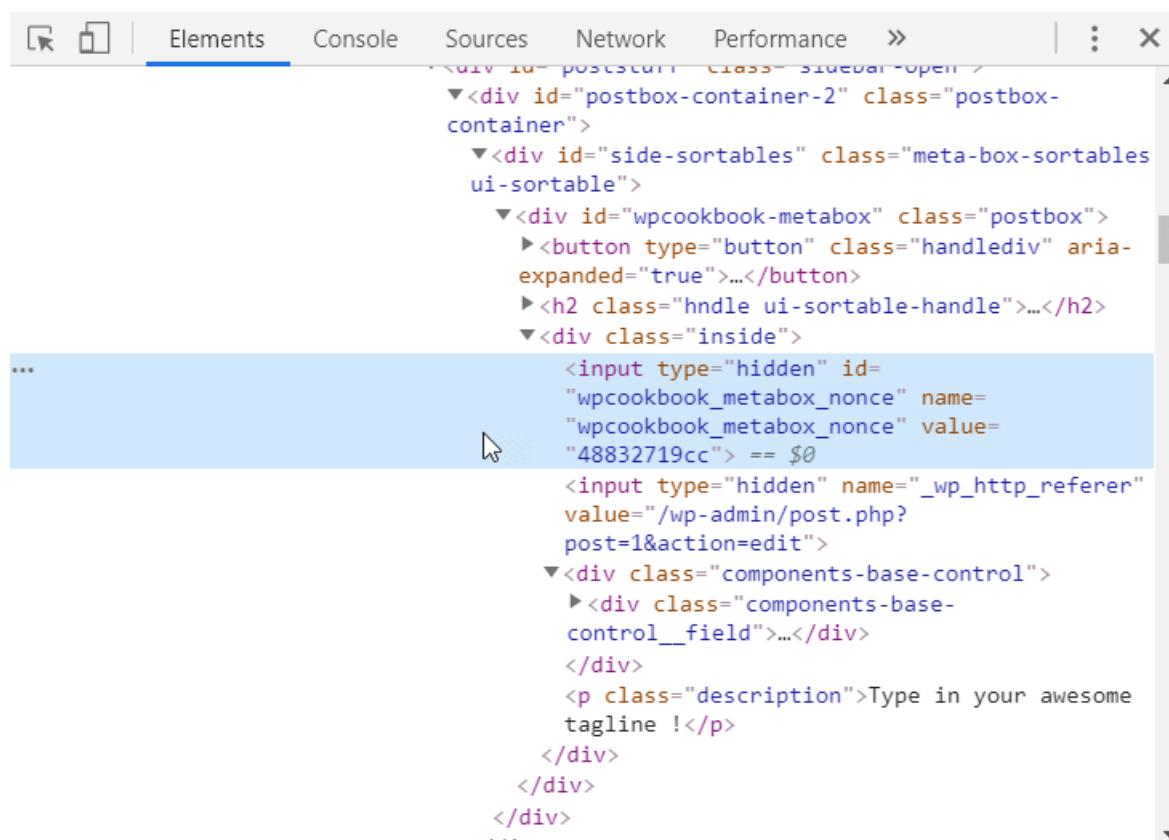
`wp_nonce_field( int|string $action = -1, string $name = '_wpnonce', bool $referer = true, bool $echo = true )` va permettre d'afficher un champ de formulaire caché contenant un nonce. Tous ses paramètres sont optionnels, cela dit il est **grandement recommandé** de passer un nom d'action unique ainsi qu'un nom unique. Chaque nonce doit être unique et lui passer une valeur unique pour le paramètre `$action` va permettre de ne pas utiliser ce même nonce pour un autre formulaire, simplement. Le paramètre `$referer` indique s'il faut créer un champ caché indiquant le référent de la requête, et `$echo` indique s'il faut afficher le champ ou juste le retourner.

`wp_nonce_url( string $actionurl, string|int $action = -1, string $name = '_wpnonce' )` va ajouter un nonce à une URL que vous lui donnez en premier paramètre. C'est la même chose que de concaténer vous-même un nonce créé avec `wp_create_nonce()` à votre URL.

`wp_verify_nonce( string $nonce, string|int $action = -1 )` permet de vérifier la validité d'un nonce. On lui passe la valeur du nonce et l'action qu'il est censé authentifier. La fonction renvoie 1 si le nonce est valide et date de moins de 12h, 2 s'il date de moins de 24h, et false s'il n'est pas valide.

Ajustez notre fonction de rappel affichant la metabox comme ceci :

```
// in main plugin file
function wpcookbook_metabox_display( $post, $args ){
    $value = get_post_meta( $post->ID, 'wpcookbook_post_tagline', true ) ?: '';
    wp_nonce_field( 'wpcookbook_metabox_save_' . $post->ID, 'wpcookbook_metabox_nonce' );
    ?>
    <div class="components-base-control">
        ...
    </div>
    <p class="description"><?php echo esc_html( $args['args'][ 'description' ] ); ?></p>
<?php
}
```



Nos champs nonce et referer sont bien ajoutés au formulaire.

Pour rendre l'action unique, j'y ai concaténé l'identifiant de l'article. Maintenant, il faut vérifier la validité du nonce dans la fonction de rappel hookée sur `save_post`.

```
// in main plugin file
function wpcookbook_metabox_save( $post_ID, $post, $update ){

    // Check if nonce is set and valid. If not, just early return.
    if ( ! isset( $_POST['wpcookbook_metabox_nonce'] ) || ! wp_verify_nonce(
        $_POST['wpcookbook_metabox_nonce'], 'wpcookbook_metabox_save_' . $post->ID ) ) {
        return;
    }
    ...
}
```

Si le nonce n'est pas envoyé ou n'est pas vérifié, on ne fait rien, simplement. Vous pouvez tester en modifiant la valeur du nonce dans les outils de développement du navigateur. Modifiez la valeur du champ, mettez à jour l'article puis rafraîchissez la page. Votre valeur restera à son ancienne valeur.

Aussi, avant de traiter notre formulaire, il faut vérifier les droits de l'utilisateur. De manière générale, pour toute action sensible il faut absolument vérifier que l'utilisateur initiant l'action ait bien les droits (capacités) nécessaires pour le faire !

Un nonce va permettre de vérifier qu'une action a bien été effectuée de façon authentique par un utilisateur connecté, mais elle ne permet pas de vérifier qu'il en avait bien le droit !

On a déjà vu comment faire dans le chapitre Comprendre les rôles et capacités de WordPress. Il suffit d'utiliser `current_user_can()`.

```
// in main plugin file
function wpcookbook_metabox_save( $post_ID, $post, $update ){

    // Check if nonce is set and valid. If not, just early return.
    if ( ! isset( $_POST['wpcookbook_metabox_nonce'] ) || ! wp_verify_nonce(
        $_POST['wpcookbook_metabox_nonce'], 'wpcookbook_metabox_save_' . $post->ID ) ) {
        return;
    }

    // Check user has permissions
    if ( ! current_user_can( 'edit_post', $post_ID ) ) {
        return;
    }

    ...
}
```

On vérifie qu'il a le droit d'éditer ce post en particulier, mais on pourrait aussi vérifier `edit_posts` par exemple. S'il ne peut pas, on ne fait rien.

Les données de notre metabox ne sont pas envoyées lors d'une sauvegarde automatique, mais le hook `save_post` est tout de même déclenché. Notre fonction va donc être exécutée et cela peut poser souci car nos valeurs dans `$_POST` ne seront pas présentes. Donc pour plus de contrôle, on ne va rien faire en cas de sauvegarde automatique et forcer notre utilisateur à valider ses choix via le bouton Mettre à jour ou Publier.

```

// in main plugin file
function wpcookbook_metabox_save( $post_ID, $post, $update ){

    // Check if nonce is set and valid. If not, just early return.
    if ( ! isset( $_POST['wpcookbook_metabox_nonce'] ) || ! wp_verify_nonce(
        $_POST['wpcookbook_metabox_nonce'], 'wpcookbook_metabox_save_' . $post->ID ) ) {
        return;
    }

    // Check user has permissions
    if ( ! current_user_can( 'edit_post', $post_ID ) ) {
        return;
    }

    // If autosaving, do nothing
    if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE ) {
        return;
    }

    ...
}

```

Seulement maintenant on peut traiter notre donnée, en vérifiant qu'elle a bien été soumise. Si la valeur soumise est vide, on peut aussi complètement supprimer la metadonnée au lieu de sauvegarder une entrée en base contenant une chaîne vide.

```

// in main plugin file
function wpcookbook_metabox_save( $post_ID, $post, $update ){

    // Check if nonce is set and valid. If not, just early return.
    if ( ! isset( $_POST['wpcookbook_metabox_nonce'] ) || ! wp_verify_nonce(
        $_POST['wpcookbook_metabox_nonce'], 'wpcookbook_metabox_save_' . $post->ID ) ) {
        return;
    }

    if ( ! current_user_can( 'edit_post', $post_ID ) ) {
        return;
    }

    if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE ) {
        return;
    }

    if( isset( $_POST['wpcookbook-post-tagline'] ) ){
        if ( empty( $_POST['wpcookbook-post-tagline'] ) ){
            delete_post_meta( $post_ID, 'wpcookbook_post_tagline' );
        } else {
            update_post_meta( $post_ID, 'wpcookbook_post_tagline', sanitize_text_field(
                $_POST['wpcookbook-post-tagline'] ) );
        }
    }
}

```

## Point intermédiaire

Pour résumer un peu ce que l'on a vu jusqu'à maintenant :

- Il existe une fonctionnalité native de WordPress pour les champs personnalisés. Mais l'utilisateur a le contrôle sur le nom des clés et on ne peut pas y mettre de valeurs complexes comme des tableaux.
- On déclare une metabox avec la fonction `add_meta_box()`, hookée sur `add_meta_boxes`. Attention à indiquer une bonne valeur pour le paramètre `$screen`, pour éviter d'afficher la metabox au mauvais endroit.
- La fonction de rappel associée à la metabox doit afficher les champs `<input>` nécessaires, y compris les champs de sécurité créés par `wp_nonce_field()`. Pas besoin d'élément `<form>` ou de bouton pour soumettre le formulaire.
- Attention à bien utiliser un nom d'action unique !
- Pour traiter nos champs personnalisés, on se hooke sur `save_post` ou `save_post_{post}` où `$post` est le type de contenu sauvegardé.
- On vérifie l'existence et la validité du nonce dans `$_POST` dans notre fonction de rappel. Puis les droits utilisateurs. Puis si on est dans une sauvegarde automatique.
- Ensuite on traite nos champs, sans oublier de les nettoyer ! Pour les sauvegarder ou supprimer, on utilise les fonctions `add_post_meta()`, `update_post_meta()` et `delete_post_meta()`.

## Un autre champ

Ajoutons un autre champ de type checkbox pour s'entraîner. Dans notre fonction de rappel affichant notre metabox, ajoutez :

```
function wpcookbook_metabox_display( $post, $args ){
    $value = get_post_meta( $post->ID, 'wpcookbook_post_tagline', true ) ?: '';
    $checked = get_post_meta( $post->ID, 'wpcookbook_post_checkbox', true );
    wp_nonce_field( 'wpcookbook_metabox_save_' . $post->ID, 'wpcookbook_metabox_nonce' );
?>
<div class="components-base-control">
    ...
</div>
<p class="description"><?php echo esc_html( $args['args']['description']); ?></p>

<div class="components-base-control">
    <div class="components-base-control__field">
        <span class="components-checkbox-control__input-container">
            <style>#wpcookbook-post-checkbox:checked {background-color:#11a0d2; border-color: #11a0d2;}</style>
            <input id="wpcookbook-post-checkbox"
                   name="wpcookbook-post-checkbox"
                   class="components-checkbox-control__input"
                   type="checkbox"
                   <?php checked( $checked ); ?>
            >
            <svg aria-hidden="true" role="img" focusable="false" class="dashicon dashicons-yes components-checkbox-control__checked" xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 20 20">
                <path d="M14.83 4.89L1.34 9.45-5.81 8.38H9.02L5.78 9.67L1.34-1.25 2.57 2.4z"></path>
            </svg>
        </span>
        <label for="wpcookbook-post-checkbox" class="components-checkbox-control__label"><?php esc_html_e( 'Example checkbox', '28-metabox' ); ?></label>
    </div>
</div>
<?php
}
```

Pour coller plus aux styles du nouvel éditeur, j'ai utilisé le même balisage. Il faut ajouter une balise conteneur `<span>` et un `<svg>` à côté de notre checkbox. La classe CSS du label a changé aussi. Par contre, le CSS de l'éditeur n'affecte pas notre checkbox. Or pour que la coche soit visible, il faut que notre checkbox soit bleue quand elle est cochée. D'où ma balise `<style>` supplémentaire. Honnêtement, vous pouvez vous en sortir avec une simple checkbox sans style particulier. Mais c'est moins classe !

Je récupère la valeur de la métadonnée `wpcookbook_post_checkbox` si elle existe et j'utilise la fonction `checked()` pour afficher l'attribut du même nom sur notre checkbox si elle est cochée.

The screenshot shows a WordPress metabox titled "WPCookBook example metabox". Inside, there's a "Tagline" field containing "Test tagline", a note "Type in your awesome tagline!", and a checked "Example checkbox" with a cursor icon pointing at it.

C'est joli, quand même. Pénible à faire mais joli.

Maintenant, il faut sauvegarder la donnée.

```
// in main plugin file
function wpcookbook_metabox_save( $post_ID, $post, $update ){

    // Security checks
    ...

    if( isset( $_POST['wpcookbook-post-checkbox'] ) ){
        update_post_meta( $post_ID, 'wpcookbook_post_checkbox', true );
    } else {
        delete_post_meta( $post_ID, 'wpcookbook_post_checkbox' );
    }
}
```

Si la case n'est pas cochée, rien n'est envoyé. Si elle l'est alors on a une valeur, donc on peut sauvegarder true. Si on n'a rien, on peut tenter d'effacer la métadonnée complètement.

## Améliorons tout ça

On a deux champs fonctionnels dans notre metabox mais si tout est rempli, on crée autant d'entrées en base de données. Comme pour nos réglages, c'est mieux de stocker nos métadonnées dans un tableau. Cela permet de n'avoir qu'une seule entrée, pour grouper des valeurs qui vont sûrement être utilisées au même endroit de toute façon.

```
// in main plugin file
function wpcookbook_metabox_save( $post_ID, $post, $update ){
    // Security checks
    ...

    $values = array();
    if( ! empty( $_POST['wpcookbook-post-tagline'] ) ){
        $values['tagline'] = sanitize_text_field( $_POST['wpcookbook-post-tagline'] );
    }
    if( isset( $_POST['wpcookbook-post-checkbox'] ) ){
        $values['checkbox'] = true;
    }

    if( ! empty( $values ) ){
        update_post_meta( $post->ID, 'wpcookbook_metabox_values', $values );
    } else {
        delete_post_meta( $post->ID, 'wpcookbook_metabox_values' );
    }
}
```

On initialise un tableau vide. Puis si le champ texte n'est pas vide, on ajoute sa valeur dans le tableau. Idem pour la checkbox. Si le tableau se retrouve vide, cela signifie que la checkbox n'a pas été cochée et le champ texte a été laissé vide. Donc on peut supprimer la metadonnée en base. Sinon, on la sauvegarde.

Cela implique aussi quelques ajustements dans la fonction de rappel de la metabox :

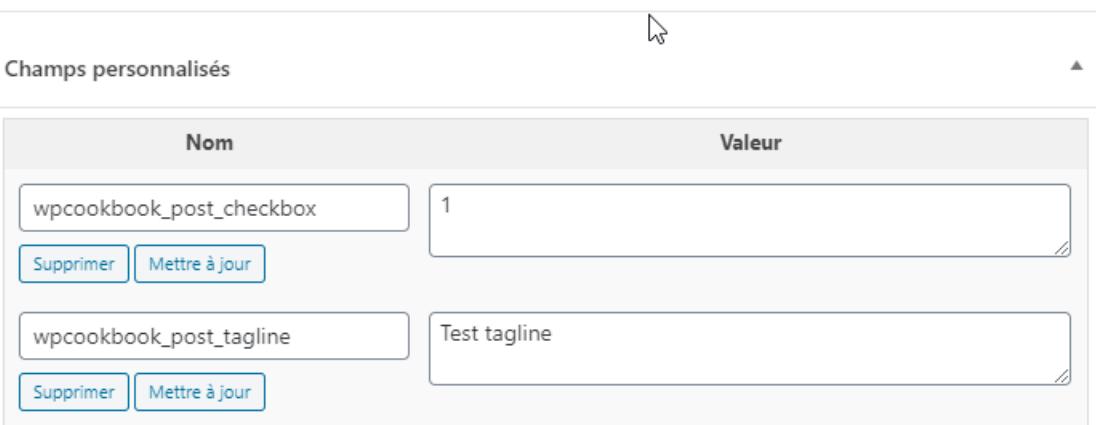
```
function wpcookbook_metabox_display( $post, $args ){
    $values = get_post_meta( $post->ID, 'wpcookbook_metabox_values', true ) ?: array();
    $tagline = ! empty( $values['tagline'] ) ? $values['tagline'] : '';
    $checked = ! empty( $values['checkbox'] );
    wp_nonce_field( 'wpcookbook_metabox_save_' . $post->ID, 'wpcookbook_metabox_nonce' );
?>
    ...
<?php
}
```

Je vous laisse utiliser \$tagline et \$checked dans les attributs des <input>.

Tout fonctionne correctement, et ce qui est magique c'est que vous n'avez pas besoin de sérialiser votre tableau avant de l'insérer en base de données, ni de désérialiser les données quand vous allez les chercher ! Exactement comme pour la Settings API, les fonctions outils de WordPress pour manipuler les metadonnées font tout pour vous !

## Champs publics et champs cachés

Si vous ouvrez l'interface des champs personnalisés natifs de WordPress, vous y trouverez les champs avec lesquels on a travaillé jusqu'à maintenant.



Champs personnalisés

| Nom                      | Valeur       |
|--------------------------|--------------|
| wpcookbook_post_checkbox | 1            |
| wpcookbook_post_tagline  | Test tagline |

Supprimer    Mettre à jour

Supprimer    Mettre à jour

Ajouter un nouveau champ personnalisé :

| Nom              | Valeur |
|------------------|--------|
| — Sélectionner — |        |

Saisissez-en un nouveau

Ajouter un champ personnalisé

Les champs personnalisés peuvent être utilisés pour ajouter des métadonnées supplémentaires à une publication, que vous pouvez ensuite [utiliser dans votre thème](#).

Nos champs sont visibles pour tous les utilisateurs

Donc potentiellement, un utilisateur pourrait modifier ses données ici plutôt que dans la metabox prévue à cet effet. Aussi, si vous utilisez des métadonnées supplémentaires pour lesquelles il n'y a pas d'interface prévue, l'utilisateur y a quand même accès.

Vous remarquerez cependant que notre métadonnée wpcookbook\_metabox\_values n'apparaît pas, simplement parce que l'interface ne peut pas afficher de tableaux mais uniquement des valeurs texte simples.

Si vous souhaitez cacher vos métadonnées des utilisateurs pour limiter les risques de corruption accidentelle, vous pouvez donc soit les ranger dans un tableau comme nous avons fait, ou alors vous pouvez préfixer les clés d'un underscore \_ . C'est tout simple, mais il faut le savoir.

## Utiliser nos métadonnées sur le devant du site

Nous avons vu en détails comment créer une metabox et sauvegarder les valeurs des métadonnées en toute sécurité. Nous avons utilisé nos données sauvegardées pour afficher les valeurs dans le formulaire.

Vous pouvez utiliser les métadonnées comme bon vous semble, et pour l'exemple, on va les utiliser sur le devant du site. Il suffit d'utiliser `get_post_meta()` pour les récupérer.

```
// in main plugin file
add_filter( 'the_title', 'wpcookbook_title_tagline', 10, 2 );
/**
 * Filters the post titles to add in our awesome tagline, if set
 *
 * @param string $title Title of the post
 * @param int    $id    ID of the post
 * @return string $title
 */
function wpcookbook_title_tagline( $title, $id ){
    if( 'post' !== get_post_type( $id ) || is_admin() ){
        return $title;
    }

    if( ! empty( $values = get_post_meta( get_the_ID(), 'wpcookbook_metabox_values', true ) )
&& ! empty( $values['tagline'] ) ){
        $tagline = sprintf( '<span class="wpcookbook-tagline" style="font-size: 60%;"> - 
%$</span>', esc_html( $values['tagline'] ) );
        $title .= $tagline;
    }

    return $title;
}
```

On filtre le titre de nos articles avec cette fonction hookée sur `the_title`. On vérifie que l'on est bien en train de filtrer le titre d'un article avec `get_post_type()` ou que l'on n'est pas dans l'administration, sinon on retourne le titre de suite. Il est très important de passer à `get_post_type()` l'identifiant de la publication passée à notre fonction par le filtre. Sinon, la fonction va vérifier le type de contenu de la globale `$post`, qui aura donc la même valeur quelque soit le titre sur lequel le filtre est en train de passer. Autrement dit, on veut vérifier le type du contenu dont on est en train de filtrer le titre !

Puis on vérifie si on a bien une valeur pour notre tagline et on l'ajoute simplement au titre. Pardonnez mon attribut `style`, mais c'est pour la simplicité de l'exemple.

WPCookBook — Apprenez à développer pour WordPress

[Page](#) [Article](#) [Catégorie](#)

## Hello world! – Awesome tagline

vincent septembre 12, 2019 Un commentaire Modifier



Welcome to WordPress. This is your first post. Edit or delete it, then start writing !

vincent septembre 12, 2019 Uncategorized Modifier

Notre tagline apparait. Nice.

## Pour aller plus loin

Comme pour la Settings API, je vous invite à vous entraîner à créer différents types de champs: radio, select, date, etc. Parcourez aussi les fonctions utilisées pour nettoyer les données, comme la famille des sanitize\_, et pour échapper les données comme la famille des esc\_ ou des wp\_kses\_.

Pour ce qui est des champs fermés dont les valeurs sont prédéfinies comme les boutons radio ou champs <select>, utilisez une fonction ou variable qui contient vos options possibles. Vous pourrez ainsi l'utiliser pour afficher les options mais aussi pour vous assurer que les valeurs des champs à traiter sur save\_post soient bien conformes. Les arguments supplémentaires passés à la fonction de rappel de la metabox par add\_meta\_box() prennent tout leur sens ici.

On n'a exploré que les post\_meta, mais il existe aussi des comment\_meta servant à stocker des données supplémentaires sur les commentaires, et des user\_meta pour les données des utilisateurs. Et tout comme pour les post\_meta, il existe aussi un groupe de fonctions add\_comment\_meta(), update\_comment\_meta(), delete\_comment\_meta(), get\_comment\_meta() qui fonctionnent de la même façon. Je vous laisse deviner le nom de celles concernant les utilisateurs, ok ?

## Ce qu'il faut retenir

- Pour créer une metabox, on utilise `add_meta_box()` hookée sur `add_meta_boxes`.
- On utilise `wp_nonce_field()` pour afficher un champ caché nonce. Attention à bien utiliser un nom d'action unique !
- Pour sauvegarder, on se hooke sur `save_post` ou `save_post_${post}`, où `$post` est le type de contenu sauvegardé.
- On n'oublie pas de vérifier le nonce, les droits et si on est dans une sauvegarde automatique avant de faire quoi que ce soit.
- On nettoie puis on utilise les fonctions `add_post_meta()`, `update_post_meta()` et `delete_post_meta()` pour sauvegarder nos métadonnées.
- Comme pour vos pages de réglages, il peut être judicieux de grouper vos données et de les stocker dans un tableau.
- Les métadonnées dont la clé commence par un underscore \_ n'apparaissent pas dans l'interface par défaut de WordPress, tout comme les métadonnées stockées sous forme de tableau.

# Comment traiter des données de formulaires

Les formulaires sont omni-présents sur le web. Votre administration en est remplie et vous avez sûrement au minimum un formulaire de contact sur votre site. Les APIs de WordPress permettent de simplifier le traitement de vos formulaires dans l'administration, en fournissant une partie du balisage pour vous ainsi que les hooks sur lesquels vous greffer.

Mais si vous codez un formulaire sur le devant du site ou un formulaire personnalisé en administration, vous aurez besoin de savoir comment on traite les données proprement dans WordPress.

Pour ce chapitre, on va créer une extension qui affiche un formulaire sur le devant du site. Ce formulaire permettra aux utilisateurs de soumettre des témoignages. On va y inclure un champ pour le nom, un pour l'URL de l'auteur et un pour le contenu du témoignage. Quelque chose de très simple et courant en somme.

Quand les utilisateurs vont remplir et soumettre leur témoignage, ce dernier sera stocké en base de données sous la forme d'une publication d'un type de contenu personnalisé. La valeur du champ nom pourra servir de titre, le contenu du témoignage sera affecté au contenu de la publication, et l'URL de l'auteur sera enregistrée dans les métadonnées. Libre ensuite à l'administrateur du site de publier, modifier ou supprimer les témoignages.

Ça paraît simple comme ça, mais en fait, il y a du travail !

- il faut créer le formulaire. Pour l'afficher de façon simple, on va utiliser un code court (Chapitre Comment créer un code court)
- il faut créer le type de contenu personnalisé qui va recevoir nos soumissions (Chapitre Comment créer un type de contenu personnalisé)
- il faut créer la metabox pour pouvoir administrer l'URL de l'auteur du témoignage (Chapitre Comment créer une metabox avec des champs personnalisés)
- il faut traiter les données du formulaire. C'est l'objectif global de ce chapitre.

Aussi, on va faire tout ça en intégrant au maximum toutes les bonnes pratiques de sécurité que l'on a vu au fil de ce guide. Par contre, on va se concentrer dans ce chapitre sur le traitement du formulaire. On va aller assez vite sur les autres aspects, car ceux-ci ont été détaillés dans les autres chapitres de ce guide. Donc je vous laisse faire toutes les optimisations et abstractions que vous jugerez utiles pour améliorer les autres aspects de cet exercice, ok ?

Sur ce, au boulot.

## Déclarer notre type de contenu personnalisé

Commencez par créer une nouvelle extension, et déclarez notre nouveau type de contenu.

```
// in main plugin file
add_action( 'init', 'wpcookbook_register_post_type' );
/**
 * Registers our Testimonials custom post type
 */
function wpcookbook_register_post_type(){
    register_post_type( 'testimonial', array(
        'label'              => _x( 'Testimonials', 'post type general name', '29-forms' ),
        'show_ui'            => true,
        'register_meta_box_cb' => 'wpcookbook_testimonial_metabox',
        'supports'           => array( 'title', 'editor', 'custom-fields' ),
    ) );
}
```

Si vous avez lu le chapitre Comment créer un type de contenu personnalisé, il n'y a rien de surprenant. On crée un type de contenu `testimonial` et on lui passe un tableau d'arguments avec le minimum vital. La valeur par défaut pour `public` est `false`, et c'est très bien car je ne souhaite pas que les témoignages soient accessibles sur le devant du site avec une URL du type <https://wpcookbook.local/testimonial/my-testimonial>. Par contre, je souhaite pouvoir les administrer. Donc on passe `show_ui` sur `true`.

On peut passer une fonction de rappel à exécuter pour enregistrer directement les metaboxes associées à ce type de contenu. C'est très pratique car cela évite de hooker la fonction sur `add_meta_boxes`, comme on a fait dans le chapitre Comment créer une metabox avec des champs personnalisés. Ce paramètre a été mentionné dans le chapitre Comment créer un type de contenu personnalisé, mais n'a pas été utilisé. Voici une bonne occasion !

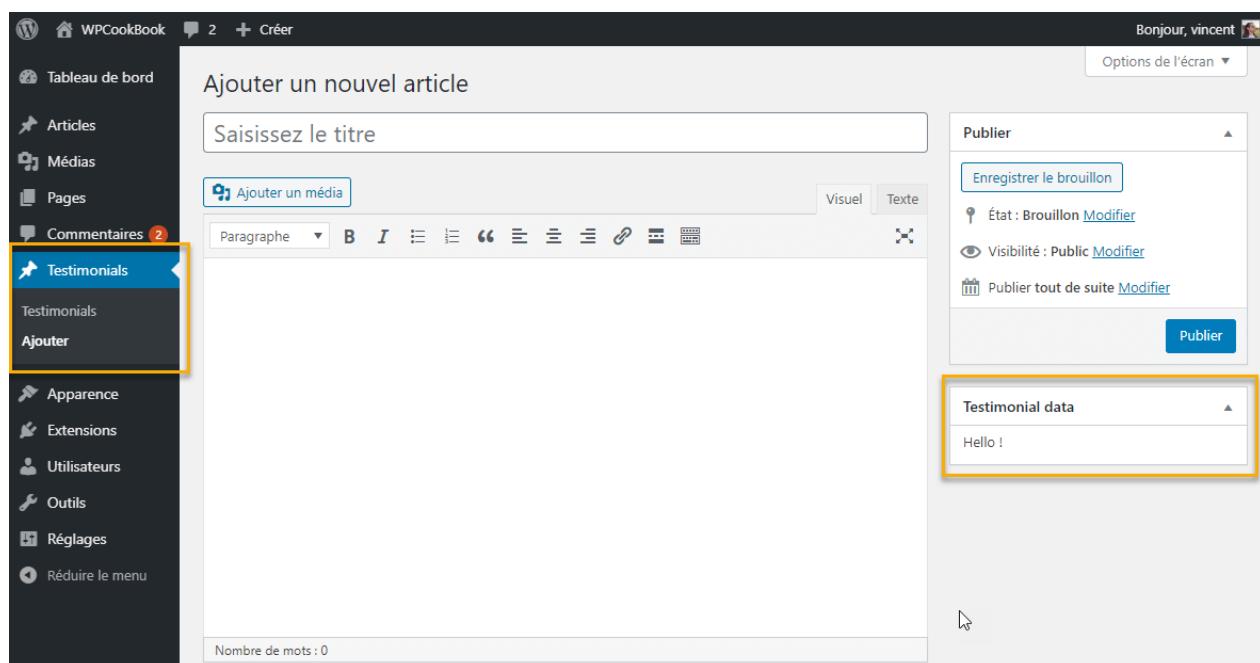
Pour le paramètre `supports`, on aurait très bien pu laisser la valeur par défaut (`array( 'title', 'editor' )`), mais pour développer je trouve qu'avoir un deuxième endroit où pouvoir inspecter la valeur d'une métadonnée est utile, puisqu'il m'est déjà arrivé de faire des fautes de frappe sur les noms des champs. On pourra supprimer cette ligne plus tard.

## Déclarer notre metabox

Déclarons rapidement la fonction de rappel enregistrant la métabox.

```
// in main plugin file
/**
 * Callback function to register our metabox
 */
function wpcookbook_testimonial_metabox(){
    add_meta_box(
        'wpcookbook-testimonial-metabox',           // ID
        __( 'Testimonial data', '29-forms' ),         // Title
        'wpcookbook_testimonial_metabox_display',     // Display callback
        'testimonial',                                // Screen or post type
        'side',                                      // Context
    );
}

/**
 * Displays our testimonial metabox content
 *
 * @param WP_Post $post The post object being edited
 * @param array   $args The parameters passed to add_meta_box() call
 */
function wpcookbook_testimonial_metabox_display( $post, $args ){
    echo 'Hello !';
}
```



Notre type de contenu est en place.

Notre type de contenu est en place. Vous remarquerez que ce dernier utilise l'éditeur classique de WordPress. Cela ne pose aucun souci, car le contenu de nos témoignages viendra d'une simple zone de texte et je pense qu'il n'y aura pas besoin de modifier le contenu ni d'y ajouter un bloc complexe disponible dans le nouvel éditeur.

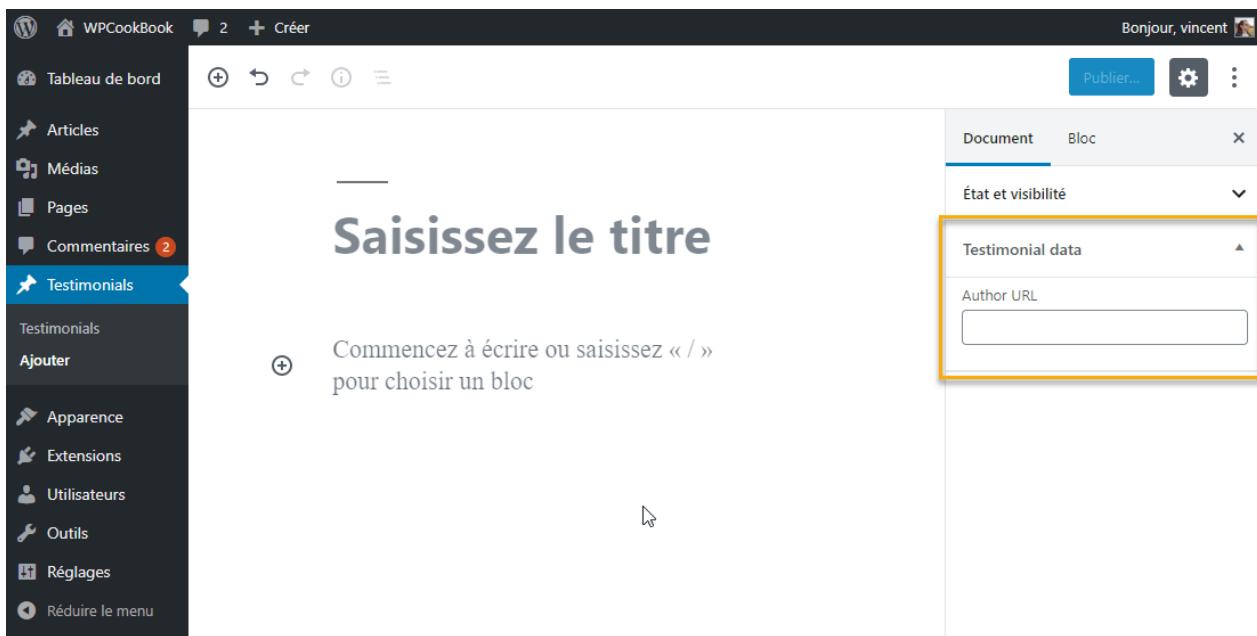
Concentrons-nous rapidement sur la metabox.

```
// in main plugin file
function wpcookbook_testimonial_metabox_display( $post, $args ){
    $url = get_post_meta( $post->ID, 'wpcookbook_testimonial_url', true ) ?: '';
    wp_nonce_field( 'wpcookbook_testimonial_metabox_save_' . $post->ID,
    'wpcookbook_testimonial_metabox_nonce' );
    ?>
    <div class="components-base-control">
        <div class="components-base-control__field">
            <label for="wpcookbook-testimonial-url" class="components-base-
control__label"><?php esc_html_e( 'Author URL', '29-forms' ); ?></label><br />
            <input id="wpcookbook-testimonial-url" name="wpcookbook-testimonial-url"
type="url" class="components-text-control__input widefat" value="<?php echo esc_attr( esc_url(
$url ) ); ?>" />
        </div>
    </div>
    <?php
}
```

Encore une fois, si vous avez lu attentivement le chapitre Comment créer une metabox avec des champs personnalisés, vous n'aurez aucun mal à comprendre ce qu'il se passe ici.

On récupère la metadonnée `wpcookbook_testimonial_url` associée à la publication. Cette metadonnée contiendra l'URL que l'auteur a entrée dans le formulaire. Puis on affiche les champs de sécurité nécessaires à l'aide de `wp_nonce_field()` et notre champ de type `url` accompagné de son label.

Si le balisage paraît complexe c'est parce que j'ai (honteusement) copié-collé le balisage utilisé dans le chapitre Comment créer une metabox avec des champs personnalisés. C'est en réalité le balisage utilisé par le nouvel éditeur. Cela ne change absolument rien dans notre cas car les classes CSS ne sont pas utilisées dans l'éditeur classique, mais au moins si vous voulez utiliser le nouvel éditeur pour ce type de contenu, vous n'aurez pas à refaire son balisage ! Il faut voir le bon côté des choses !



La metabox s'affiche sans souci dans le nouvel éditeur aussi !

## Traitement des données de la metabox

Maintenant, il faut traiter les données de la metabox en se hookant sur `save_post_testimonial`, comme expliqué dans le chapitre précédent.

```
add_action( 'save_post_testimonial', 'wpcookbook_metabox_save', 10, 3 );
/**
 * Saves our metabox fields
 *
 * @param int      $post_ID  Post ID.
 * @param WP_Post  $post     Post object.
 * @param bool     $update   Whether this is an existing post being updated or not.
 */
function wpcookbook_metabox_save( $post_ID, $post, $update ) {
    // Check if nonce is set and valid. If not, just early return.
    if ( ! isset( $_POST['wpcookbook_testimonial_metabox_nonce'] ) || ! wp_verify_nonce(
        $_POST['wpcookbook_testimonial_metabox_nonce'], 'wpcookbook_testimonial_metabox_save_'
        . $post_ID ) ) {
        return;
    }
    if ( ! current_user_can( 'edit_post', $post_ID ) ) {
        return;
    }
    if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE ) {
        return;
    }

    if( ! empty ( $_POST['wpcookbook-testimonial-url'] ) ){
        update_post_meta( $post_ID, 'wpcookbook_testimonial_url', esc_url_raw(
            $_POST['wpcookbook-testimonial-url'] ) );
    } else {
        delete_post_meta( $post_ID, 'wpcookbook_testimonial_url' );
    }
}
```

On vérifie l'existence et la validité du nonce dans les données soumises, puis les droits utilisateurs, puis on s'assure que l'on est pas dans le cadre d'une sauvegarde automatique pour pouvoir enfin traiter nos données. Si la valeur du champ `wpcookbook-testimonial-url` n'est pas vide, on la sauvegarde en remplaçant la précédente à l'aide de `update_post_meta()`, sinon on tente de la supprimer. Bien sûr, on utilise `esc_url_raw()` pour échapper notre url avant de l'insérer en base de données.

WPCookBook

Créer

Bonjour, vincent

Tableau de bord

Articles

Médias

Pages

Commentaires 2

**Testimonials**

Testimonials

Ajouter

Apparence

Extensions

Utilisateurs

Outils

Réglages

Réduire le menu

Modifier l'article Ajouter

Publication mise à jour.

First testimonial

Ajouter un média Visuel Texte

Paragraphe B I Éléments “ Éléments ¶ Éléments

This is awesome !

Nombre de mots : 3 Dernière modification par vincent, le 29 novembre 2019 à 14 h 10 min

Testimonial data

Author URL  
https://vincentdubroeucq.com/wpcool

Champs personnalisés

| Nom                        | Valeur                                  |
|----------------------------|---|
| wpcookbook_testimonial_url | https://vincentdubroeucq.com/wpcookbook |

Supprimer Mettre à jour

Tout fonctionne correctement. Notre donnée peut être sauvegardée.

## Affichons notre formulaire

Pour afficher le formulaire, on va créer un code court simple sans aucun paramètre.

```
// in main plugin file
add_shortcode( 'wpcookbook_testimonial_form', 'wpcookbook_testimonial_form' );
/**
 * Displays our testimonial form.
 *
 * @param array $atts      Array of shortcode attributes
 * @param string $content Content between shortcode tags.
 * @param string $tag      Shortcode tag itself
 * @return string          String of html content
 */
function wpcookbook_testimonial_form( $atts, $content, $tag ){
    ob_start();
    ?>
        <form id="wpcookbook-testimonial-form" class="wpcookbook-testimonial-form"
method="POST">
            <p class="wpcookbook-form-field">
                <label for="name"><?php esc_html_e( 'Your name', '29-forms' ); ?> <span
class="required"><?php esc_html_e( '(Required)', '29-forms' ); ?></span></label>
                    <input type="text" id="name" name="name" required />
            </p>
            <p class="wpcookbook-form-field">
                <label for="url"><?php esc_html_e( 'Your website URL', '29-forms' ); ?>
</label>
                    <input type="url" id="url" name="url" />
            </p>
            <p class="wpcookbook-form-field">
                <label for="content"><?php esc_html_e( 'Your testimonial', '29-forms' ); ?>
<span class="required"><?php esc_html_e( '(Required)', '29-forms' ); ?></span></label>
                    <textarea id="content" name="content" placeholder="<?php esc_attr_e( 'Say
something nice ! ', '29-forms' ); ?>" rows="4" required ></textarea>
            </p>
                    <input type="submit" value="<?php esc_attr_e( 'Send testimonial', '29-forms' ); ?>" />
        </form>
        <?php wp_enqueue_style( 'wpcookbook-testimonial', plugins_url( 'assets/css/style.css',
__FILE__ ) ); ?>
    <?php
        return ob_get_clean();
    }
}
```

On déclare un code court `wpcookbook_testimonial_form`. Dans la fonction de rappel, j'utilise un tampon avec `ob_start()` qui va me permettre de fermer ma balise PHP et d'insérer plus facilement le code HTML du formulaire. En fin de fonction, il ne faut pas oublier de retourner la chaîne contenue dans le tampon avec `ob_get_clean()`.

Dans le formulaire, on a simplement trois paragraphes avec trois champs simples : un champ texte requis, un champ URL optionnel et une zone de texte requise. Je charge aussi une feuille de style placée dans le dossier `assets/css/` de l'extension. Elle contient juste de quoi mettre les éléments `<label>` en `display: block;` pour l'instant.

Créez une page avec ce code court, et rendez-vous sur le devant de votre site.

# Testimonial form

Your name (Required)



Your website URL

Your testimonial (Required)

Say something nice !

**Send testimonial**

Notre formulaire est prêt.

## Comment traiter notre formulaire

Ok, les choses sérieuses commencent maintenant.

Pour le moment, si vous remplissez et soumettez le formulaire vous obtiendrez une erreur, car celui-ci n'envoie ses données nulle part. Notez aussi que si vous ne remplissez pas le champ nom ou la zone de texte, vous aurez une erreur de validation du formulaire et ne pourrez le soumettre. Même chose si vous entrez une URL non valide dans le champ Your website URL. C'est l'avantage quand on utilise les types de champ standards HTML5 avec les bons attributs.

Dans une application PHP typique, vous allez envoyer les données du formulaire vers un script présent sur votre serveur, qui va traiter ces données.

Mais pour une extension WordPress, on procède un peu différemment. On pourrait envoyer les données vers un script autonome contenu dans votre extension, mais le souci est que si on fait de cette manière nous n'avons accès ni aux fonctions de WordPress, ni aux fonctions de notre extension. On serait complètement hors de WordPress !

On pourrait les inclure manuellement mais encore faudrait-il être sûr de leur place dans l'arborescence des fichiers et puisque tous les dossiers de WordPress sont personnalisables ou déplaçables, l'exercice est plus que périlleux !

L'idée est donc d'envoyer nos données vers un script qui nous permettrait de rester dans l'environnement WordPress et d'avoir ainsi accès à toutes les fonctions de WordPress et de nos extensions.

## admin-post.php

Ce script existe, et c'est `admin-post.php`. Il est situé dans le dossier `wp-admin/` et son rôle est de charger l'environnement de WordPress pour pouvoir y traiter des données envoyées avec des requêtes POST ou GET. Aussi, son nom et son emplacement indiquent qu'il fait partie de l'administration de WordPress et il va charger tout ce qui est typiquement disponible dans l'administration de WordPress.

En fait, si vous examinez de plus près le fichier `admin-post.php`, vous verrez qu'il ne fait pas grand chose. Il charge WordPress et les fonctions disponibles normalement dans l'administration, déclenche le hook `admin_init`, puis expose différents hooks selon si l'utilisateur est connecté ou non, et la présence d'une donnée nommée `action` dans les paramètres de la requête. On dispose donc des hooks suivants :

- `admin_post` qui sera déclenché pour toute requête sans valeur pour `action` venant d'un utilisateur connecté.
- `admin_post_{$action}` qui sera déclenché pour toute requête contenant un paramètre `action` venant d'un utilisateur connecté.
- `admin_post_nopriv` qui sera déclenché pour toute requête sans valeur pour `action` venant d'un utilisateur non connecté.
- `admin_post_nopriv_{$action}` qui sera déclenché pour toute requête contenant un paramètre `action` venant d'un utilisateur non connecté.

Le formulaire à traiter est disponible pour les utilisateurs connectés ou non. Il faut donc hooker notre fonction de rappel sur les hooks `admin_post`, mais aussi sur `admin_post_nopriv`. Mais pour éviter de déclencher notre fonction à chaque fois que le hook `admin_post` est déclenché, on va simplement ajouter un champ caché `action` unique dans notre formulaire, et hooker sur `admin_post_{$action}` et `admin_post_nopriv_{$action}` pour ne déclencher notre fonction que quand c'est pertinent.

```
// in main plugin file
function wpcookbook_testimonial_form( $atts, $content, $tag ){
    ...
    <form id="wpcookbook-testimonial-form" class="wpcookbook-testimonial-form"
method="POST" action="php echo esc_attr( esc_url( admin_url( 'admin-post.php' ) ) ); ?&gt;"&gt;
        &lt;input type="hidden" name="action" value="wpcookbook_testimonial_form_submit" /&gt;
        ...
    &lt;/form&gt;
    ...
}</pre
```

On a utilisé la fonction `admin_url()` pour récupérer l'URL vers le fichier `admin-post.php`, puis on l'affiche dans l'attribut `action` du formulaire en l'échappant proprement. Puis on ajoute un champ caché ayant pour attribut `name="action"` et pour valeur un nom unique.

On peut maintenant créer notre fonction de rappel qui va traiter le formulaire.

```
//in main plugin file
add_action( 'admin_post_wpcookbook_testimonial_form_submit',
'wpcookbook_testimonial_form_handler' );
add_action( 'admin_post_nopriv_wpcookbook_testimonial_form_submit',
'wpcookbook_testimonial_form_handler' );
/**
 * Handles our testimonial form data
 */
function wpcookbook_testimonial_form_handler(){
    ob_start();
    echo '<pre>';
    print_r( $_POST );
    echo '</pre>';
    wp_die( ob_get_clean());
}
```

Comme expliqué précédemment, on la hooke sur `admin_post_${action}` et `admin_post_nopriv_${action}` car il faut traiter le formulaire que l'utilisateur soit connecté ou non. La fonction ne fait rien de spécial pour l'instant : elle affiche juste le contenu de la variable `$_POST`, contenant donc les données de notre formulaire, dans un `wp_die()`. Un peu sale, mais cela permet de vérifier que tout est bien hooké correctement. Remplissez et soumettez le formulaire pour tester.

```
Array
(
    [action] => wpcookbook_testimonial_form_submit
    [name] => Vincent Dubroeucq
    [url] => https://vincentdubroeucq.com
    [content] => This is nice !
)
```

Tout est hooké correctement.

Si vous changez la valeur du champ `action` dans les outils de développement du navigateur, notre hook n'est pas déclenché et vous aurez une page blanche, car notre fonction de rappel se déclenche uniquement pour une valeur de `action` bien précise. Aussi, notre fonction est bien déclenchée, que l'on soit connecté ou non.

## Traitons les données de notre formulaire

Tout est en place. Nos données de formulaire sont disponibles ainsi que toutes les fonctions de WordPress. Y'a plus qu'à.

On aimerait sauvegarder notre entrée dans une publication du type `testimonial`. Le champ nom sera le titre, le contenu du champ zone de texte sera le contenu de notre publication et l'URL sera sauvegardée dans une metadonnée.

Pour insérer une publication en base de données, WordPress fournit une fonction super pratique : `wp_insert_post()`.

`wp_insert_post( array $postarr, bool $wp_error = false )` prends deux paramètres : un tableau d'arguments et un booléen indiquant s'il faut renvoyer une `WP_Error` en cas de souci ou simplement 0. La fonction renvoie l'identifiant de la publication nouvellement créée, ou alors 0 ou une `WP_Error` en cas d'échec selon le deuxième paramètre.

On ne va pas passer en revue tous les arguments acceptés car ce n'est pas vraiment l'objet du chapitre. Je vous laisse lire [la documentation de la fonction](#). Dans notre fonction de rappel, on va donc faire ceci :

```
// in main plugin file
function wpcookbook_testimonial_form_handler(){

    // Prepare arguments to create the post
    $args = array(
        'post_type'      => 'testimonial',
        'post_title'     => sanitize_text_field( $_POST['name'] ),
        'post_content'   => wp_kses_post( $_POST['content'] ),
    );
    if( ! empty( $_POST['url'] ) ){
        $args['meta_input'] = array(
            'wpcookbook_testimonial_url' => esc_url_raw( $_POST['url'] )
        );
    }
    $testimonial_id = wp_insert_post( $args, true );
}
```

On prépare d'abord les arguments pour `wp_insert_post()`. On veut insérer en base de données une publication de type `testimonial`, en lui donnant un titre, un contenu et une metadonnée dans un tableau `meta_input`.

On va utiliser `sanitize_text_field()` pour nettoyer notre champ nom et `wp_kses_post()` pour notre champ zone de texte sachant que la fonction autorise un peu de code HTML simple (celui autorisé dans l'éditeur). On aurait aussi pu l'utiliser pour le champ nom.

Aussi, on vérifie qu'une URL a bien été soumise avant de l'ajouter aux arguments de `wp_insert_post()` pour d'éviter d'ajouter une metadonnée vide, et on l'échappe à l'aide de la fonction `esc_url_raw()` qui échappe et nettoie une URL pour l'insérer en base de donnée. La fonction `wp_insert_post()` va s'occuper toute seule de faire les `update_post_meta()` nécessaires.

Si vous testez, vous aurez une joli page blanche sur le devant du site. Par contre, si vous retournez ensuite

dans l'administration de WordPress, vous verrez que votre publication a bien été ajoutée en brouillon !

The screenshot shows the WordPress admin interface. The left sidebar is titled 'Tableau de bord' and includes links for Articles, Médias, Pages, Commentaires (with 2 notifications), Testimonials (selected), Ajouter, Apparence, Extensions, Utilisateurs, Outils, Réglages, and Réduire le menu. The main content area is titled 'Testimonials' with a blue 'Ajouter' button. It shows a list of items under 'Tous (1) | Brouillon (1)'. There is one item: 'Vincent Dubroeucq — Brouillon'. The item details show 'Titre' and 'Dernière modification il y a 8 secondes'. Below the list are buttons for 'Actions groupées' and 'Appliquer'. At the bottom of the page, there is a message: 'Notre témoignage est bien sauvegardé. Cool.'

Notre témoignage est bien sauvegardé. Cool.

## Redirection

La page blanche sur le devant du site est due au fait que le fichier `admin-post.php` ne fait rien, si ce n'est exposer des hooks. De ce fait c'est à notre fonction d'effectuer la redirection qui convient après avoir fait son travail.

Nous souhaitons rediriger vers la page du formulaire, donc nous avons besoin de son URL. Dans notre fonction de rappel traitant nos données, on peut lire le référent de la requête avec la fonction `wp_get_referer()` qui va lire `$_REQUEST['_wp_http_referer']` ou `$_SERVER['REQUEST_URI']`. On peut aussi si l'on souhaite afficher un champ référent dans le formulaire avec la fonction `wp_referer_field()`.

```
// in main plugin file
function wpcookbook_testimonial_form( $atts, $content, $tag ){
    ...
    <form id="wpcookbook-testimonial-form" class="wpcookbook-testimonial-form"
method="POST" action="<?php echo esc_attr( esc_url( admin_url( 'admin-post.php' ) ) ); ?>">
        <input type="hidden" name="action" value="wpcookbook_testimonial_form_submit" />
        <?php wp_referer_field(); ?>
    ...
    </form>
    ...
}
```

On aura alors accès à `$_POST['_wp_http_referer']` qu'on peut lire avec la fonction `wp_get_referer()` pour effectuer notre redirection.

```
// in main plugin file
function wpcookbook_testimonial_form_handler(){
    // Prepare arguments to create the post
    ...

    $referer = wp_get_referer() ?: home_url();
    wp_safe_redirect( $referer );
    exit;
}
```

On récupère le référent sinon on redirige vers l'accueil du site. `wp_safe_redirect()` va nettoyer notre URL et s'assurer qu'elle soit bien locale, c'est-à-dire interne au site.

`wp_safe_redirect( string $location, int $status = 302, string $x_redirect_by = 'WordPress' )` est une fonction très pratique qui permet d'effectuer une redirection locale en toute sécurité, car l'URL est validée et nettoyée au préalable. Vous pouvez aussi passer un code HTTP différent et un header X-Redirect-By si vous le souhaitez. Par contre, il ne faut pas oublier d'arrêter l'exécution du script avec un `die()` ou `exit` juste après.

## Gestion des spams et sécurité

Ok, cela fonctionne et on est bien redirigé vers le formulaire après soumission. Mais on n'a absolument rien fait en ce qui concerne la sécurité, la validation de nos champs, ni le traitement des éventuelles erreurs.

On doit vérifier que toutes les données souhaitées sont présentes et fournir un retour à l'utilisateur quant au succès ou non de sa demande.

Par contre, en ce qui concerne la sécurité, ici un nonce serait malheureusement inutile. Un nonce est associé à un identifiant utilisateur. Or dans notre cas le formulaire est disponible pour les utilisateurs connectés et non-connectés. Si tout le monde a accès au formulaire, quelque soient leurs droits, pourquoi chercher à identifier l'utilisateur et vérifier ses droits ? Aussi, tous les utilisateurs non-connectés auront le même nonce. Si le jeton de sécurité est le même pour tous, il est inutile.

Par contre, si le formulaire était différent selon l'utilisateur connecté ou non, alors le traitement dans `admin-post.php` serait différent, et on aurait besoin de déterminer l'identité de cet utilisateur pour traiter le formulaire de façon appropriée. Là, le nonce prendrait tout son sens.

Dans notre exemple, le formulaire sur le devant du site et son traitement sont les mêmes pour tous. Donc on ne mets pas de nonce. Mais le souci, c'est qu'on va très probablement être sujet aux spams. On peut mettre un captcha (recommandé) ou l'anti-spam du pauvre : un champ de type honeypot.

```

function wpcookbook_testimonial_form( $atts, $content, $tag ){
    ob_start();
    ?>
    ...
    <form id="wpcookbook-testimonial-form" class="wpcookbook-testimonial-form"
method="POST" action="<?php echo esc_attr( esc_url( admin_url( 'admin-post.php' ) ) ); ?>">
        <input type="hidden" name="action" value="wpcookbook_testimonial_form_submit" />
        <input type="checkbox" id="cancel-submission" name="cancel-submission"
class="screen-reader-text" tabindex="-1" autocomplete="off" aria-hidden="true" />
        <?php wp_referer_field(); ?>
        ...
    </form>

    <?php
    return ob_get_clean();
}

```

L'idée est simple. On ajoute simplement un champ inutile, que l'on va rendre inaccessible aux lecteurs d'écran (`aria-hidden`), inaccessible au clavier (`tabindex=-1`) et inaccessible pour vos petits yeux (`class="screen-reader-text"`), pour éviter qu'un humain puisse le remplir. La classe `screen-reader-text` du thème sert à cacher le champ visuellement.

J'ai choisi une checkbox mais on peut utiliser un champ texte ou autre. Si les bots parcourant votre page remplissent le formulaire, il y a de fortes chances pour qu'ils remplissent ce champ. Donc si la checkbox est cochée, cela veut dire que le formulaire a très probablement été rempli par un bot ! Ce n'est pas infaillible, mais ça aide. Il est tout de même recommandé d'utiliser un vrai captcha.

```

// in main plugin file
function wpcookbook_testimonial_form_handler(){
    $referer = wp_get_referer() ?: home_url();

    if( isset( $_POST['cancel-submission'] ) ){
        $referer = add_query_arg( 'message', 'unauthorized', $referer );
        wp_safe_redirect( $referer );
        exit;
    }
    ...
}

```

Ici, si la case est cochée, on redirige vers le référent en ajoutant une chaîne de requête ?`message=unauthorized` sur l'URL à l'aide de la fonction `add_query_arg()`. On peut aussi faire un simple `wp_die()` et afficher un message d'erreur. Comme vous le désirez.

`add_query_arg( string $key, string $value, string $url )` permet d'ajouter des paramètres à une URL. Vous pouvez lui passer soit une clé, une valeur et l'url sur laquelle ajouter les paramètres ou alors vous pouvez passer les clés-valeurs sous forme d'un tableau.

Remarquez qu'on a aussi extrait la ligne récupérant le référent, car on va avoir besoin de la variable à plusieurs reprises. Donc on l'affecte de suite.

De la même façon, on peut ajouter un paramètre d'URL pour notre redirection en fin de traitement, selon le succès ou l'échec de l'insertion en base de données.

```
// in main plugin file
function wpcookbook_testimonial_form_handler(){
    ...
    // Prepare arguments to create the post
    ...
    $testimonial_id = wp_insert_post( $args, true );

    $message_key = ! is_wp_error( $testimonial_id ) ? 'success' : 'failed';
    $referer = add_query_arg( 'message', $message_key, $referer );
    wp_safe_redirect( $referer );
    exit;
}
```

\$testimonial\_id va contenir soit un identifiant numérique soit une WP\_Error. On vérifie donc que l'insertion en base de données s'est bien déroulée, puis on ajoute un paramètre d'URL adapté selon le cas au référent juste avant notre redirection.

## Feedback sur le devant du site

Il faut maintenant utiliser ces paramètres d'URL sur le devant du site pour afficher des notices. On va créer quelques fonctions pour ce faire.

```
// in main plugin file
/**
 * Returns the list of available notices for the form
 *
 * @return array Array of message-key => message-notice
 */
function wpcookbook_get_form_notices(){
    return apply_filters( 'wpcookbook_form_notices', array(
        'unauthorized' => __( 'You are not allowed to do this.', '29-forms' ),
        'failed'        => __( 'Something went wrong. Please try again.', '29-forms' ),
        'success'       => __( 'Your testimonial has been successfully submitted!', '29-
forms' ),
    ) );
}

/**
 * Displays notices above the form, depending on query variables.
 */
function wpcookbook_testimonial_form_notice(){
    if ( empty( $_GET['message'] ) ){
        return ;
    }
    $messages      = wpcookbook_get_form_notices();
    $message_key   = $_GET['message'];
    $message_text  = array_key_exists( $message_key , $messages ) ? $messages[ $message_key ] :
    '';
    $class         = 'success' === $message_key ? 'wpcookbook-testimonial-notice-success' :
    'wpcookbook-testimonial-notice-error';
    $format        = '<div class="wpcookbook-testimonial-notice %s"><p>%s</p></div>';

    if( ! empty( $text ) ){
        $html = sprintf( $format, $class, $message_text );
        $html = apply_filters( 'wpcookbook_form_notice_html', $html, $format, $class,
$message_text );
        echo wp_kses_post( $html );
    }
}
```

La première fonction `wpcookbook_get_form_notices()` va simplement renvoyer un tableau associatif filtrable des messages disponibles, avec les clés correspondant aux valeurs que l'on va utiliser pour le paramètre `$_GET['message']` qui est ajouté lors du traitement.

La deuxième fonction `wpcookbook_testimonial_form_notice()` vérifie que le paramètre d'URL `message` est non-vide et affiche une balise `<div>` avec une classe CSS et le texte correspondant, en vérifiant si la valeur du paramètre est valide (c'est à dire si elle fait partie de notre whitelist de messages retournée par `wpcookbook_get_form_notices()`).

Maintenant, utilisons ces fonctions dans notre template de formulaire.

```
function wpcookbook_testimonial_form( $atts, $content, $tag ){
    ob_start();
    ?>
        <div id="wpcookbook_testimonial_notice_container" role="alert"><?php
        wpcookbook_testimonial_form_notice(); ?></div>
        <form id="wpcookbook-testimonial-form" class="wpcookbook-testimonial-form"
        method="POST" action="<?php echo esc_attr( esc_url( admin_url( 'admin-post.php' ) ) ); ?>">
            ...
        </form>
    ...
}
```

On utilise simplement le template tag que l'on a créé au dessus de notre formulaire. J'ai ajouté des styles dans `style.css` pour que ce soit plus joli et lisible.

Aussi, vous remarquerez que l'on a placé notre fonction dans un conteneur avec pour attribut `role="alert"`. Cela va améliorer l'accessibilité du formulaire quand nous allons plus tard y ajouter les notices de façon dynamiques en JavaScript, car les lecteurs d'écran vont alerter l'utilisateur si le contenu de ce conteneur change.

---

## Testimonial form

Your testimonial has been successfully submitted !

Your name (Required)

Vincent Dubroeucq



Your website URL

<https://vincentdubroe>

Nous avons du feedback après soumission. C'est bien mieux !

## Validation des champs

Maintenant, validons les autres champs. Ce que l'on peut faire, c'est utiliser la classe `WP_Error` pour gérer nos codes et messages d'erreur. Après la vérification des champs, si on a une erreur quelconque, on redirige vers la page du formulaire juste avant le `wp_insert_post()`.

```
// in main plugin file
function wpcookbook_testimonial_form_handler(){

    $referer = wp_get_referer() ?: home_url();
    $messages = wpcookbook_get_form_notices();
    $error = new WP_Error();

    // Check fields
    if( isset( $_POST['cancel-submission'] ) ){
        $error->add( 'unauthorized', esc_html( $messages['unauthorized'] ) );
        $referer = add_query_arg( 'message', 'unauthorized', $referer );
    }
    if( empty( $_POST['name'] ) ){
        $error->add( 'missing-name', esc_html( $messages['missing-name'] ) );
        $referer = add_query_arg( 'message', 'missing-name', $referer );
    }
    if( empty( $_POST['content'] ) ){
        $error->add( 'missing-content', esc_html( $messages['missing-content'] ) );
        $referer = add_query_arg( 'message', 'missing-content', $referer );
    }
    if( $error->has_errors() ){
        wp_safe_redirect( $referer );
        exit;
    }
    ...
}
```

On vérifie si chaque champ requis est bien là. Sinon, on ajoute une nouvelle erreur dans notre `WP_Error` avec le code et le message correspondant (on a récupéré les messages possibles au préalable dans `$messages`), puis on ajoute à notre URL de redirection le paramètre qui permettra d'afficher le message sur le devant du site. Je vous laisse ajuster aussi la fonction `wpcookbook_get_form_notices()` pour ajouter nos nouveaux messages. Avant notre `wp_insert_post()`, on redirige vers le formulaire s'il y a des erreurs.

Vous remarquerez que l'on remplit une `WP_Error` mais qu'au final, on ne l'utilise pas, puisqu'on fait une redirection. Cela dit, la `WP_Error` va se révéler bien pratique quand on améliorera notre formulaire plus tard en le traitant via JavaScript. Soyez patient et faites-moi confiance !

# Testimonial form

It looks like the name field is missing.

Your name (Required)

Your website URL



En cas d'erreur, rien n'est sauvegardé !

Attention, ce n'est pas parce que le champ dispose d'un attribut required qu'il sera forcément présent dans \$\_POST quand vous traiterez le formulaire ! Comment croyez-vous que j'ai effectué ce screenshot ? J'ai supprimé l'attribut required avec l'inspecteur ! Facile !

De manière générale, le backend n'a aucune connaissance de ce qu'il se passe sur le devant du site ! Donc il faut absolument ne rien présupposer et vérifier toutes les informations dont on dispose en backend.

## Une pause, s'il vous plaît !

C'était du boulot tout ça, non ?

Pour résumer un peu :

- On a créé un type de contenu 'testimonial' pour stocker nos témoignages.
- On a déclaré une metabox et sa procédure de sauvegarde sécurisée pour le champ URL.
- On a déclaré un code court pour afficher le formulaire.
- On utilise admin-post.php pour traiter un formulaire en POST.
  - admin-post.php ne fait rien, si ce n'est charger WordPress et exposer des hooks.
  - admin\_url() permet de passer l'URL vers le fichier admin-post.php pour l'utiliser dans l'attribut action du formulaire.
  - wp\_referer\_field() permet d'ajouter un champ référent au formulaire.
  - Utiliser une nonce est inutile dans notre cas. Les utilisateurs non-connectés auront tous le même nonce et les utilisateurs connectés n'ont aucun intérêt à l'être : le formulaire et son traitement sont les mêmes pour tous.
  - On peut utiliser un champ piège de type honeypot pour réduire le spam.
  - On vérifie que les champs attendus soient là. La validation HTML n'est pas suffisante !
  - On nettoie les valeurs récupérées avant de les utiliser.
  - admin\_post.php ne fait rien tout seul ! Il faut donc faire les redirections manuellement avec

`wp_safe_redirect()` et ne pas oublier de terminer le script avec `exit`.

- On a utilisé `add_query_arg()` pour ajouter des paramètres à l'URL de redirection, et on les récupère sur le devant du site pour ajouter des messages utiles.

L'objectif jusqu'ici était de montrer comment traiter un formulaire de façon simple et sécurisée, tout en offrant une expérience acceptable sur le devant du site. Pourtant, vous pouvez voir qu'il y a pas mal de choses à penser !

Mais ce n'est pas fini. On peut encore faire mieux.

## Soumettre notre formulaire via JavaScript

Pour améliorer l'expérience utilisateur, on peut aussi éviter le rafraîchissement de page et les redirections pour à la place tout faire en JavaScript.

Pour ça, on va faire une requête AJAX (Asynchronous JavaScript And XML) quand le formulaire sera soumis, puis attendre la réponse de notre backend pour modifier le devant du site en conséquence. C'est simplement une requête asynchrone effectué via JavaScript.

**Le JavaScript est censé être une amélioration et ne doit pas remplacer un traitement standard effectué par une bonne vieille requête POST synchrone.**

Assurez-vous que votre formulaire fonctionne bien avec une requête synchrone toute simple en POST vers `admin-post.php` avant de vous occuper du JavaScript. Si vous tombez sur un utilisateur qui a désactivé le JavaScript (possible mais assez rare) ou si pour une raison ou une autre votre script ne s'est pas chargé, alors votre formulaire reste tout à fait utilisable.

**Vous pouvez même vous arrêter là pour notre exemple précis.** Personnellement en tant qu'utilisateur, un formulaire qui fait une requête en POST et qui me redirige après traitement ne me dérange pas du tout. Par contre, une page qui demande à charger des caisses de JavaScript et qui déconne quand j'essaie de soumettre le formulaire, ça, ça m'énerve. Mais bon, c'est moi.

## admin-ajax.php

Comme pour les requêtes simples synchrones, WordPress met à notre disposition un fichier qui va permettre de rester dans l'environnement WordPress et de traiter nos requêtes AJAX. C'est `admin-ajax.php`.

Si vous lisez un peu le fichier `wp-admin/admin-ajax.php`, vous verrez que son fonctionnement est similaire à celui de `admin-post.php`. Il définit la constante `DOING_AJAX`, charge WordPress, déclenche le hook `admin_init`, gère les actions AJAX par défaut de WordPress, puis expose des hooks que nous allons utiliser.

Par contre, vous pourrez remarquer qu'il n'y a pas de hook générique du type `admin_ajax` ou `admin_ajax_nopriv`. Il faut obligatoirement fournir une valeur pour `action`.

Chargeons donc un fichier JavaScript avec notre formulaire. Un simple appel à `wp_enqueue_script()` suffit. Aussi, créez un fichier `script.js` dans le dossier `assets/js`.

```
// in main plugin file
function wpcookbook_testimonial_form( $atts, $content, $tag ){
    ...
    <form id="wpcookbook-testimonial-form" class="wpcookbook-testimonial-form"
method="POST" action="php echo esc_attr( esc_url( admin_url( 'admin-post.php' ) ) ); ?"&gt;"
    ...
    </form>
    <?php wp_enqueue_style( 'wpcookbook-testimonial', plugins_url( 'assets/css/style.css',
__FILE__ ) ); ?>
    <?php wp_enqueue_script( 'wpcookbook-testimonial', plugins_url( 'assets/js/script.js',
__FILE__ ) ); ?>
    ...
}
```

```
// in assets/js/script.js
document.addEventListener('DOMContentLoaded', e => {
    console.log('Hello');
    const form = document.getElementById('wpcookbook-testimonial-form');
    form.addEventListener('submit', event => {
        event.preventDefault();
        console.log('Form submitted!');
    });
});
```

Le petit script attend que tout le DOM soit disponible, puis affiche ‘Hello’ dans la console, pour vérifier que tout est bien chargé. Puis, on prend notre formulaire avec `getElementById()` et on lui attache une fonction de rappel quand l’événement `submit` est déclenché.

Dans cette fonction de rappel, on empêche l’envoi normal du formulaire avec `event.preventDefault()`, puis on affiche un message pour l’instant. Ouvrez la console dans les outils de développement du navigateur, rechargez la page de notre formulaire et faites un petit test.

Your name  
Vincent Dubroeucq

Your website URL  
https://vincentdubroe

Your testimonial  
This is an AJAX test

**Send testimonial**

```
Hello      script.js?ver=5.3:2
Form submitted !  script.js?ver=5.3:6
```

Le script est chargé et la soumission du formulaire sous contrôle.

## Soumettre le formulaire vers admin-ajax.php

Maintenant, il faut envoyer les données du formulaire vers admin-ajax.php. On peut récupérer l'URL vers admin-ajax.php grâce à `admin_url()`, mais il faut passer cette valeur à notre script. Pour cela, on utilise `wp_localize_script()`.

```
wp_localize_script( string $handle, string $object_name, array $l10n )
```

La fonction sert normalement à passer des chaînes traduisibles à utiliser dans le script comme messages ou notices. Vu que traduire des chaînes est compliqué en JavaScript, on utilise la fonction pour passer les chaînes traduites via PHP. C'est très pratique. Sauf qu'ici on ne va pas passer uniquement des chaînes traduites mais on va en profiter pour passer l'URL vers le fichier admin-ajax.php !

Le paramètre `$handle` doit correspondre au script auquel on veut passer des chaînes, `$object_name` est le nom de l'objet dans lequel nos données vont être rangées, et `$l10n` est un tableau associatif de chaînes à passer au script.

Dans notre formulaire, on va donc ajouter ceci:

```
// in main plugin file
function wpcookbook_testimonial_form( $atts, $content, $tag ){
    ...
    <form id="wpcookbook-testimonial-form" class="wpcookbook-testimonial-form"
method="POST" action="<?php echo esc_attr( esc_url( admin_url( 'admin-post.php' ) ) ); ?>">
    ...
    </form>
    <?php wp_enqueue_style( 'wpcookbook-testimonial', plugins_url( 'assets/css/style.css',
__FILE__ ) ); ?>
    <?php wp_enqueue_script( 'wpcookbook-testimonial', plugins_url( 'assets/js/script.js',
__FILE__ ) ); ?>
    <?php wp_localize_script(
        'wpcookbook-testimonial',
        'scriptData',
        array(
            'ajaxurl' => esc_url( admin_url( 'admin-ajax.php' ) ),
            'messages' => wpcookbook_get_form_notices(),
        )
    );?>
    <?php
    return ob_get_clean();
}
```

On passe un objet `scriptData` au script `wpcookbook-testimonial` déclaré à la ligne précédente. Cet objet contient un tableau avec nos messages d'erreur (dont on aura besoin plus tard) et une chaîne `ajaxurl` qui a pour valeur l'URL vers le fichier `admin-ajax.php`. Aussi, on va ajuster notre script en lui demandant d'afficher les données reçues pour vérifier que tout est ok.

```
// in assets/js/script.js
document.addEventListener('DOMContentLoaded', e => {
    console.log('Hello');
    console.log(scriptData);
    const form = document.getElementById('wpcookbook-testimonial-form');
    form.addEventListener('submit', event => {
        event.preventDefault();
        console.log('Form submitted !');
    });
});
```

```

Hello
script.js?ver=5.3:2
script.js?ver=5.3:3

▼{ajaxurl: "https://wpcookbook.local/wp-admin/admin-ajax.php", messages: {...}}
  ajaxurl: "https://wpcookbook.local/wp-admin/admin-ajax.php"
  ▼messages:
    failed: "Something went wrong. Please try again."
    missing-content: "It looks like the content field is missing."
    missing-name: "It looks like the name field is missing."
    success: "Your testimonial has been successfully submitted !"
    unauthorized: "You are not allowed to do this."
    ▶ __proto__: Object
    ▶ __proto__: Object

```

Tout va bien, notre script a accès aux données passées via `wp_localize_script()`

Maintenant, on va ajuster notre script pour faire la requête vers cette URL.

```
// in assets/js/script.js
document.addEventListener('DOMContentLoaded', e => {
  console.log('Hello');
  console.log(scriptData);
  const form = document.getElementById('wpcookbook-testimonial-form');
  form.addEventListener('submit', event => {
    event.preventDefault();
    console.log('Form submitted !');
    fetch(scriptData.ajaxurl, { method: "POST", body: new FormData(form) })
      .then(response => response.json())
      .then(response => console.log(response))
      .catch(error => {
        console.log(error);
      });
  });
});
```

Dans la fonction de rappel déclenchée à la soumission du formulaire, on utilise `fetch` pour faire une requête en POST vers `admin-ajax.php` (dont l'URL est dans `scriptData.ajaxurl`), et dans le corps de la requête on utilise un objet `FormData` pour récupérer tous les champs du formulaire d'un seul coup.

`fetch()` est une fonction asynchrone qui renvoie une promesse, donc il faut attendre sa réponse avant de faire quoi que ce soit. On lui enchaîne donc `then()` qui va prendre une fonction de rappel en paramètre et lui passer la réponse du `fetch()`. Puis il faut convertir la réponse en JSON lisible facilement, qui est fait à l'aide de la méthode `json()`. Mais celle-ci renvoie aussi une promesse ! Donc on enchaîne un second `then()` avec une autre fonction avant de pouvoir faire un bête `console.log()` de notre réponse finale. Si une des promesses n'est pas résolue, le `catch` va capturer les erreurs éventuelles et les afficher.

Pour faire simple : on fait l'appel AJAX, puis (then) on convertit la réponse en JSON, puis on l'affiche. Il faut maintenant hooker une fonction PHP qui va traiter cette requête.

```
// in main plugin file
add_action( 'wp_ajax_wpcookbook_testimonial_form_submit', 'wpcookbook_testimonial_ajax_handler'
);
function wpcookbook_testimonial_ajax_handler(){
    wp_send_json( $_POST );
}
```

Les données envoyées par notre script vont contenir la valeur du champ `action`, donc on peut se hooker directement sur `wp_ajax_wpcookbook_testimonial_form_submit` sans avoir besoin de passer une action à notre script. Notre fonction ne fait rien : elle renvoie simplement les données reçues à l'aide de `wp_send_json()`.

`wp_send_json( mixed $response, int $status_code = null )` permet d'envoyer les données `$response` encodées en JSON en réponse à une requête AJAX. Aussi, la fonction fait `undie()` donc on n'a pas besoin de le faire nous-même.

The screenshot shows a browser window with a dark theme. On the left, there is a testimonial form with fields for 'Your name' (containing 'Vincent Dubroeuqc'), 'Your website URL' (containing 'https://vincentdubroeu'), and 'Your testimonial' (containing 'AJAX test !'). Below the form is a blue button labeled 'Send testimonial'. On the right, the browser's developer tools are open, specifically the 'Console' tab. The console output shows the following JSON response:

```
Hello
script.js?ver=5.3:2
script.js?ver=5.3:3
▶ ajaxurl: "https://wpcookbook.local/wp-admin/admin-ajax.php"
Form submitted !
script.js?ver=5.3:7
script.js?ver=5.3:10
▶ _wp_http_referer: "/testimonial-form/", action: "wpcookbook_testim
ontial_form_submit", name: "Vincent Dubroeuqc", url: "https://vincen
tdubroeuqc.com", content: "AJAX test !"
  action: "wpcookbook_testimonial_form_submit"
  content: "AJAX test !"
  name: "Vincent Dubroeuqc"
  url: "https://vincentdubroeuqc.com"
  _wp_http_referer: "/testimonial-form/"
▶ __proto__: Object
```

Nos données font bien un aller-retour

## Un formulaire, une seule fonction de rappel

Le script fonctionne, tout est en place. Mais ce serait dommage d'utiliser une autre fonction de rappel pour le traitement du formulaire via AJAX, étant donné que le traitement des champs et les messages d'erreurs vont être les mêmes. Si on peut éviter de copier-coller du code, c'est mieux, non ?

La seule chose qui va changer, c'est la façon dont nous allons répondre. Quand on envoie vers `admin-post.php`, on doit faire des redirections vers le formulaire avec les bons paramètres d'URL. Quand on

envoie vers admin-ajax.php, on doit simplement envoyer une réponse JSON.

Ajustons donc notre fonction de rappel de traitement de formulaire comme suit :

```
// in main plugin file
add_action( 'wp_ajax_wpcookbook_testimonial_form_submit', 'wpcookbook_testimonial_form_handler'
);
add_action( 'wp_ajax_nopriv_wpcookbook_testimonial_form_submit',
'wpcookbook_testimonial_form_handler' );
add_action( 'admin_post_wpcookbook_testimonial_form_submit',
'wpcookbook_testimonial_form_handler' );
add_action( 'admin_post_nopriv_wpcookbook_testimonial_form_submit',
'wpcookbook_testimonial_form_handler' );
/**
 * Handles our testimonial form data
 */
function wpcookbook_testimonial_form_handler(){

    $referer = wp_get_referer() ?: home_url();
    $messages = wpcookbook_get_form_notices();
    $error = new WP_Error();

    // Check fields
    ...
    if( $error->has_errors() ){
        if( defined( 'DOING_AJAX' ) && DOING_AJAX ) {
            wp_send_json_error( $error );
        } else {
            wp_safe_redirect( $referer );
            exit;
        }
    }

    // Prepare arguments to create the post
    ...

    // Redirect or send Ajax response
    if( defined( 'DOING_AJAX' ) && DOING_AJAX ) {
        if( ! is_wp_error( $testimonial_id ) ){
            wp_send_json_success();
        } else {
            wp_send_json_error( $testimonial_id );
        }
    } else {
        $message_key = ! is_wp_error( $testimonial_id ) ? 'success' : 'failed';
        $referer = add_query_arg( 'message', $message_key, $referer );
        wp_safe_redirect( $referer );
        exit;
    }
}
```

On hooke donc notre fonction sur les hooks wp\_ajax\_wpcookbook\_testimonial\_form\_submit et wp\_ajax\_nopriv\_wpcookbook\_testimonial\_form\_submit pour que tout fonctionne. Le reste de la fonction est identique, SAUF au niveau des redirections.

On vérifie si on est en train de traiter une requête AJAX avec if( defined( 'DOING\_AJAX' ) &&

DOING\_AJAX ), puis on envoie une réponse JSON appropriée à l'aide de `wp_send_json_error()` ou `wp_send_json_success()`. Ces deux fonctions sont similaires à `wp_send_json()` mais permettent d'accompagner son premier paramètre d'une valeur pour `success` valant `true` ou `false`. En cas d'erreur, on renvoie la `WP_Error` que l'on a construite juste avant, et qui contient donc notre code d'erreur et le message correspondant. En cas de succès, on ne renvoie aucune donnée supplémentaire.

Si on teste sur le devant du site, on obtient ceci en cas d'erreur :

The screenshot shows a browser window with a "Testimonial form" page. On the left, there's a form with fields for "Your name" (containing "Vincent Dubroeuq") and "Your website URL" (containing "https://vincentdubroe"). Below these is a text area labeled "Your testimonial" containing the text "I checked the spam trap checkbox !". At the bottom is a blue "Send testimonial" button. A yellow arrow points to the checkbox input field. On the right, the browser's developer tools are open, specifically the "Console" tab. It displays the following JSON response:

```
Hello
script.js?ver=5.3:2
script.js?ver=5.3:3
▶ {ajaxurl: "https://wpcookbook.local/wp-admin/admin-ajax.php"}
Form submitted !
script.js?ver=5.3:7
script.js?ver=5.3:10
▼ {success: false, data: Array(1)}
  ▼ data: Array(1)
    ▶ 0: {code: "unauthorized", message: "You are not allowed to do ... length: 1 __proto__: Array(0) success: false __proto__: Object
  ▶ >
  ▶ >
```

La réponse contient ce dont nous avons besoin !

On a un objet contenant une valeur pour `success`, et notre tableau de données `data` contenant notre `WP_Error` avec le code et le message d'erreur.

L'erreur a été provoquée en enlevant la classe `screen-reader-text` de notre piège à spams pour faire apparaître la coche et la cocher, simplement.

## Feedback utilisateur

Maintenant, il faut utiliser le retour JSON pour afficher nos notices. Le retour contient les messages d'erreurs et leurs codes. Aussi, nous avons passé tous les messages d'erreur à notre script via `wp_localize_script()`. Donc même si on ne disposait qu'uniquement des codes, on pourrait s'en sortir.

Ajustons notre script

```
document.addEventListener('DOMContentLoaded', e => {
    console.log('Hello');
    console.log(scriptData);
    const form = document.getElementById('wpcookbook-testimonial-form');
    form.addEventListener('submit', event => {
        event.preventDefault();
        const noticeContainer =
            document.getElementById('wpcookbook_testimonial_notice_container');
        noticeContainer.innerHTML = '';
        fetch(scriptData.ajaxurl, { method: "POST", body: new FormData(form) })
            .then(response => response.json())
            .then(response => {
                console.log(response);
                let messages = '';
                if (response.success) {
                    messages = `<p>${scriptData.messages.success}</p>`;
                } else {
                    if (response.data) {
                        response.data.forEach(error => {
                            messages += `<p>${error.message}</p>`;
                        });
                    }
                }
                if (messages) {
                    const className = response.success ? 'wpcookbook-testimonial-success' : 'wpcookbook-testimonial-error';
                    const html = `<div class="wpcookbook-testimonial-notice ${className}">${messages}</div>`;
                    noticeContainer.innerHTML = html;
                }
            })
            .catch(error => {
                console.log(error);
            });
    });
});
```

Quand le formulaire est soumis, on vide notre conteneur de notice.

Puis quand on reçoit la réponse du serveur, on vérifie si c'est un succès. Si oui, alors on crée le paragraphe contenant le message en utilisant le message de succès passé dans `scriptData`. Si non, on peut avoir plusieurs erreurs, donc il faut faire une petite boucle sur le contenu de `response.data` et on concatène nos paragraphes contenant les messages d'erreur.

Enfin, si on a des messages à afficher on créer une `<div>` avec la bonne classe CSS et on injecte la notice juste avant le formulaire avec `element.innerHTML`. En fait, on a en quelque sorte reproduit le

comportement de notre fonction PHP affichant les notices.

## Testimonial form

Your testimonial has been successfully submitted !

Your name

Vincent Dubroeucq

Your website URL

<https://vincentdubroe>



Your testimonial

The form works properly, both with or without JavaScript !

**Send testimonial**

Notre formulaire et ses notices fonctionne ! Avec ou sans JS !

## C'est fini ?

C'était du boulot ! Si vous êtes arrivés jusqu'ici sans vous perdre, félicitations ! C'était le plus gros morceau de ce guide. Essayez de prendre un peu de recul et de contempler ce que vous avez accompli :

- Vous avez créé un type de contenu personnalisé pour stocker vos témoignages, avec sa petite metabox pour le champ URL.
- Vous avez créé un code court pour afficher un formulaire.
- Vous avez mis en place tout ce qu'il faut pour traiter le formulaire de façon synchrone, avec une bonne vieille requête standard de type POST vers admin-post.php.
- Vous avez géré les erreurs et les notices sur le devant du site.
- Vous avez amélioré votre formulaire avec du JavaScript, en faisant les requêtes de soumission de façon asynchrone vers admin-ajax.php, avec gestion des notices et tout et tout !
- Vous avez fait tout ça en limitant le spam, et en vous assurant que ce qui entre en base de données est propre et sécurisé !

Franchement, vous pouvez vous féliciter. Ce n'est pas rien ! Vous avez fait du bon travail. Cela dit, ce n'est pas fini. On peut toujours aller plus loin !

- Un petit spinner dans le bouton du formulaire pour montrer que la requête est en train d'être traité en asynchrone, ça pourrait être sympa.
- Il manque un code court pour visualiser vos témoignages sur le devant du site.
- Le code HTML du formulaire est en dur. Donner la possibilité à d'autres utilisateurs et extensions de filtrer ces champs serait bien. C'est faisable soit via un template surchargeable (on a déjà vu comment faire), ou alors via un tableau de champs filtrable et une fonction outil qui va les afficher. Ou les deux.
- Du coup, si vous laissez la possibilité d'ajouter des champs, il faut aussi laisser la possibilité d'intervenir lors du traitement de ces champs et des erreurs.
- Pour le moment, on n'affiche qu'une seule notice à la fois pour les requêtes synchrones. Que faire si plusieurs erreurs sont déclenchées ? Il faudrait revoir la gestion des paramètres d'URL pour ce faire.

Ce ne sont que des suggestions d'amélioration. Développer proprement une extension représente beaucoup de travail, mais c'est extrêmement gratifiant ! Il y a toujours quelque chose que l'on peut améliorer !

## Ce qu'il faut retenir

- `admin_ajax.php` fonctionne presque comme `admin-post.php`, sauf qu'il faut que vos données envoyées contiennent une valeur pour `action`. Aussi, le fichier déclare une constante utile `DOING_AJAX`.
- On passe l'URL vers le fichier `admin_ajax.php` à notre script en utilisant `wp_localize_script()`. Les données sont accessibles dans le script via un objet.
- Grâce à la constante `DOING_AJAX`, on peut utiliser la même fonction de rappel pour traiter notre formulaire, que la requête soit envoyée vers `admin-post.php` ou `admin_ajax.php`, et simplement différencier les actions de réponse.
- En JavaScript, `fetch()` est très puissant et simple à utiliser. Aussi, on peut facilement récupérer les données de notre formulaire avec `FormData`.

## Bon à savoir

- Il est possible d'échapper des chaînes de caractère en JavaScript mais nous ne pouvons pas traiter de tout dans ce guide. Donc on a échappé les chaînes avant de les renvoyer vers notre script. On a de ce fait pu bénéficier des fonctions outils de WordPress.
- De la même façon, il existe maintenant dans WordPress un script qui permet d'utiliser les fonctions de traduction comme `__()` dans votre JavaScript. Son utilisation n'est pas forcément complexe, mais cela nécessite un peu de mise en place supplémentaire. Donc nous avons utilisé `wp_localize_script()`, qui est plus simple à prendre en main.
- Nous n'avons pas utilisé de nonce, car nous n'avons aucun besoin de vérifier l'authenticité des utilisateurs. Par contre, pour tout formulaire dans l'administration, que vous le gériez via `admin-post.php` ou `admin_ajax.php`, le nonce est essentiel ! Car vous DEVEZ vérifier que l'utilisateur est connecté, qu'il est bien qui il prétend être, qu'il a bien les droits d'entreprendre l'action en question, et qu'il l'a effectuée bien volontairement.

# Comment créer, modifier et gérer les utilisateurs et leurs données

Si votre site n'est pas un site vitrine ou un simple blog, alors il y a de fortes chances que vous ayez à gérer des utilisateurs un jour ou l'autre.

Gérer les utilisateurs n'est pas trop complexe, mais il faut être prudent. Encore une fois (encore !) nous avons toutes les fonctions dont nous avons besoin, incluses dans le cœur de WordPress.

Pour ce chapitre, nous allons créer une petite extension qui va permettre aux utilisateurs de s'inscrire sur le site via un formulaire personnalisé, pour pouvoir accéder à du contenu exclusif.

Ils seront invités pendant leur processus d'inscription à communiquer leur catégorie d'article préférée et auront accès aux articles de cette catégorie réservés aux abonnés. Le contenu protégé sera inaccessible pour les utilisateurs non-connectés, évidemment.

Nous aurons donc besoin de créer les formulaires d'inscription et de connexion, de gérer l'inscription des utilisateurs et la sauvegarde de leur préférence, de leur permettre de gérer leur profil et de leur donner accès au contenu protégé.

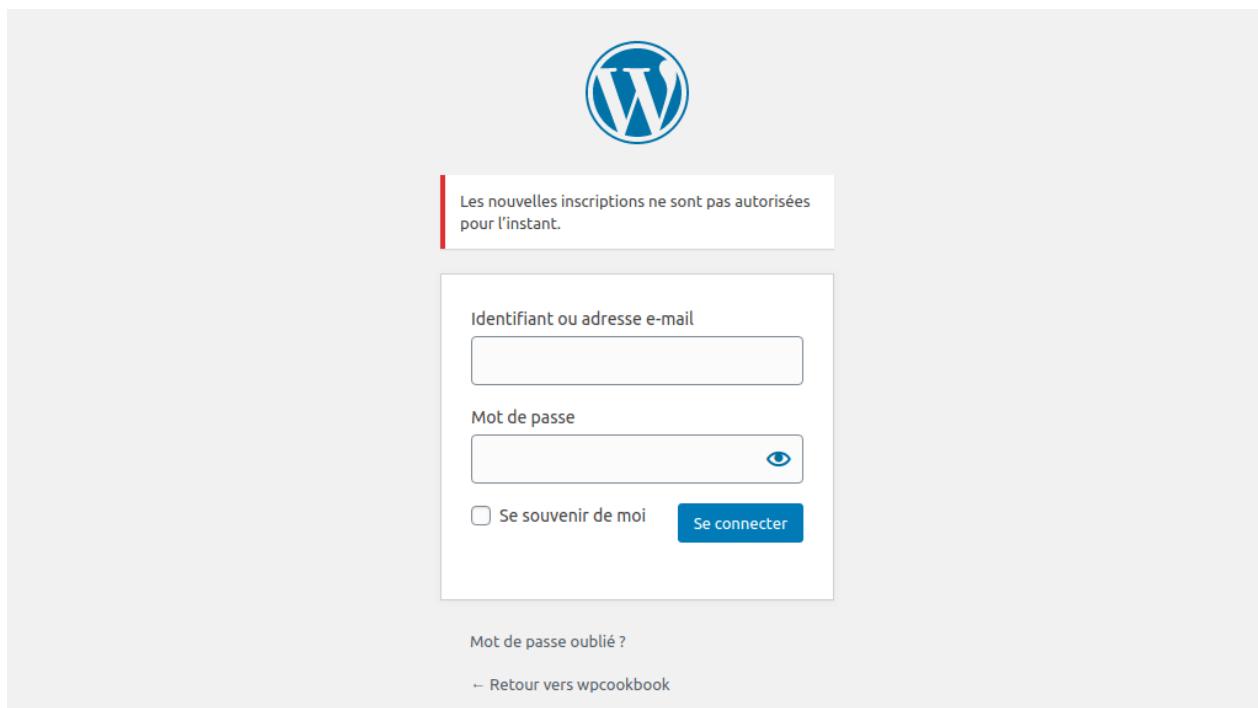
Ready ? C'est parti !

## Le formulaire natif

Avant de coder et réinventer la roue, il faut d'abord étudier ce qui est disponible de base dans WordPress.

Par défaut, WordPress met à notre disposition un formulaire de connexion, disponible à l'URL <https://mon-site.com/wp-login.php>. Quand on tente de se connecter via /wp-admin, on est simplement redirigé vers ce formulaire. Mais il met aussi à notre disposition un formulaire d'inscription disponible à [/wp-login.php?action=register](https://mon-site.com/wp-login.php?action=register).

Les inscriptions ne sont pas autorisées par défaut, donc si vous tentez d'aller sur cette URL, vous aurez un message d'erreur vous indiquant que les inscriptions ne sont pas autorisées et vous serez invités à vous connecter à la place.



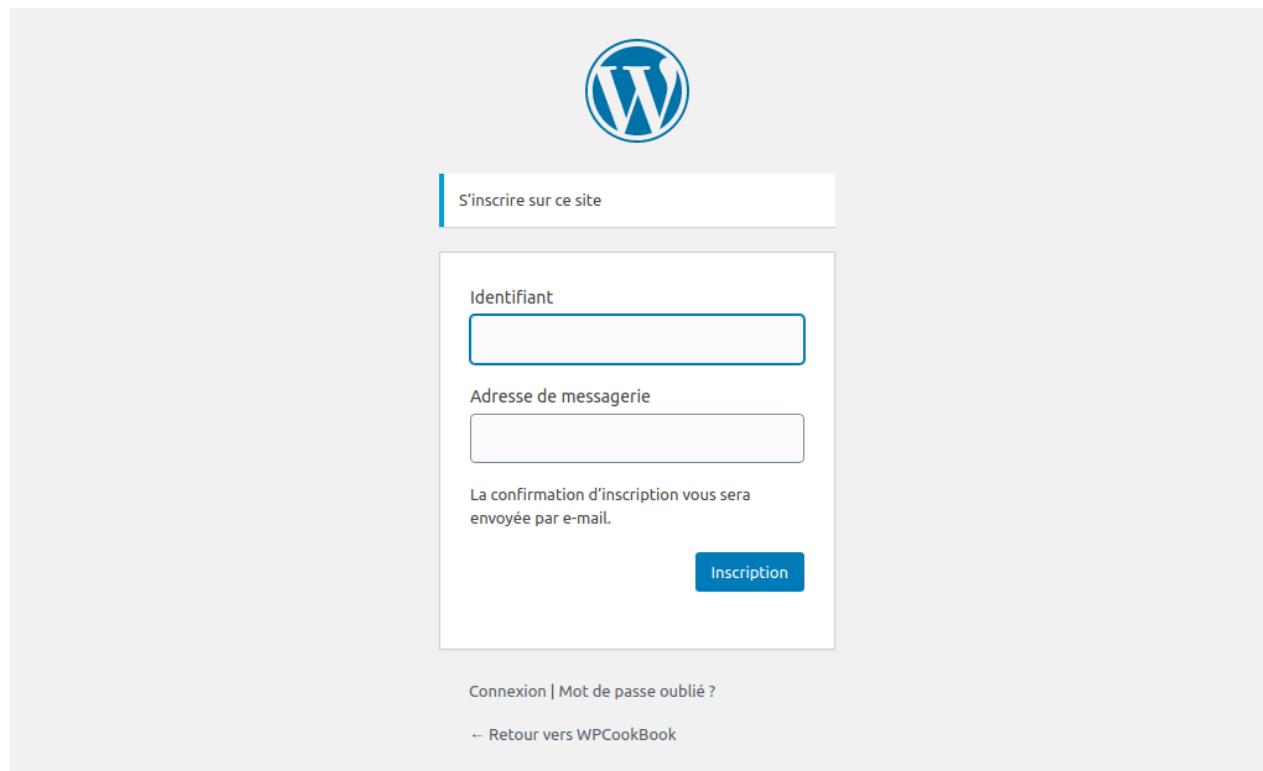
Les inscriptions ne sont pas autorisées par défaut

Pour autoriser les inscriptions et accéder au formulaire correspondant, il faut modifier un réglage situé dans Réglages > Général.

A screenshot of the WordPress admin dashboard under the "Réglages" (Settings) menu, specifically the "Général" (General) tab. On the left is a sidebar with various settings options like "Articles", "Médias", "Pages", etc. The "Réglages" and "Général" items are highlighted in blue. The main content area shows several configuration fields: "Titre du site" (WPCookBook), "Slogan" (Apprenez à développer pour WordPress), "Adresse web de WordPress (URL)" (https://wpcookbook.local), "Adresse web du site (URL)" (https://wpcookbook.local), "Adresse e-mail d'administration" (vincent@vincentdubroeucq.com), and an "Inscription" section which is outlined in yellow. In the "Inscription" section, there is a checkbox labeled "Tout le monde peut s'enregistrer" which is currently unchecked. Below this, another section "Rôle par défaut de tout nouvel utilisateur" has a dropdown menu set to "Abonné".

Il faut autoriser les inscriptions

En cochant cette case, le formulaire de connexion va afficher un petit lien [Inscription](#) menant vers `/wp-login.php?action=register` à côté du lien Mot de passe oublié.



Le formulaire d'inscription est disponible.

Maintenant, les utilisateurs peuvent s'inscrire simplement en entrant un nom d'utilisateur et un email. Ils reçoivent ensuite une invitation à changer leur mot de passe par mail.

## Personnaliser le formulaire natif

Il est possible d'ajouter des champs à ce formulaire via le hook `do_action( 'register_form' )`, et en coulisses, il utilise la fonction `register_new_user()` qui utilise la fonction `wp_create_user()` qui elle-même utilise la fonction `wp_insert_user()`.

Ces trois fonctions peuvent être utilisées pour créer des utilisateurs, et la dernière expose des hooks utiles pour traiter des champs additionnels (notamment `wp_pre_insert_user_data`, qui permet de modifier les données de l'utilisateur juste avant de l'insérer en base de données, et `insert_user_meta`, qui permet de filtrer les métadonnées à enregistrer pour l'utilisateur une fois qu'il a été inséré en base).

On pourrait donc intercepter les données du formulaire sur le filtre `wp_pre_insert_user_data` ou `insert_user_meta` selon les besoins, et y ajouter les valeurs de nos champs personnalisés.

Pour tester, ajoutez le code suivant dans votre extension :

```
add_action( 'register_form', 'wpcookbok_additional_registration_field' );
/**
 * Adds a very useful 'favorite color' field to the default registration form
 */
function wpcookbok_additional_registration_field(){
    ?>
        <label for="favorite-color"><?php _e( 'Favorite Color', '30-forms' ); ?></label>
        <input id="favorite-color" type="text" name="favorite-color" placeholder="orange" />
    </?php
}

add_filter( 'insert_user_meta', 'wpcookbok_handle_additional_registration_field', 10, 3 );
/**
 * Handles our new field during user creation, just before it is inserted in the database
 *
 * @param array $meta Metadata to be inserted
 * @param WP_User $user User being created or updated.
 * @param bool $update Whether the user is being created or updated
 * @return $data
 */
function wpcookbok_handle_additional_registration_field ( $meta, $user, $update ){
    wp_die( var_dump( $meta, $_POST ) );
}
```

Ici, on se hooke sur `register_form` pour ajouter un champ Favorite color, puis on se hooke sur le filtre `insert_user_meta` pour traiter notre métadonnée.

Ce filtre prend trois paramètres qu'on prend soin de passer à notre fonction de rappel :`$meta` est un tableau contenant les données qui vont être insérées en base de données, `$user` est l'objet `WP_User` de l'utilisateur en question et `$update` un booléen qui indique si l'on est en train de mettre à jour ou créer un utilisateur.

Dans la fonction de rappel, je fais simplement un `wp_die()` pour afficher les données dont nous disposons, et vous montrer que l'on dispose bien des données de notre formulaire dans `$_POST`.

```
/home/vincent/Local Sites/wpcookbook/app/public/wp-content/plugins/30-users/30-users.php:96:  
array (size=11)  
  'nickname' => string 'test' (length=4)  
  'first_name' => string '' (length=0)  
  'last_name' => string '' (length=0)  
  'description' => string '' (length=0)  
  'rich_editing' => string 'true' (length=4)  
  'syntax_highlighting' => string 'true' (length=4)  
  'comment_shortcuts' => string 'false' (length=5)  
  'admin_color' => string 'fresh' (length=5)  
  'use_ssl' => int 0  
  'show_admin_bar_front' => string 'true' (length=4)  
  'locale' => string '' (length=0)  
  
/home/vincent/Local Sites/wpcookbook/app/public/wp-content/plugins/30-users/30-users.php:96:  
array (size=5)  
  'user_login' => string 'test' (length=4)  
  'user_email' => string 'test@example.com' (length=16)  
  'favorite-color' => string 'blue' (length=4)  
  'redirect_to' => string '' (length=0)  
  'wp-submit' => string 'Inscription' (length=11)
```

On a bien accès à toutes les données du formulaire d'inscription.

Le formulaire de connexion de WordPress est lui aussi personnalisable, tout comme celui de récupération de mot de passe ! Je vous invite à lire le fichier `wp-login.php` pour découvrir les procédures utilisées et les hooks exposés. On ne va pas se mentir, ce fichier est un peu sale et très peu abstrait. Mais du coup, il est simple à lire et il n'y a pas besoin de naviguer dans tous les fichiers du cœur pour comprendre comment il fonctionne ! Il faut voir le bon côté des choses !

Nous verrons plus en détails comment utiliser toutes les fonctions de création/gestion des utilisateurs au fil de ce chapitre. Pour l'instant, retenez juste que les formulaires par défaut de WordPress sont personnalisables, et peuvent donc tout à fait convenir selon le besoin.

Pour notre exemple, nous allons créer un formulaire de toute pièce, pour que vous puissiez découvrir et apprendre à utiliser les fonctions de WordPress relatives aux utilisateurs. Commentez donc nos deux hooks de tests pour éviter tout parasitage, et en avant.

## Un formulaire d'inscription maison

Créer un formulaire d'inscription de A à Z a ses avantages : votre formulaire est intégré à l'interface du devant du site (vous ne basculez pas vers l'interface classique de WordPress), et il est très facile d'y insérer les champs désirés.

Pour le traitement du formulaire, on peut envoyer ses données vers `admin-post.php` et se hooker sur `admin-post` pour le traiter.

## Créer un formulaire d'inscription

Pour ce formulaire d'inscription, on va faire simple et on va utiliser un code simple et court. Créez une petite extension si ce n'est pas déjà fait et dans le fichier bootstrap de l'extension, ajoutez ceci :

```
// in main plugin file
add_shortcode( 'wpcookbook_registration_form', 'wpcookbook_registration_form' );
/**
 * Registers our registration form shortcode
 *
 * @param array $atts      Array of shortcode attributes
 * @param string $content Content between shortcode tags.
 * @param string $tag      Shortcode tag itself
 * @return string          String of html content
 */
function wpcookbook_registration_form( $atts = array(), $content = null, $tag =
'wpcookbook_registration_form' ){
    ob_start();
    ?>
        <form class="wpcookbook-registration-form" method="POST" action="<?php echo esc_attr(
esc_url( admin_url( 'admin-post.php' ) ) ); ?>">
            <p class="wpcookbook-form-field">
                <label for="username"><?php _e( 'Username', '30-users' ); ?></label>
                <input type="text" id="username" name="username" placeholder="<?php esc_attr_e(
'jdoe', '30-users' ); ?>" />
            </p>
            <p class="wpcookbook-form-field">
                <label for="email"><?php _e( 'Email', '30-users' ); ?></label>
                <input type="email" id="email" name="email" placeholder="<?php esc_attr_e(
'jdoe@example.com', '30-users' ); ?>" />
            </p>
            <p class="wpcookbook-form-field">
                <label for="password"><?php _e( 'Password', '30-users' ); ?></label>
                <input type="password" id="password" name="password" value="<?php echo
esc_attr( wp_generate_password( 16 ) ); ?>" />
                <span class="show-password">
                    <input type="checkbox" id="show_password" name="show_password" />
                    <label for="show_password"><?php esc_attr_e( 'Show password', '30-users'
); ?></label>
                </span>
            </p>
            <input type="hidden" name="action" value="wpcookbook_register" />
            <input type="checkbox" name="cancel-submission" class="screen-reader-text"
tabindex="-1" autocomplete="off" aria-hidden="true" />
            <input type="submit" value="<?php esc_attr_e( 'Register', '30-users' );?>" />
        </form>
    <?php
        wp_enqueue_style( 'wpcookbook-forms', plugins_url( 'inc/forms.css', __FILE__ ), array(),
time() );
        wp_enqueue_script( 'wpcookbook-forms', plugins_url( 'inc/forms.js', __FILE__ ), array(),
time(), true );
    
```

```
    return ob_get_clean();
}
```

Ici on déclare un code court qui va permettre d'afficher un formulaire d'inscription avec le minimum

d'informations : nom d'utilisateur, email et mot de passe. On pourrait ne pas demander de mot de passe, et en générer un nous-même. D'ailleurs, c'est ce que fait le formulaire d'inscription par défaut de WordPress. Idem pour le nom d'utilisateur au final. On pourrait le générer à l'aide de l'email.

On charge aussi un peu de CSS pour la mise en page et un petit script qui permet de voir le mot de passe en basculant le type de champ de `text` à `password` et vice-versa quand la checkbox est cochée.

```
// in forms.js
const checkbox = document.querySelector('.wpcookbook-form-field [type="checkbox"]');
const field = document.querySelector('[name="password"]');

if (checkbox && field) {
    checkbox.addEventListener('change', e => {
        const value = checkbox.checked ? 'text' : 'password';
        field.setAttribute('type', value);
    });
}
```

Si vous allez sur le devant du site, vous devriez obtenir ceci :

## Inscription

Username

jdoe

Email

jdoe@example.com

Password

.....

Show password

**Register**

Notre formulaire d'inscription

Vous remarquerez que notre formulaire envoie ses données vers le fichier `admin-post.php`, en utilisant la méthode POST. Il nous faudra donc traiter notre formulaire exactement comme nous l'avons appris dans le chapitre Comment traiter des données de formulaires. Le formulaire dispose d'un champ caché `action` qui

va nous permettre de nous hooker sur `admin_post` de façon plus précise.

Aussi, pour les mêmes raisons que pour l'exemple du chapitre précédent, nous ne mettons pas de nonce. Les utilisateurs accédant au formulaire d'inscription ne sont pas encore membre du site (logique). Il est donc inutile de chercher à vérifier leur identité et authentifier leur action. Par contre, on va mettre un petit champ caché de type `spam_trap` quand même.

Pour le champ `password`, j'ai utilisé la fonction `wp_generate_password()` pour en suggérer un. Pas forcément indispensable, mais maintenant, vous connaissez l'astuce !

**Note :** Aucun des champs n'est obligatoire, pour pouvoir faire nos propres vérifications en administration plus facilement. Il convient au moins de leur mettre un attribut *required* une fois les vérifications en place. Mais prudence ! Un attribut *required* sur le devant du site ne signifie pas que la donnée sera présente lors du traitement !

## Traiter notre formulaire

Traitons donc notre formulaire en nous hookant au bon endroit.

```

// in main plugin file
add_action( 'admin_post_nopriv_wpcookbook_register', 'wpcookbook_handle_registration_form' );
/**
 * Handles our form registration
 */
function wpcookbook_handle_registration_form(){

    // If the spam box is checked, do nothing.
    if( isset( $_POST['cancel-submission'] ) ){
        wp_die( __( 'You are not allowed to do this.', '30-users' ) );
    }

    $referer = wp_get_referer() ?: home_url( '/profil' );
    $error   = new WP_Error();

    // Check the fields
    if( empty( $_POST['username'] ) ){
        $error->add( 'missing-name', __( 'The username is missing.', '30-users' ) );
    }
    if( empty( $_POST['email'] ) ){
        $error->add( 'missing-email', __( 'The email field is missing.', '30-users' ) );
    }

    // Redirect back to form if any error.
    if( $error->has_errors() ){
        $code      = $error->get_error_codes()[0];
        $referer = add_query_arg( 'error', $code, $referer );
        wp_safe_redirect( $referer );
        exit;
    }

    // Register our user

    // Redirect
    wp_safe_redirect( $referer );
    exit;
}

```

Ici, le code est très similaire à celui produit au chapitre précédent. Nous n'allons pas traiter l'affichage des erreurs sur le devant du site. Je vous laisse vous référer au chapitre Comment traiter des données de formulaires pour l'affichage des notices sur le devant du site. Aussi, nous ne développerons pas la validation ou le traitement du formulaire via JavaScript dans ce chapitre.

Les gardes-fous de base sont en place. Il nous reste à créer notre utilisateur. Pour ce faire, on peut utiliser trois fonctions différentes : `register_new_user()`, `wp_create_user()`, ou `wp_insert_user()`.

`register_new_user( string $user_login, string $user_email )` ne prend que deux paramètres : `$user_login` est le nom d'utilisateur et `$user_email` est l'email de l'utilisateur. C'est la fonction utilisée par le formulaire par défaut de WordPress. Elle va automatiquement générer un mot de passe, vérifier la validité et l'existence du nom d'utilisateur et de l'email passés en paramètre, et va renvoyer une `WP_Error` si elles sont invalides, déjà utilisés ou si la création de l'utilisateur a échoué. En cas de succès, elle renvoie l'identifiant du nouvel utilisateur. Elle est très pratique, mais nous demandons directement un mot de passe dans notre formulaire.

`wp_create_user( string $username, string $password, string $email = '' )` prend un nom d'utilisateur, un mot de passe et un email (optionnel) et va simplement utiliser `wp_insert_user()` pour créer un utilisateur avec toutes les options par défaut.

`wp_insert_user( $userdata )` ne prend qu'un seul paramètre `$userdata` qui peut soit être un tableau d'arguments, un objet standard ou une instance de `WP_User`, et retourne soit une `WP_Error` en cas de souci, soit l'identifiant de l'utilisateur fraîchement créé. C'est la fonction la plus puissante des trois, car elle donne plus de contrôle sur les données de l'utilisateur et c'est celle qui de toute façon est utilisée par les deux autres !

## Validation des champs

La fonction `wp_insert_user( $userdata )` vérifie que le nom d'utilisateur et l'email soient valides et non-utilisés. D'ailleurs, vous remarquerez qu'aucune vérification dans ce sens n'est encore faite lors du traitement de notre formulaire. On peut donc soit le faire nous-même, soit nous reposer sur les vérifications faites par `wp_insert_user()`.

Pour le sport, mais surtout pour découvrir des fonctions utilisées pour ce faire, nous allons vérifier dans un premier temps.

```
function wpcookbook_handle_registration_form(){

    // If the spam box is checked, do nothing.
    if( isset( $_POST['cancel-submission'] ) ){
        wp_die( __( 'You are not allowed to do this.', '30-users' ) );
    }

    $referer = wp_get_referer() ?: home_url();
    $error   = new WP_Error();

    // Check the fields
    if( empty( $_POST['username'] ) ){
        $error->add( 'missing-name', __( 'The username is missing.', '30-users' ) );
    }
    if( username_exists( $_POST['username'] ) ){
        $error->add( 'existing-name', __( 'That username already exists.', '30-users' ) );
    }

    $username = sanitize_user( $_POST['username'], true );
    if( $username !== $_POST['username'] ){
        $error->add( 'invalid-name', __( 'That username is invalid. Only use alphanumeric characters, - and _.', '30-users' ) );
    }

    if( empty( $_POST['email'] ) ){
        $error->add( 'missing-email', __( 'The email field is missing.', '30-users' ) );
    }
    if( email_exists( $_POST['email'] ) ){
        $error->add( 'existing-email', __( 'That email already exists.', '30-users' ) );
    }
    if( ! is_email( $_POST['email'] ) ){
        $error->add( 'not-an-email', __( 'The email field\'s value is not an email.', '30-users' ) );
    }
}
```

```

$email = sanitize_email( $_POST['email'] );
if( $email !== $_POST['email'] ){
    $error->add( 'invalid-email', __( 'The email is invalid.', '30-users' ) );
}

// Redirect back to form if any error.
if( $error->has_errors() ){
    $code    = $error->get_error_codes()[0];
    $referer = add_query_arg( 'error', $code, $referer );
    wp_safe_redirect( $referer );
    exit;
}

// Register our user.
$password = ! empty( $_POST['password'] ) ? $_POST['password'] : wp_generate_password( 16
);
$user_id  = wp_create_user( $username, $password, $email );

// Redirect
if( is_wp_error( $user_id ) ){
    $code = $user_id->get_error_codes()[0];
    $referer = add_query_arg( 'error', $code, $referer );
} else {
    $referer = add_query_arg( 'success', true, $referer );
}
wp_safe_redirect( $referer );
exit;
}

```

On vérifie qu'un email ou nom d'utilisateur n'est pas déjà utilisé à l'aide des fonctions `username_exists( string $username )` et `email_exists( string $email )`. Ces fonctions renvoient l'identifiant de l'utilisateur existant ou `false` s'il n'existe pas déjà. La fonction `is_email( string $email )` vérifie simplement que la chaîne passée en paramètre est bien un email valide.

Les fonctions `sanitize_user( string $username, bool $strict = false )` et `sanitize_email( string $email )` vont nettoyer les chaînes passées en paramètres. Attention, car si on ne fournit pas de feedback à l'utilisateur, il peut se retrouver avec un nom d'utilisateur différent de ce qu'il a entré dans le champ ! D'où notre petite vérification maison.

Encore une fois, on ne s'occupera pas de l'affichage en front, donc lors des redirections, je n'ajoute en paramètre d'URL que la première erreur rencontrée pour faire plus simple.

Une fois tous ces tests passés, on peut utiliser `wp_create_user()` pour créer notre utilisateur, en lui créant un mot de passe si besoin est. La fonction utilise `wp_insert_user()` en coulisses, en effectuant sa petite série de tests aussi et renvoie l'identifiant de l'utilisateur en cas de succès.

Notre utilisateur est bien enregistré !

Notre formulaire fonctionne ! L'utilisateur est enregistré avec le rôle Abonné, qui est le rôle par défaut. Vous pouvez changer cela dans Réglages > Général. Dans notre extension, on pourrait aussi créer un rôle et lui assigner. Référez-vous au chapitre Comprendre les rôles et capacités de WordPress pour réviser comment assigner un rôle.

## Zoom sur `wp_insert_user()`

Cette fonction est la cousine de `wp_insert_post()` qui va s'occuper de créer des publications.

`wp_insert_user( array|object|WP_User $userdata )` prend un seul paramètre qui peut être soit un tableau d'arguments, soit un objet, soit un `WP_User`. Le tableau passé peut prendre les clés suivantes :

- `ID`: l'identifiant de l'utilisateur en question dans le cas d'une mise à jour.
- `user_pass`: le mot de passe de l'utilisateur s'il faut le mettre à jour.
- `user_login`: le nom d'utilisateur
- `user_nicename`: le nom d'utilisateur utilisé dans les URL de page auteur par exemple.
- `user_url`: l'URL de l'utilisateur
- `user_email`: l'email de l'utilisateur
- `display_name`: le nom d'affichage utilisé. Par défaut, le `user_login` est utilisé.
- `nickname`: le surnom. Par défaut, le `user_login` est utilisé aussi.
- `first_name`: le prénom
- `last_name`: le nom
- `description`: la biographie / description de l'utilisateur
- `rich_editing`: `true` par défaut. `false` si non vide. Détermine s'il faut interdire l'accès à l'éditeur WYSIWYG. Si oui, un simple champ texte sera utilisé pour écrire le contenu.
- `syntax_highlighting`: `true` par défaut. `false` si non vide. Indique s'il faut désactiver le surlignage syntaxique dans l'éditeur de code intégré.
- `comment_shortcuts`: `false` par défaut. Indique s'il faut activer les raccourcis clavier pour la

modération de commentaires.

- `admin_color`: le thème de couleurs utilisé dans l'administration. `fresh` par défaut.
- `use_ssl`: booléen indiquant s'il faut forcer l'utilisateur à accéder à l'administration via `https://`. Vaut `false` par défaut.
- `user_registered`: date d'inscription de l'utilisateur
- `user_activation_key`: clé pour réinitialiser le mot de passe. Vide sauf en cas de demande de réinitialisation en cours.
- `spam`: booléen indiquant si l'utilisateur a été signalé.
- `show_admin_bar_front`: indique si l'utilisateur a accès à la barre d'administration sur le devant du site.
- `role`: le rôle de l'utilisateur.
- `locale`: la langue de l'utilisateur. Vide par défaut.

La fonction va vérifier qu'on lui passe bien un identifiant pour savoir si elle doit mettre à jour ou créer l'utilisateur. Tout ce dont elle a besoin est un `user_login` et un `user_pass` lors de la création. Elle va effectuer toutes les vérifications nécessaires : longueur du `user_login`, si les identifiants et emails sont déjà utilisés, etc. Elle est très puissante et simple d'utilisation, tout comme sa cousine.

## Création d'une page Profil

Le nouvel utilisateur est bien enregistré, mais il n'est ni connecté, ni redirigé au bon endroit. Si vos utilisateurs disposent d'une page Tableau de bord ou Profil, c'est un bon endroit pour une redirection.

Créons donc une page Profil, connectons notre utilisateur fraîchement créé et redirigeons-le vers cette page.

```
// in main plugin file
function wpcookbook_handle_registration_form(){

    ...

    // Register our user.
    $username = sanitize_user( $_POST['username'], true );
    $email    = sanitize_email( $_POST['email'] );
    $password = ! empty( $_POST['password'] ) ? trim( $_POST['password'] ) :
    wp_generate_password( 16 );
    $user_id  = wp_create_user( $username, $password, $email );

    // Redirect
    $code = false;
    if( is_wp_error( $user_id ) ){
        $code = $user_id->get_error_codes()[0];
    } else {
        // Log him in
        $user = wp_signon( array(
            'user_login'    => $username,
            'user_password' => $password,
            'remember'      => true,
        ) );
        if( is_wp_error( $user ) ){
            $code = $user->get_error_codes()[0];
        } else {
            $referer = add_query_arg( 'success', true, home_url( 'profil' ) );
        }
    }

    if( $code ){
        $referer = add_query_arg( 'error', $code, $referer );
    }

    wp_safe_redirect( $referer );
    exit;
}
```

Comme expliqué précédemment, `wp_create_user()` utilise `wp_insert_user()` et cette dernière renvoie des `WP_Error` en cas de souci. On peut donc simplifier notre code en supprimant nos vérifications et en nous reposant sur les erreurs retournées par `wp_create_user()`.

Pour connecter un utilisateur, on utilise la fonction `wp_signon( array $credentials, bool $secure_cookie )` qui prend deux paramètres : un tableau de données d'identification et un booléen qui indique s'il faut utiliser un cookie sécurisé. Si vous omettez ce paramètre, la fonction utilise `is_ssl()` pour déterminer si votre site utilise `https://`.

Le tableau d'identifiants doit contenir le `user_login`, le `user_password`, et un booléen `remember` qui va indiquer s'il faut garder le cookie de connexion plus longtemps. On pourrait ajouter une case à cocher `Se souvenir de moi` au formulaire pour gérer ce paramètre. Pour l'instant, mettons cette valeur à `true`.

La fonction renvoie l'instance de `WP_User` en cas de succès ou une `WP_Error` sinon. On va donc modifier notre URL de redirection en fonction du retour de la fonction.

## Notre page Profil

Notre page Profil va afficher des informations sur l'utilisateur courant. Elle n'est donc accessible que pour les utilisateurs connectés. On peut vérifier cela simplement avec `is_user_logged_in()`. Créons un code court pour afficher du contenu basique.

```
add_shortcode( 'wpcookbook_profile', 'wpcookbook_profile' );
/**
 * Displays basic information about the logged in user.
 */
function wpcookbook_profile( $atts = array(), $content = null, $tag = 'wpcookbook_profile' ){
    ob_start();
    ?>
    <?php if ( is_user_logged_in() ): $user = wp_get_current_user(); ?>
        <div class="wpcookbook_profile">
            <?php echo get_avatar($user->user_email, 128); ?>
            <div class="wpcookbook_profile_info">
                <ul>
                    <li><?php _e( 'Username: ', '30-users' ); echo esc_html( $user->user_login );?></li>
                    <li><?php _e( 'Display name: ', '30-users' ); echo esc_html( $user->display_name );?></li>
                    <li><?php _e( 'Email: ', '30-users' ); echo esc_html( $user->user_email );?></li>
                </ul>
            </div>
        </div>
        <?php wp_enqueue_style( 'wpcookbook-profile', plugins_url( 'inc/profile.css', __FILE__ ), array(), time() ); ?>
    <?php else : ?>
        <p><?php _e( 'Sorry, you cannot see this content.', '30-users' ); ?></p>
    <?php endif; ?>
<?php
    return ob_get_clean();
}
```

Si l'utilisateur n'est pas connecté et tente d'accéder à cette page, il ne verra que le message d'erreur. Sinon, il peut consulter quelques informations sur son compte utilisateur.

---

# Profil



- Username: jdoe
- Display name: jdoe
- Email: vdubroeucq@gmail.com

*La page profil.*

On récupère les données de l'utilisateur courant à l'aide de la fonction `wp_get_current_user()`. La fonction renvoie une instance de `WP_User` correspondant à l'utilisateur courant. On a donc toutes ses informations de base. Il n'y a plus qu'à se servir.

Aussi, on a utilisé la fonction `get_avatar()` pour récupérer le gravatar de l'utilisateur, s'il en a un. La fonction prend plusieurs paramètres pour personnaliser la balise `<img />` ou pour déterminer la taille de l'avatar à récupérer. Je vous laisse consulter la documentation de cette fonction pour plus de détails.

## Mettre à jour un utilisateur

Notre page Profil n'est pas très utile si elle ne permet pas d'effectuer d'autres actions, comme mettre à jour notre profil. On peut donc ajouter un petit formulaire sous nos informations pour changer le nom d'affichage, par exemple.

```
/in main plugin file
add_shortcode( 'wpcookbook_profile', 'wpcookbook_profile' );
/**
 * Displays basic information about the logged in user.
 */
function wpcookbook_profile( $atts = array(), $content = null, $tag = 'wpcookbook_profile' ) {
    ob_start();
    ?>
    <?php if ( is_user_logged_in() ): $user = wp_get_current_user(); ?>
        <div class="wpcookbook-profile">
            ...
            <form method="POST" action="<?php echo esc_attr( esc_url( admin_url( 'admin-post.php' ) ) ); ?>">
                <p class="wpcookbook-form-field">
                    <label for="display_name"><?php _e( 'Change your display name', '30-users' ); ?></label>
                    <input type="text" id="display_name" name="display_name" ?>
                </p>
                <?php wp_nonce_field( 'update_profile', 'profile_nonce' ); ?>
                <input type="hidden" name="action" value="update_profile" />
                <input type="submit" value="<?php esc_attr_e( 'Update', '30-forms' ) ?>" />
            </form>
        </div>
        <?php wp_enqueue_style( 'wpcookbook-profile', plugins_url( 'inc/profile.css', __FILE__ ), array(), time() ); ?>
        <?php wp_enqueue_style( 'wpcookbook-forms', plugins_url( 'inc/forms.css', __FILE__ ), array(), time() ); ?>
        <?php else : ?>
            <p><?php _e( 'Sorry, you cannot see this content.', '30-users' ); ?></p>
        <?php endif; ?>
    <?php
    return ob_get_clean();
}
```

J'en profite pour charger notre feuille de style pour les formulaires, à laquelle j'ai ajouté quelques lignes. Voici le devant du site :

WPCookBook — Apprenez à développer pour WordPress  
[Inscription](#) [Profil](#) [Contact](#)

# Profil



- Username: jdoe
- Display name: jdoe
- Email: vdubroeucq@gmail.com

Change your display name

**Update**

Notre page profil et son formulaire.

Vous remarquerez que la barre d'administration est présente, et affiche le nom de l'utilisateur en haut à droite. Cela peut être désirable ou non. On pourra enlever cette barre plus tard, pas de soucis.

Pour l'instant, concentrons-nous sur notre formulaire. Il est structuré de la même façon que ceux vus au chapitre précédent, mais cette fois-ci, on utilise un nonce ! En effet, il est essentiel de s'assurer que la demande de modification du nom d'affichage viennent d'un utilisateur connecté, qui a visité la page. Il faut donc s'assurer de l'identité de l'utilisateur !

Comme pour les autres formulaires créés précédemment, on inclut un champ caché `action` qui va nous permettre de nous hooker sur `admin-post`. Par contre, on se hookera bien sur `admin_post_update_profile` et non `admin_post_nopriv_update_profile`, car on veut que l'utilisateur soit authentifié !

Dans notre extension, ajoutez ceci :

```
add_action( 'admin_post_update_profile', 'wpcookbook_handle_profile_update' );
/**
 * Handles the profile update.
 */
function wpcookbook_handle_profile_update(){
    // Check nonce and permission
    $user = wp_get_current_user();
    if( ! isset( $_POST['profile_nonce'] ) || ! wp_verify_nonce( $_POST['profile_nonce'], 'update_profile' ) || ! current_user_can( 'edit_user', $user->ID ) ){
        wp_die( __( 'You are not allowed to do this.', '30-users' ) );
    }

    $referer = wp_get_referer() ?: home_url();

    // Override display name
    $user->display_name = sanitize_text_field( $_POST['display_name'] );
    $user_id = wp_update_user( $user );

    if( is_wp_error( $user_id ) ){
        $referer = add_query_arg( 'error', $user_id->get_error_codes()[0], $referer );
    } else {
        $referer = add_query_arg( 'success', true, $referer );
    }

    wp_safe_redirect( $referer );
    exit;
}
```

Comme expliqué au chapitre Comment créer une metabox avec des champs personnalisés, on vérifie le nonce à l'aide de `wp_verify_nonce()`. Ensuite, on récupère l'utilisateur courant à l'aide de `wp_get_current_user()` comme nous l'avons fait pour le formulaire, puis on vérifie bien que l'utilisateur ait le droit d'éditer son propre profil. Nous n'avons pas besoin de vérifier que l'utilisateur soit bien connecté, car notre fonction de rappel ne se déclenchera que s'il l'est !

Ensuite, nous allons mettre à jour l'utilisateur en utilisant `wp_update_user( array|object|WP_User $userdata )`. Comme `wp_insert_user()`, la fonction ne prend qu'un seul paramètre, mais celui-ci peut être un tableau, un objet ou un `WP_User`. Et ça tombe bien, car on a un `WP_User` sous le coude grâce à `wp_get_current_user()` ! On va donc simplement lui remplacer son `display_name` !

Et comme pour l'inscription de l'utilisateur, la fonction `wp_update_user()` s'occupe d'effectuer les vérifications nécessaires concernant le `display_name`, car elle utilise aussi `wp_insert_user()`. L'utilisateur passé en paramètre dispose d'un ID, donc la fonction sait qu'elle doit le mettre à jour et non le créer. Easy.

La fonction `wp_update_user()` est aussi utile pour effectuer les changements de mot de passe, car elle va s'occuper d'envoyer les emails de notifications de changement de mot de passe et d'email.

# Profil



- Username: jdoe
- Display name: John Doe, Dummy User
- Email: vdubroeucq@gmail.com

Change your display name

**Update**

Un peu long pour un nom d'affichage.

## Un formulaire de connexion

Nous avons mis en place un formulaire d'inscription et une page profil. Nous savons maintenant créer des utilisateurs et les mettre à jour. Mais qu'en est-il des utilisateurs existants qui reviennent sur le site ? Il doivent pouvoir se connecter ! Certes, ils peuvent le faire via /wp-admin ou /wp-login.php, mais si on a fourni une interface pour s'inscrire autant en fournir une pour se connecter plutôt qu'utiliser celle de WordPress. Et accessoirement, le but c'est d'apprendre à le faire, non ?

### Créer le formulaire de connexion

Pour créer le formulaire, rien de plus simple : WordPress nous fournit une fonction `wp_login_form()` ! La fonction prend un tableau d'arguments pour personnaliser le formulaire.

```

wp_login_form(
    array(
        'echo'          => true,
        'redirect'      => ( is_ssl() ? 'https://' : 'http://' ) . $_SERVER['HTTP_HOST'] .
$_SERVER['REQUEST_URI'], // Default 'redirect' value takes the user back to the request URI.
        'form_id'       => 'loginform',
        'label_username' => __( 'Username or Email Address' ),
        'label_password' => __( 'Password' ),
        'label_remember' => __( 'Remember Me' ),
        'label_log_in'   => __( 'Log In' ),
        'id_username'   => 'user_login',
        'id_password'   => 'user_pass',
        'id_remember'   => 'rememberme',
        'id_submit'     => 'wp-submit',
        'remember'       => true,
        'value_username' => '',
        'value_remember' => false, // Set 'value_remember' to true to default the "Remember me"
checkbox to checked.
    );
);

```

- echo est un booléen indiquant s'il faut afficher ou retourner le code HTML du formulaire.
- redirect est l'URL absolue vers laquelle rediriger l'utilisateur en cas de succès.
- form\_id est l'identifiant du formulaire
- les valeurs des champs de type label\_\* et id\_\* permettent de personnaliser les intitulés des champs correspondants et leurs identifiants.
- remember est un booléen indiquant s'il faut afficher la case à cocher Se souvenir de moi
- value\_username et value\_remember sont les valeurs par défaut des champs correspondants

Pour notre exemple, les valeurs par défaut sont très bien, sauf pour redirect, car nous souhaitons rediriger vers la page Profil après connexion.

Dans notre extension, ajoutez ceci :

```
// in main plugin file
add_shortcode( 'wpcookbook_login_form', 'wpcookbook_login_form' );
/**
 * Displays a simple login form
 */
function wpcookbook_login_form( $atts = array(), $content = null, $tag =
'wpcookbook_login_form' ){
    ob_start();
    ?>
    <?php if( ! is_user_logged_in() ) : ?>
        <div class="wpcookbook-login">
            <?php wp_login_form( array(
                'redirect' => esc_url( home_url( '/profil' ) ),
            ) ); ?>
        </div>
        <?php wp_enqueue_style( 'wpcookbook-forms', plugins_url( 'inc/forms.css', __FILE__ ),
), array(), time() ); ?>
        <?php else: ?>
            <p>
                <?php printf(
                    __( 'You are already logged in. Did you mean to go to your <a
href="%s">profile page</a> ?', '30-users' ),
                    esc_url( home_url( 'profil' ) )
                ); ?>
            </p>
        <?php endif; ?>
    <?php
    return ob_get_clean();
}
```

Ajoutez une page Connexion avec le code court correspondant.

# Connexion

Identifiant ou adresse e-mail

Mot de passe

Se souvenir de moi

**Se connecter**

Notre formulaire est prêt !

L'avantage énorme, c'est que nous n'avons pas à nous soucier du traitement du formulaire ! Si vous inspectez le code HTML généré, vous remarquerez que le formulaire envoie ses données vers `wp-login.php`. Ce qui signifie qu'il utilise exactement la même procédure que le formulaire natif !

D'ailleurs, nous aurions aussi pu envoyer les données de notre formulaire d'inscription vers `wp-login.php` et le laisser travailler au lieu de le traiter nous-même.

## Cacher la barre d'administration

Selon le besoin, il peut être utile de cacher la barre d'administration. Il suffit simplement de se hooker sur `show_admin_bar`.

```
// in main plugin file
add_filter( 'show_admin_bar', 'wpcookbook_show_admin_bar', 10, 1 );
/**
 * Hides the admin bar for Subscribers on the front end
 */
function wpcookbook_show_admin_bar( $show ){
    $user = wp_get_current_user();
    $roles = ( array ) $user->roles;
    if( in_array( 'subscriber', $roles ) ){
        $show = false;
    }
    return $show;
}
```

On récupère les rôles de l'utilisateur courant, et on vérifie qu'ils contiennent bien `subscriber`, car on ne veut pas cacher la barre pour les administrateurs. Si c'est le cas, on renvoie `false`, tout simplement.

Autre méthode encore plus simple. Vous vous souvenez peut-être que `wp_insert_user()` prend un tableau d'arguments et l'un d'eux est `show_admin_bar_front`. Ce réglage contrôle si l'utilisateur doit voir la barre d'administration sur le devant du site. Donc si vous utilisez `wp_insert_user()` pour créer l'utilisateur, vous pouvez simplement passer ce réglage à `false` pour lui cacher la barre d'administration. Si vous utilisez `wp_create_user()` alors vous pouvez simplement ajuster la valeur de la métadonnée correspondante en ajoutant cette ligne en cas de succès :

```
// Create the user
$user_id = wp_create_user( $username, $password, $email );
// Save the metadata to modify the default setting
if( $user_id ){
    update_user_meta( $user_id, 'show_admin_bar_front', false );
}
```

On va parler de la fonction `update_user_meta()` un peu plus loin, pas de panique.

## Ajouter un lien de déconnexion

C'est cool, mais en cachant la barre d'administration, l'utilisateur n'a plus accès au lien de déconnexion. Il faut donc le lui ajouter quelque part, sur la page Profil par exemple.

Dans notre code court affichant le profil, on peut ajouter le lien de déconnexion :

```
// in main plugin file
function wpcookbook_profile( $atts = array(), $content = null, $tag = 'wpcookbook_profile' ){
    ...
    <div class="wpcookbook-profile-info">
        <ul>
            ...
        </ul>
        <a href="php echo esc_url( wp_logout_url() ); ?&gt;"&gt;&lt;?php _e( 'Log out' ); ?&gt;&lt;/a&gt;
    &lt;/div&gt;
    ...
}</pre
```

La fonction `wp_logout_url()` fournit l'URL utilisée nativement pour la procédure de déconnexion, avec le nonce qui va bien. Par contre, quand on clique dessus, on est redirigé vers le formulaire de connexion par défaut de WordPress ! On peut rediriger vers notre formulaire maison en passant son URL comme paramètre pour `wp_logout_url()`.

```
<a href="php echo esc_url( wp_logout_url( home_url( 'connexion' ) ) ); ?&gt;"&gt;&lt;?php _e( 'Log out' ); ?&gt;&lt;/a&gt;</pre
```

Voilà qui est mieux !

## On peut aller plus loin

Nous avons jusqu'à maintenant créé un formulaire d'inscription et mis en place son traitement, nous avons une page *Profil* sur laquelle l'utilisateur peut mettre à jour son profil, un formulaire de connexion, et l'utilisateur peut donc se connecter et déconnecter normalement.

Mais nous n'avons pas de procédure pour la réinitialisation du mot de passe ! Comme petit exercice, je vous laisse le soin d'ajouter un lien de réinitialisation du mot de passe et si vous êtes aventurier, de gérer la procédure associée. Je vous donne quelques indications tout de même, pas de panique !

- La fonction `wp_lostpassword_url()` va vous fournir l'URL du lien. Attention à son paramètre.
- L'URL fournie est filtrable, si vous voulez implémenter vous-même le formulaire et la procédure (filtre `lostpassword_url`).
- Lisez le fichier `wp-login.php` pour connaître la procédure utilisée, les fonctions et URLs utilisées pour les différents formulaires.

Nous allons en rester là pour ce qui est de la procédure d'inscription et connexion. Nous allons nous concentrer sur les champs personnalisés pour les utilisateurs.

## Ajoutons des champs supplémentaires

Revenons à l'idée de base. Le but du jeu était que les utilisateurs puissent s'inscrire pour du contenu exclusif. On peut imaginer que chaque catégorie propose des articles accessibles à tous et d'autres uniquement aux personnes inscrites ET ayant choisi cette catégorie comme catégorie favorite. Il faut donc demander aux utilisateurs lors de leur inscription à quel contenu exclusif ils souhaitent pouvoir accéder. On va donc ajouter un champ supplémentaire sur notre formulaire d'inscription.

J'ai mis en place rapidement trois catégories pour que l'on puisse travailler.

```

// in main plugin file
function wpcookbook_registration_form( $atts = array(), $content = null, $tag =
'wpcookbook_registration_form' ){
    ...
    <form class="wpcookbook-registration-form" method="POST" action="<?php echo esc_attr(
esc_url( admin_url( 'admin-post.php' ) ) ); ?">
    ...
    <p class="wpcookbook-form-field">
        <label for="exclusive-content"><?php _e( 'Exclusive topic you would like to
have access to.', '30-users' ); ?></label>
        <?php $categories = get_categories(); ?>
        <select id="exclusive_content" name="exclusive_content">
            <option value=""><?php esc_html_e( 'Choose a category', '30-users' ); ?>
        </option>
        <?php foreach ( $categories as $category ) : ?>
            <option value="<?php echo esc_attr( $category->term_id ); ?>"><?php
echo esc_html( $category->name ); ?></option>
            <?php endforeach; ?>
        </select>
    </p>
    ...
    </form>
    ...
}

```

On utilise la fonction `get_categories()` pour obtenir la liste des catégories disponibles, sous forme d'objets `WP_Term`. Puis on fait une petite boucle pour afficher un champ `<select>` en passant à chaque `<option>` l'identifiant du terme pour `value`, et le nom de la catégorie.

# Inscription

Username

Email

Password

Show password

Exclusive topic you would like to have access to.

**Register**

Notre nouveau champ est en place.

## Sauvegarder notre donnée

Maintenant, il nous reste à le traiter. Nous ne pouvons pas sauvegarder la valeur du champ dans les données standard d'un WP\_User, comme son nom ou son email. Il nous faut sauvegarder le champ dans les métadonnées de l'utilisateur, exactement comme on l'a déjà fait pour les articles !

Vous vous souvenez ? On a vu que dans WordPress, les contenus sont stockés dans la table wp\_posts mais tous les champs additionnels sont stockés dans la table wp\_postmeta. WordPress fournit donc une petite API pour y accéder facilement: get\_post\_meta(), add\_post\_meta(), update\_post\_meta() et delete\_post\_meta().

Exactement de la même façon, les données utilisateurs sont stockées dans la table wp\_users et leurs métadonnées dans la table wp\_usermeta. Et nous disposons aussi d'une API permettant d'y accéder facilement : get\_user\_meta(), add\_user\_meta(), update\_user\_meta() et delete\_user\_meta() !

Si je vous dis que les commentaires et les termes de taxonomies sont organisés de la même manière, vous n'aurez aucun mal à deviner le nom des tables et des fonctions de l'API associée !

Pour sauvegarder notre champ utilisateur supplémentaire, il nous faut donc utiliser update\_user\_meta() ou add\_user\_meta() lors de l'inscription.

Revenons donc sur la fonction traitant notre formulaire d'inscription :

```

// in main plugin file
function wpcookbook_handle_registration_form(){

    // Register our user.
    $username = sanitize_user( $_POST['username'], true );
    $email    = sanitize_email( $_POST['email'] );
    $password = ! empty( $_POST['password'] ) ? trim( $_POST['password'] ) :
wp_generate_password( 16 );
    $user_id  = wp_create_user( $username, $password, $email );

    // Redirect
    if( is_wp_error( $user_id ) ){
        $code = $user_id->get_error_codes()[0];
    } else {
        // Check if a category was provided
        $category = ! empty( $_POST['exclusive_content'] ) ? (int) $_POST['exclusive_content']
: false;
        if( $category && in_array( $category, get_categories( array( 'fields' => 'ids' ) ) ) ){
            update_user_meta( $user_id, 'wpcookbook_exclusive_content', $category );
        }
    }

    // Log him in
    $user = wp_signon( array(
        'user_login'    => $username,
        'user_password' => $password,
        'remember'      => true,
    ) );

    if( is_wp_error( $user ) ){
        $code = $user->get_error_codes()[0];
    } else {
        $referer = add_query_arg( 'success', true, home_url( 'profil' ) );
    }
}

...
}

```

Nous avons besoin d'un \$user\_id pour utiliser update\_user\_meta(), donc il faut le faire directement après sa création, avant même de le connecter. Nous vérifions que la valeur passée par le formulaire correspond bien à une catégorie existante, et nous sauvegardons la donnée avec update\_user\_meta().

update\_user\_meta( int \$user\_id, string \$meta\_key, mixed \$meta\_value, mixed \$prev\_value = '' ) prend exactement les mêmes paramètres que update\_post\_meta() et fonctionne exactement de la même manière. La donnée sauvegardée est associée à un utilisateur et non à un contenu, c'est tout.

## Ajouter le champ dans notre page profil

Il faut maintenant aussi afficher la valeur sur le profil.

```
// in main plugin file
function wpcookbook_profile( $atts = array(), $content = null, $tag = 'wpcookbook_profile' ){
    ...
    <div class="wpcookbook-profile-info">
        <ul>
            <li><?php _e( 'Username: ', '30-users' ); echo esc_html( $user->user_login );?></li>
            <li><?php _e( 'Display name: ', '30-users' ); echo esc_html( $user->display_name );?></li>
            <li><?php _e( 'Email: ', '30-users' ); echo esc_html( $user->user_email );?></li>
            <?php
                if ( $term_id      = get_user_meta( $user->ID,
                    'wpcookbook_exclusive_content', true ) ) :
                    $category_name = get_category( (int) $term_id )->name;
                ?>
                <li><?php _e( 'Exclusive content: ', '30-users' ); echo esc_html(
                    $category_name );?></li>
            <?php endif; ?>
        </ul>
        <a href="<?php echo esc_url( wp_logout_url( home_url( 'connexion' ) ) ); ?>"><?php _e( 'Log out' ); ?></a>
    </div>
    ...
}
```

On récupère l'identifiant de la catégorie (qui est la valeur sauvegardée, rappelez-vous) en utilisant `get_user_meta()` avec la bonne clé et on passe `true` en troisième paramètre pour ne récupérer qu'une valeur unique et non un tableau. Puis, on utilise cet identifiant de terme de taxonomie s'il existe pour chercher la catégorie correspondante et afficher son nom.

`get_user_meta( int $user_id, string $key = '', bool $single = false )` prend exactement les mêmes paramètres que `get_post_meta()` et fonctionne exactement de la même manière. La donnée récupérée est associée à un utilisateur et non à un contenu, c'est tout.

Et oui, j'ai copié-collé ce dernier paragraphe. Du coup, je ne donnerai pas d'exemple avec `add_user_meta()` ni `delete_user_meta()`, ok ? Ahem.

---

# Profil



- Username: jdoe
- Display name: John Doe, Dummy User
- Email: vdubroeucq@gmail.com
- Exclusive content: JavaScript

[Déconnexion](#)

Change your display name

**Update**

Notre profil affiche correctement notre champ.

## Comment restreindre le contenu

Simple. On va faire une petite metabox pour les articles, avec une bête case à cocher indiquant si le contenu est exclusif ou pas. Les utilisateurs arrivant sur du contenu exclusif devront être connectés et avoir fait le souhait de voir ce contenu pour y accéder.

## Mettre en place notre gestion de contenu en administration

D'abord notre metabox et son traitement :

```
// in main plugin file
add_action( 'add_meta_boxes_post', 'wpcookbook_exclusive_content_metabox' );
/**
 * Registers our exclusive content metabox
 */
function wpcookbook_exclusive_content_metabox(){
    add_meta_box(
        'wpcookbook-exclusive-content-metabox', // ID
        __( 'Exclusive content', '30-users' ), // Title
        'wpcookbook_exclusive_metabox_display', // Callback
        'post', // Screen. Default null.
        'side', // Context. Default 'advanced'
        'low', // Priority. Default 'default'
    );
}

/**
 * Displays our exclusive content metabox
 */
function wpcookbook_exclusive_metabox_display( $post, $args ){
    $checked = get_post_meta( $post->ID, 'wpcookbook_is_exclusive', true );
    wp_nonce_field( 'wpcookbook_exclusive_metabox_save_' . $post->ID,
    'wpcookbook_exclusive_metabox_nonce' );
    ?>
    <div class="components-base-control">
        <div class="components-base-control__field">
            <span class="components-checkbox-control__input-container">
                <style>#wpcookbook-exclusive-checkbox:checked {background-color:#11a0d2; border-color: #11a0d2;}</style>
                <input id="wpcookbook-exclusive-checkbox"
                    name="wpcookbook-exclusive-checkbox"
                    class="components-checkbox-control__input"
                    type="checkbox"
                    <?php checked( $checked ); ?>
                >
                <svg aria-hidden="true" role="img" focusable="false" class="dashicon dashicons-yes components-checkbox-control__checked" xmlns="http://www.w3.org/2000/svg" width="20" height="20" viewBox="0 0 20 20">
                    <path d="M14.83 4.89l1.34 9.44-5.81 8.38H9.02L5.78 9.67l1.34-1.25 2.57 2.4z">
                </path>
            </span>
            <label for="wpcookbook-exclusive-checkbox" class="components-checkbox-control__label"><?php esc_html_e( 'Mark this content as exclusive', '30-users' ); ?></label>
        </div>
    </div>
<?php
```

```

        }

        add_action( 'save_post_post', 'wpcookbook_exclusive_metabox_save', 10, 3 );
        /**
         * Saves our metabox fields
         *
         * @param int      $post_ID Post ID.
         * @param WP_Post  $post    Post object.
         * @param bool     $update   Whether this is an existing post being updated or not.
         */
        function wpcookbook_exclusive_metabox_save( $post_ID, $post, $update ){

            // Check if nonce is set and valid. If not, just early return.
            if ( ! isset( $_POST['wpcookbook_exclusive_metabox_nonce'] ) || ! wp_verify_nonce(
                $_POST['wpcookbook_exclusive_metabox_nonce'], 'wpcookbook_exclusive_metabox_save_'
            . $post->ID ) ) {
                return;
            }

            if ( ! current_user_can( 'edit_post', $post_ID ) ) {
                return;
            }

            if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE ) {
                return;
            }

            if( isset( $_POST['wpcookbook-exclusive-checkbox'] ) ){
                update_post_meta( $post_ID, 'wpcookbook_is_exclusive', true );
            } else {
                delete_post_meta( $post_ID, 'wpcookbook_is_exclusive' );
            }
        }
    }
}

```

La première fonction déclare notre metabox pour les articles, la seconde affiche son contenu, et la troisième sauvegarde la valeur de la case à cocher dans une métadonnée de l'article en question. Pour les détails, je vous renvoie au chapitre [Comment créer une metabox avec des champs personnalisés](#).

The screenshot shows the WordPress dashboard with a post titled "Hello Galaxy" in the center. On the left, there's a sidebar with various menu items like "Tableau de bord", "Articles", "Médias", etc. The main area has buttons for "Enregistrer en brouillon", "Prévisualiser", "Mettre à jour", and a gear icon. A metabox on the right is expanded, showing sections for "Document" and "Bloc". The "Document" section includes fields for "État et visibilité" (set to "Public"), "Publier" (set to "janvier 15, 2020 2:35"), and a checkbox for "Épingler en haut du blog". Below that is a button labeled "Mettre à la corbeille". Other collapsed sections include "Permalien", "Catégories", "Étiquettes", "Image mise en avant", "Extrait", and "Discussion". A specific section, "Exclusive content", is highlighted with a yellow border and contains a checked checkbox labeled "Mark this content as exclusive".

La metabox et son contenu s'affiche et la valeur est sauvegardée normalement.

Marquons un article dans la catégorie JavaScript comme exclusif, et ajoutons un article non-restréint dans cette même catégorie.

## Mettre en place la restriction sur le devant du site

On peut restreindre le contenu de plusieurs manières. On peut par exemple exclure le contenu exclusif des résultats de recherche ou de la boucle principale sauf si l'utilisateur est connecté et a choisi de voir ce contenu, en utilisant le hook `pre_get_posts` par exemple. Pour faire simple, on va utiliser un bête filtre sur `the_content`. On pourrait aussi filtrer `the_excerpt`. Comme vous le souhaitez.

```
add_filter( 'the_content', 'wpcookbook_restrict_content' );
/**
 * Filters the content to display restricted content only to users who have the permission
 *
 * @param string $content Content of the post
 * @return string $content
 */
function wpcookbook_restrict_content( $content ){
    global $post;
    $is_restricted = get_post_meta( $post->ID, 'wpcookbook_is_exclusive', true );
    if( ! is_admin() && is_main_query() && $is_restricted ){
        if( ! is_user_logged_in() ){
            return '<p>' . esc_html__( 'You must be logged in to view this content.', '30-users' )
) . '</p>';
        }

        $user_exclusive_content = get_user_meta( get_current_user_id(),
'wpcookbook_exclusive_content', true );
        if( ! in_category( $user_exclusive_content ) ){
            return '<p>' . esc_html__( 'Sorry, this content is restricted.', '30-users' ) .
'</p>';
        }
    }
    return $content;
}
```

On vérifie simplement que le contenu est marqué comme exclusif, en vérifiant s'il y a bien une valeur pour la meta `wpcookbook_is_exclusive`. Si non, on ne fait rien. Si oui, on va vérifier que l'utilisateur courant est bien connecté pour voir le contenu. Si oui, on va vérifier ensuite que l'article en question inclut bien la catégorie sélectionnée à l'inscription par notre utilisateur.

On utilise `get_current_user_id()` pour récupérer l'identifiant de l'utilisateur courant, et `in_category( int|string|array $category, int|object $post = null )` pour vérifier que la catégorie dont on passe l'identifiant en premier paramètre est bien dans la liste des catégories de l'article courant. Cette fonction est bien pratique, car on peut lui passer un slug, un nom, un identifiant ou un tableau d'identifiants, slugs ou noms ! Elle permet aussi de passer un identifiant de publication en second paramètre et de vérifier cette publication. Par défaut, elle vérifie dans la publication en cours dans la boucle.

Un utilisateur non-connecté ne peut pas voir le contenu d'un article exclusif. Mais il peut voir les articles de la même catégorie qui ne sont pas restreints.

vel maleficia? De apocalypsi gorger omero undead survivor dictum mauris. Hi mindless mortuis soulless creaturas, imo evil stalking monstra adventus resi dentevil vultus comedat cerebella viventium. Qui animated corpse, cricket bat max brucks terribilem incessu zomby. The voodoo sacerdos flesh eater, suscitat mortuos comedere carnem virus. Zonbi tattered for solum oculi eorum defunctis go lum cerebro. Nescio brains an Undead zombies. Sicut malus putrid voodoo horror. Nigh toth eliv ingdead.

👤 vincent ⏰ janvier 15, 2020

💻 JavaScript

💬 Laisser un commentaire

## Hello Galaxy

You must be logged in to view this content.

👤 vincent ⏰ janvier 15, 2020

💻 JavaScript

💬 Laisser un commentaire

Un utilisateur non-connecté n'a pas accès aux articles exclusif.

Notre utilisateur jdoe a choisi la catégorie JavaScript comme catégorie favorite. Il a effectivement accès aux articles restreints de cette catégorie. Par contre, l'utilisateur vincent a beau être connecté, il ne peut pas voir l'article marqué exclusif de la catégorie JavaScript. Simplement parce que vu qu'il n'est pas passé par notre formulaire d'inscription maison (il existait déjà avant), il n'a pas eu l'occasion de communiquer ses préférences. La métadonnée associée est donc absente.

## Ajouter notre champ dans l'administration

Pour laisser la possibilité aux administrateurs de modifier la valeur de notre champ Exclusive content, il faut ajouter le champ dans la page de profil utilisateur. On dispose de deux hooks pour ce faire : `show_user_profile` et `edit_user_profile`. Le premier permet d'afficher du code HTML sur la page d'édition du profil de l'utilisateur quand il clique sur le lien *Mon profil* dans l'administration. Le deuxième permet d'afficher du code HTML quand un administrateur édite le profil d'un autre utilisateur. Vous pouvez ainsi rendre certains champs visibles uniquement par un administrateur (ou plus précisément un utilisateur autorisé à éditer ce profil).

La fonction de rappel utilisée par le hook prend un seul paramètre : le `WP_User` dont il est question. Pour coller au balisage de la page, on va utiliser une `<table>` pour notre réglage.

```

// in main plugin file
add_action( 'show_user_profile', 'wpcookbook_user_fields' );
add_action( 'edit_user_profile', 'wpcookbook_user_fields' );
/**
 * Adds a new section to display our custom user fields.
 *
 * @param WP_User $user The user being edited.
 */
function wpcookbook_user_fields( $user ) {
    $categories = get_categories();
    wp_nonce_field( 'wpcookbook_update_user_' . $user->ID, 'wpcookbook_user_update_nonce' );
    ?>
        <h2><?php esc_html_e( 'Content settings', '30-users' ); ?></h2>
        <table class="form-table" role="presentation">
            <tr>
                <th><label for="exclusive_content"><?php esc_html_e( 'Exclusive content preference: ', '30-users' ); ?></label></th>
                <td>
                    <select id="exclusive_content" name="exclusive_content">
                        <?php foreach ( $categories as $category ) : ?>
                            <option value="<?php echo esc_attr( $category->term_id ); ?>">
                                <?php echo esc_html( $category->name ); ?>
                            <?php endforeach; ?>
                    </select>
                </td>
            </tr>
        </table>
    <?php
}

```

C'est le même code qui est utilisé pour afficher notre `<select>` sur le formulaire d'inscription. Il est juste mis en page dans une `<table>`. La class CSS `form-table` est celle utilisée par WordPress. Aussi, on ajoute un nonce pour vérifier que l'action vienne bien de notre utilisateur.

The screenshot shows the WordPress User Profile edit screen. On the left, there's a sidebar with 'Outils', 'Réglages', and 'Réduire le menu'. The main area has sections for 'Illustration du profil' (with a note about changing profile picture to Gravatar) and 'Gestion de compte'. Under 'Gestion de compte', there are buttons for 'Nouveau mot de passe' (Generate new password) and 'Sessions' (Logout everywhere). A note below the sessions button says: 'Avez-vous perdu votre téléphone ou laissé votre compte ouvert sur un ordinateur public ? Vous pouvez rester connecté ici tout en vous déconnectant partout ailleurs.' In the center, there's a box titled 'Content settings' containing a dropdown menu for 'Exclusive content preference'. The dropdown menu has options: 'Choose a category', 'Choose a category' (which is highlighted in blue), 'CSS', 'JavaScript', and 'WordPress'. A yellow box highlights the 'Content settings' section. At the bottom of the screen, there's a footer note: 'Notre champ est bien disponible mais ne sauvegarde encore rien !' (Our field is available but hasn't saved anything yet!).

Notre champ est bien disponible mais ne sauvegarde encore rien !

## Sauvegarder la donnée

Le champ s'affiche bien, mais maintenant il faut sauvegarder la donnée. Comme pour l'affichage, deux hooks sont à utiliser pour sauvegarder nos champs : `personal_options_update` et `edit_user_profile_update`. Le premier sert quand un utilisateur édite son propre profil. Le second quand un utilisateur autorisé édite le profil d'un autre utilisateur.

```
// in main plugin file
add_action( 'personal_options_update', 'wpcookbook_update_profile' );
add_action( 'edit_user_profile_update', 'wpcookbook_update_profile' );
/**
 * Updates the user's profile with our custom field's value.
 *
 * @param int $user_id ID of the user being updated.
 */
function wpcookbook_update_profile( $user_id ) {

    // Check nonce
    if( ! isset( $_POST['wpcookbook_user_update_nonce'] ) || ! wp_verify_nonce(
        $_POST['wpcookbook_user_update_nonce'], 'wpcookbook_update_user_' . $user_id ) ){
        return;
    }

    if ( ! current_user_can( 'edit_user', $user_id ) ) {
        return;
    }

    $category = ! empty( $_POST['exclusive_content'] ) ? (int) $_POST['exclusive_content'] :
    false;
    if( $category && in_array( $category, get_categories( array( 'fields' => 'ids' ) ) ) ){
        update_user_meta( $user_id, 'wpcookbook_exclusive_content', $category );
    } else {
        delete_user_meta( $user_id, 'wpcookbook_exclusive_content' );
    }
}
```

Il faut bien vérifier le nonce puis que l'utilisateur courant ait les droits nécessaires pour éditer le profil en question. Pour ce faire, on utilise la meta-capacité `edit_user` en passant l'identifiant de l'utilisateur en question. Puis on sauvegarde la donnée si elle est non vide. Si elle l'est alors il faut supprimer la métadonnée, au lieu de ne rien faire. On pouvait se permettre de ne rien faire lors de la création de l'utilisateur mais pas lors de sa mise à jour, car un administrateur pourrait vouloir lui retirer ses droits d'accès.

Notre donnée est bien sauvegardée, mais il faut l'afficher sur le formulaire, en sélectionnant automatiquement l'`<option>` qui convient.

Ajustons notre balisage :

```
// in main plugin file
function wpcookbook_user_fields( $user ) {
    $exclusive_content = get_user_meta( $user->ID, 'wpcookbook_exclusive_content', true );
    $categories        = get_categories();
    wp_nonce_field( 'wpcookbook_update_user_' . $user->ID, 'wpcookbook_user_update_nonce' );
?>
    <h2><?php esc_html_e( 'Content settings', '30-users' ); ?></h2>
    <table class="form-table" role="presentation">
        ...
        <select id="exclusive_content" name="exclusive_content">
            <option value=""><?php esc_html_e( 'Choose a category', '30-users' );
?></option>
            <?php foreach ( $categories as $category ) : ?>
                <option value="<?php echo esc_attr( $category->term_id ); ?>">
<?php selected( $exclusive_content , $category->term_id ); ?> ><?php echo esc_html( $category-
>name ); ?></option>
            <?php endforeach; ?>
        </select>
        ...
    }
}
```

On utilise la fonction `selected( $selected:mixed, $current:mixed, $echo:boolean )` pour afficher l'attribut `selected` sur la bonne option. Elle compare simplement ses deux paramètres et affiche l'attribut s'il sont identiques. Le troisième sert à indiquer s'il faut afficher ou retourner l'attribut. Pour son premier paramètre, on va lui passer la valeur de notre champ que l'on a sauvegardée, et que l'on récupère précédemment grâce à `get_user_meta()`. Pour le second on lui passe la `value` de l'`<option>` courante.

Maintenant, les utilisateurs et les administrateurs peuvent éditer les profils !

## C'est fini ?

Oui et non ! On a vu que les formulaires et procédures de connexion et d'inscription natifs de WordPress sont personnalisables et donc pourraient tout à fait convenir si nos besoins sont simples. Il faut donc vraiment se poser la question si réinventer la roue et coder ses propres formulaires en vaut la chandelle.

Pour l'exemple, nous l'avons fait :

- nous avons créé un formulaire d'inscription, géré son traitement et la création de notre utilisateur,
- nous avons connecté automatiquement notre nouvel utilisateur et l'avons redirigé vers sa page profil.
- nous avons créé une page profil protégée et avons mis en place un formulaire pour mettre à jour ses informations.
- nous avons créé un formulaire de connexion, caché la barre d'administration et ajouté un lien de déconnexion sur le profil.
- Sur la page d'inscription, nous avons ajouté un champ personnalisé dont la valeur est stockée dans les métadonnées de l'utilisateur.
- Nous avons affiché cette donnée sur la page profil.
- Nous avons créé un système de contenu restreint très basique en ajoutant une simple metabox pour les articles et un filtre sur `the_content()` qui va vérifier si l'utilisateur est connecté et a bien le droit de consulter le contenu.

- Nous avons mis en place notre champ personnalisé dans l'administration de WordPress pour permettre aux administrateurs de modifier la valeur de notre nouveau champ sur le profil des utilisateurs.

Ouf !

Vous savez maintenant créer des utilisateurs, les connecter, les mettre à jour, leur ajouter des métadonnées, afficher les champs correspondants sur le devant du site et en administration, personnaliser leur page profil, et utiliser les métadonnées et données utilisateurs standards pour faire ce dont vous avez envie/besoin. Cool.

Pourtant, on est loin d'avoir fini !

- En l'état, pas moyen pour un utilisateur de changer son mot de passe sur sa page profil, ni sa préférence de contenu exclusif !
- Sur le formulaire de connexion, le clic sur le lien Mot de passe perdu mène vers le formulaire natif de WordPress.
- En cas d'échec de connexion, on retourne aussi sur le formulaire natif !
- Notre utilisateur `jdoe` peut quand même accéder à l'administration de WordPress en entrant `/wp-admin` dans la barre d'adresse.
- Quelques lignes de codes sont répétées, notamment la sauvegarde de nos valeurs. Il reste des optimisations à faire !
- Le contenu exclusif apparaît dans les différentes boucles. Pourquoi ne pas complètement retirer les articles concernés des résultats de recherche et des boucles ?

Il y a encore plein de soucis plus ou moins gros, et pleins d'améliorations à faire. Donc à vous de jouer maintenant !

## Ce qu'il faut retenir

- On utilise `register_new_user()`, `wp_create_user()` ou `wp_insert_user()` pour créer des utilisateurs et `wp_insert_user()` ou `wp_update_user()` pour les mettre à jour. Sachant que dans tous les cas, c'est `wp_insert_user()` qui est utilisée au final, car c'est la plus puissante et flexible.
- On utilise `wp_signon()` pour connecter un utilisateur.
- Pour les champs personnalisés que l'on souhaite ajouter, on utilise les hooks `show_user_profile` et `edit_user_profile` pour les afficher sur les pages profil dans l'administration, et on se hooke sur `personal_options_update` et `edit_user_profile_update` pour enregistrer nos valeurs.
- On dispose d'une API pour gérer facilement nos métadonnées : `get_user_meta()`, `add_user_meta()`, `update_user_meta()` et `delete_user_meta()`. Ces fonctions s'utilisent exactement comme celles concernant les publications: `get_post_meta()`, `add_post_meta()`, `update_post_meta()` et `delete_post_meta()`, sauf qu'elles traitent des métadonnées des utilisateurs et non des publications.
- Les formulaires de connexion et d'inscription natifs sont personnalisables, y compris les différentes redirections effectuées. Creusez un peu dans `wp-login.php` pour les hooks accessibles.

# Comment planifier des tâches récurrentes

Sur votre site WordPress, plusieurs actions sont exécutées régulièrement, de façon complètement automatique. Par exemple, votre site va automatiquement vérifier si des mises à jour pour vos thèmes, extensions ou le cœur de WordPress sont disponibles. Vos données temporaires expirées (les fameux transients) sont régulièrement supprimées. Vous pouvez aussi programmer des articles pour qu'ils soient publiés plus tard.

Certaines extensions ajoutent aussi leurs tâches automatisées, comme les extensions qui permettent de faire des sauvegardes régulières de votre site. Vous n'avez rien à faire : la sauvegarde est prévue et s'effectue plus ou moins au moment où vous l'avez demandé automatiquement (on reviendra un peu plus loin sur le plus ou moins).

Tout ça est possible grâce aux tâches planifiées.

## Le CRON

Le CRON (ou CronTab ou Chrono Table) est un système sur votre serveur qui est chargé d'effectuer des tâches prédéfinies à des heures ou intervalles réguliers. Le service tourne en arrière plan et se réveille pour effectuer telle ou telle tâche à l'heure demandée.

C'est un système puissant, mais qui nécessite quelques connaissances et configurations au niveau du serveur pour effectuer ce que l'on veut.

On ne va pas parler plus en détails du "vrai" CRON serveur, car WordPress met à notre disposition un outil super pratique : le WP-CRON ou les tâches planifiées de WordPress.

## WP-CRON vs CRON

Les différences principales entre les tâches planifiées de WordPress et les tâches planifiées du serveur sont la **précision du timing** et le **contexte d'exécution**.

Les tâches planifiées de WordPress sont vérifiées et le cas échéant exécutées à chaque chargement de page. Plus précisément, après que WordPress ait généré une page il va vérifier s'il n'avait pas une ou plusieurs tâches à exécuter à ce moment-là et si oui, il les exécute.

Donc, pour que WordPress exécute ses tâches planifiées, il faut absolument que WordPress se charge. Ça paraît logique. Mais cela implique qu'il faut une visite sur le site ! Soit sur le devant du site, soit dans l'administration, peu importe. Mais il faut charger WordPress !

Ce qui signifie que parfois, il se peut que votre tâche soit exécutée un peu plus tard que prévu ! Par exemple, si vous aviez programmé une sauvegarde à 3h du matin, mais que vous n'avez reçu aucune visite cette nuit, et que votre première visite sur le site est à 7h du matin, alors votre sauvegarde sera faite à ce moment-là !

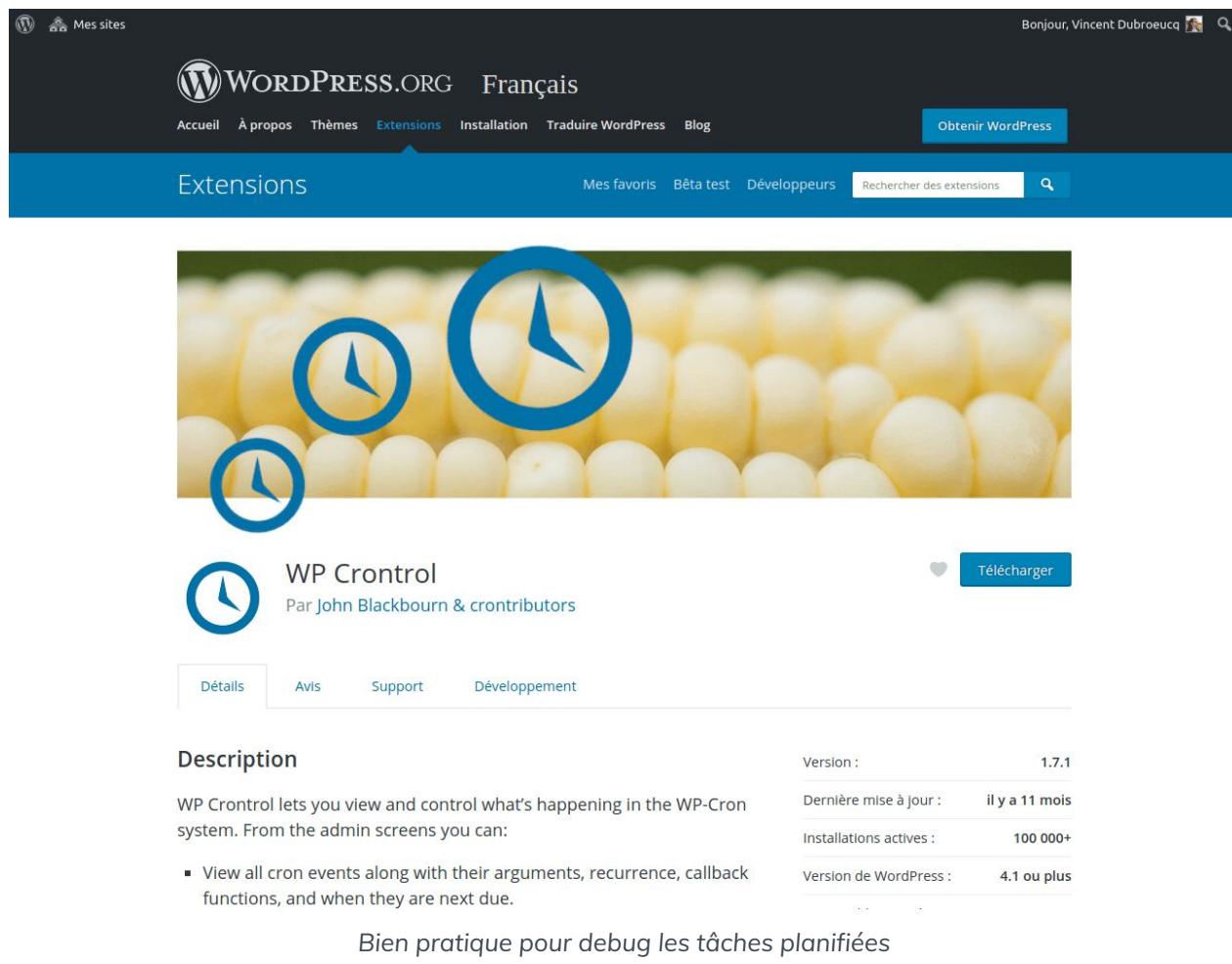
Est-ce un gros souci ? Dans 95% des cas, non, mais bon. C'est important d'en avoir conscience !

L'autre différence est le contexte d'exécution. Quand on planifie une tâche avec WordPress, on enregistre le déclenchement de certains hooks après un chargement de page. Ce qui signifie que l'on reste dans l'environnement WordPress. Nous avons donc accès aux fonctions de WordPress et de nos extensions activées. Ça c'est cool.

Avec un vrai script CRON serveur, nous ne sommes pas dans WordPress. Si on doit exécuter une tâche spécifique à WordPress, il faut le charger manuellement. Bof. Autant utiliser une tâche planifiée de WordPress.

## Lister les tâches planifiés

Pour visualiser les tâches planifiées par WordPress, on va utiliser une petite extension bien pratique : [WP Crontrol](#).



The screenshot shows the WordPress.org Extensions page. At the top, there's a navigation bar with links for Accueil, À propos, Thèmes, Extensions (which is highlighted in blue), Installation, Traduire WordPress, and Blog. On the right, it says "Bonjour, Vincent Dubroeuq" with a profile picture and a search icon. Below the navigation is a blue header bar with the text "Extensions", "Mes favoris", "Bêta test", "Développeurs", a search input field "Rechercher des extensions", and a magnifying glass icon. The main content area features a large image of yellow corn with three blue clock icons overlaid. Below the image, the plugin name "WP Crontrol" is displayed, along with the developer "Par John Blackburn & contributors". There's a "Télécharger" button with a heart icon. At the bottom of the page, there are tabs for "Détails", "Avis", "Support", and "Développement". To the right, there's a sidebar with plugin statistics: Version 1.7.1, Dernière mise à jour il y a 11 mois, Installations actives 100 000+, and Version de WordPress 4.1 ou plus. A note at the bottom says "Bien pratique pour debug les tâches planifiées".

En installant et activant WP Crontrol, on a accès à une page bien pratique dans Outils > Evénements Cron. Cette page liste tous les événements programmés : le hook déclenché, les arguments passés à la fonction de rappel, la fonction de rappel utilisée, la prochaine exécution, la fréquence et on peut modifier, exécuter ou supprimer chaque tâche.

| Nom du crochet                     | Arguments | Actions                                | Prochaine exécution                          | Fréquence          |
|------------------------------------|-----------|--|--|--------------------|
| wp_privacy_delete_old_export_files | Aucun     | wp_privacy_delete_old_export_files()   | 2020-03-03 15:10:50 (59 minutes 17 secondes) | Une fois par heure |
| recovery_mode_clean_expired_keys   | Aucun     | WP_Recovery_Mode->clean_expired_keys() | 2020-03-03 18:10:46 (3 heures 59 minutes)    | Une fois par jour  |
| wp_version_check                   | Aucun     | wp_version_check()                     | 2020-03-03 18:10:50 (3 heures 59 minutes)    | Deux fois par jour |
| wp_update_plugins                  | Aucun     | wp_update_plugins()                    | 2020-03-03 18:10:51 (3 heures 59 minutes)    | Deux fois par jour |
| wp_update_themes                   | Aucun     | wp_update_themes()                     | 2020-03-03 18:10:51 (3 heures 59 minutes)    | Deux fois par jour |
| wp_scheduled_delete                | Aucun     | wp_scheduled_delete()                  | 2020-03-03 18:22:10 (4 heures 10 minutes)    | Une fois par jour  |
| delete_expired_transients          | Aucun     | delete_expired_transients()            | 2020-03-03 18:22:10 (4 heures 10 minutes)    | Une fois par jour  |
| wp_scheduled_auto_draft_delete     | Aucun     | wp_delete_auto_drafts()                | 2020-03-03 18:22:13 (4 heures 10 minutes)    | Une fois par jour  |

Date et heure actuelle du site : 2020-03-03 14:11:33

[Supprimer les événements sélectionnés](#)

[Ajouter un événement cron](#) [Ajouter PHP Cron Event](#)

Les événements cron déclenchent des actions dans votre code. Un événement cron nécessite un crochet d'action quelque part dans le code, par exemple le fichier `functions.php` dans votre thème.

**Nom du crochet**

**Arguments (facultatif)**  Utiliser un tableau encodé en JSON, tel que `[25]`, `["asdf"]`, ou `["i","want",25,"cakes"]`

**Prochaine exécution**  Maintenant  Demain

Interface de WP Control

On peut même créer des événements planifiés directement via l'interface. Cool.

## Les intervalles

L'API fournie par WordPress pour les tâches planifiées permet par défaut de planifier des tâches toutes les heures, toutes les 12h ou toutes les 24h.

Pour planifier une tâche, on va simplement utiliser la fonction : `wp_schedule_event()` en lui passant l'heure de la première occurrence de l'événement, le nom du hook à déclencher, et un intervalle: '`hourly`', '`twicedaily`' ou '`daily`'. Pas de panique on y revient un peu plus bas.

Le souci, c'est que pour faire nos petites expériences, on va avoir besoin d'un intervalle bien plus court. Dans nos développements en production, effectuer certaines tâches tous les jours peut être suffisant, mais en développement on n'a pas vraiment le temps d'attendre une journée pour voir si notre fonction est bien déclenchée, non ?

On va donc filtrer les intervalles disponibles pour venir ajouter le nôtre. Dans le fichier racine de notre extension, ajoutez le code suivant :

```
add_filter( 'cron_schedules', 'wpcookbook_cron_interval', 10, 1 );
/**
 * Adds a new short WP Scheduled Events interval, for testing purposes
 *
 * @param array $schedules Existing schedules, in array format
 * @return array $schedules
 */
function wpcookbook_cron_interval( $schedules ) {
    $schedules['ten_seconds'] = array(
        'interval' => 10,
        'display'  => __( 'Every ten Seconds', '31-wp-cron' ),
    );
    return $schedules;
}
```

Le filtre nous donne les intervalles disponibles dans un tableau et on vient simplement ajouter le nôtre dans le même format : un tableau ayant pour clé 'ten\_seconds' avec une entrée `interval` ayant pour valeur un nombre de secondes et une entrée `display` ayant pour valeur une chaîne à afficher.

Vous pouvez vous rendre sur Réglages > Planifications Cron pour voir la liste des intervalles disponibles.

The screenshot shows the WordPress dashboard with the 'Réglages' (Settings) menu item selected. On the left, there's a sidebar with various settings categories. The main content area is titled 'Planifications WP-Cron' and contains a table of existing cron intervals. A new row has been added at the bottom of the table:

| Nom  | Intervalle        | Nom affiché                                    | Supprimer |
|--|-------------------|--|-----------|
| ten_seconds  | 10 (10 secondes)  | Every Ten Seconds                              |           |
| hourly   | 3600 (1 heure)    | Une fois par heure                             |           |
| twicedaily   | 43200 (12 heures) | Deux Fois par jour                             |           |
| daily  | 86400 (1 jour)    | Une fois par jour                              |           |
| <a href="#">Gérer les événements cron</a>  |                   |  |           |
| <b>Ajouter une planification cron</b>  |                   |  |           |
| Ajouter une nouvelle planification cron vous permettra de planifier des événements qui se reproduisent dans l'intervalle donnée. |                   |  |           |
| <b>Nom interne</b>   |                   | <input type="text" value="ten_seconds"/>       |           |
| <b>Intervalle (secondes)</b>   |                   | <input type="text" value="10"/>                |           |
| <b>Nom affiché</b>   |                   | <input type="text" value="Every Ten Seconds"/> |           |
| <b>Ajouter une planification cron</b>  |                   |  |           |

Notre nouvel intervalle est bien enregistré.

## Planifier des tâches

Maintenant on peut utiliser notre intervalle pour programmer des événements récurrents. Il suffit d'utiliser deux fonctions: `wp_next_scheduled()` et `wp_schedule_event()`.

```
wp_schedule_event( int $timestamp, string $recurrence, string $hook, array $args = array() )
```

Cette fonction va enregistrer une tâche à effectuer. On lui passe le `$timestamp` (l'horodatage) de la première occurrence de l'événement, une chaîne de caractères correspondant à l'identifiant de l'intervalle à utiliser `$recurrence`, un hook à déclencher `$hook`, et un tableau optionnel d'arguments. Ces arguments seront passés à la fonction de rappel déclenchée par le hook `$hook`.

Elle va simplement déclencher le `$hook` spécifié tous les `$recurrence` à partir de `$timestamp`. Easy, non ?

En fait, on dit simplement à WordPress : “Tu veux bien déclencher ce hook toutes les dix secondes, s'il te plaît ?”

```
wp_next_scheduled( string $hook, array $args = array() )
```

Cette fonction va vérifier si le hook `$hook` fait partie des tâches planifiées à déclencher prochainement, et si oui, va retourner le timestamp de la prochaine occurrence. Sinon, elle retourne `false`.

Elle est très utile pour vérifier qu'un hook n'est pas déjà au programme ou pour annuler une tâche planifiée.

En pratique, cela donne ça :

```
add_action( 'init', 'wpcookbook_schedule_events' );
/**
 * Schedule our different events
 */
function wpcookbook_schedule_events(){
    if ( ! wp_next_scheduled( 'wpcookbook_cron_tasks' ) ) {
        wp_schedule_event( time(), 'ten_seconds', 'wpcookbook_cron_tasks' );
    }
}
```

Ici, on se hooke sur `init`. Dans notre fonction de rappel, on utilise `wp_next_scheduled()` pour vérifier qu'une tâche nommée `wpcookbook_cron_tasks` n'est pas déjà enregistrée. Si non, on programme le déclenchement de ce hook toutes les dix secondes (en utilisant notre nouvel intervalle `ten_second`) à partir de maintenant (`time()` renvoie le timestamp courant), à l'aide de la fonction `wp_schedule_event()`.

On pourrait passer un tableau d'arguments en quatrième paramètre de la fonction. Ce tableau serait passé en paramètre à la fonction de rappel hookée sur notre hook. Souvenez-vous, les tâches planifiées de WordPress sont juste des déclenchements de hooks programmés.

The screenshot shows the WordPress dashboard with the 'Outils' (Tools) menu selected. Under the 'Événements Cron' (Cron Events) section, a table lists scheduled tasks. The table has columns for 'Nom du crochet' (Hook name), 'Arguments' (Arguments), 'Actions' (Action), 'Prochaine exécution' (Next execution), and 'Fréquence' (Frequency). A blue button at the bottom left says 'Supprimer les événements sélectionnés' (Delete selected events).

| Nom du crochet                     | Arguments | Actions                                | Prochaine exécution                        | Fréquence          |                          |                                     |                           |
|------------------------------------|-----------|--|--|--------------------|--------------------------|-------------------------------------|---------------------------|
| wpcookbook_cron_tasks              | Aucun     |  | 2020-03-03 14:41:49 (maintenant)           | Every Ten Seconds  | <a href="#">Modifier</a> | <a href="#">Exécuter maintenant</a> | <a href="#">Supprimer</a> |
| wp_privacy_delete_old_export_files | Aucun     | wp_privacy_delete_old_export_files()   | 2020-03-03 15:10:50 (29 minutes 1 seconde) | Une fois par heure | <a href="#">Modifier</a> | <a href="#">Exécuter maintenant</a> | <a href="#">Supprimer</a> |
| recovery_mode_clean_expired_keys   | Aucun     | WP_Recovery_Mode->clean_expired_keys() | 2020-03-03 18:10:46 (3 heures 28 minutes)  | Une fois par jour  | <a href="#">Modifier</a> | <a href="#">Exécuter maintenant</a> | <a href="#">Supprimer</a> |
| wp_version_check                   | Aucun     | wp_version_check()                     | 2020-03-03 18:10:50 (3 heures 29 minutes)  | Deux fois par jour | <a href="#">Modifier</a> | <a href="#">Exécuter maintenant</a> |                           |
| wp_update_plugins                  | Aucun     | wp_update_plugins()                    | 2020-03-03 18:10:51 (3 heures 29 minutes)  | Deux fois par jour | <a href="#">Modifier</a> | <a href="#">Exécuter maintenant</a> |                           |
| wp_update_themes                   | Aucun     | wp_update_themes()                     | 2020-03-03 18:10:51 (3 heures 29 minutes)  | Deux fois par jour | <a href="#">Modifier</a> | <a href="#">Exécuter maintenant</a> |                           |
| wp_scheduled_delete                | Aucun     | wp_scheduled_delete()                  | 2020-03-03 18:22:10 (3 heures 40 minutes)  | Une fois par jour  | <a href="#">Modifier</a> | <a href="#">Exécuter maintenant</a> |                           |
| delete_expired_transients          | Aucun     | delete_expired_transients()            | 2020-03-03 18:22:10 (3 heures 40 minutes)  | Une fois par jour  | <a href="#">Modifier</a> | <a href="#">Exécuter maintenant</a> |                           |
| wp_scheduled_auto_draft_delete     | Aucun     | wp_delete_auto_drafts()                | 2020-03-03 18:22:13 (3 heures 40 minutes)  | Une fois par jour  | <a href="#">Modifier</a> | <a href="#">Exécuter maintenant</a> |                           |

Date et heure actuelle du site : 2020-03-03 14:41:49

Notre hook est bien enregistré.

Note : Les fonctions utilisées ici sont disponibles très tôt lors du chargement de WordPress. J'ai hooké mes trois lignes de code sur *init*, mais j'aurais très bien pu ne pas les hooker du tout, tout comme on peut ne pas hooker un appel à *add\_shortcode()*. Aussi, la vérification des tâches à effectuer est faite à chaque chargement de page, sur *init*. Cela peut être désirable ou non. Une option peut être plus efficace serait de programmer les tâches à l'activation de l'extension. Cela dépend des tâches et de votre besoin.

Voilà ! Vous avez créé votre première tâche programmée ! Le hook `wpcookbook_cron_tasks` est bien déclenché toutes les dix secondes. Mais le souci, c'est qu'on a rien hooké sur ce hook !

## Exécuter une tâche

Notre hook `wpcookbook_cron_tasks` est déclenché. Maintenant, il faut hooker une fonction qui va faire le travail. Pour notre exemple, vu que je sais que vous êtes des bourreaux de travail, on va créer un nouveau brouillon d'article toutes les dix secondes.

```
add_action( 'wpcookbook_cron_tasks', 'wpcookbook_create_drafts', 10 ,1 );
/**
 * Create a draft post.
 * This function is a task scheduled to run on `wpcookbook_cron_tasks`
 *
 * @param array $args Arguments passed when the `wpcookbook_cron_tasks` hook is scheduled.
 */
function wpcookbook_create_drafts( $args = array() ){
    wp_insert_post( array(
        'post_title'  => __( 'New Post', '31-wp-cron' ),
        'post_content' => __( 'Write something interesting.', '31-wp-cron' ),
    ) );
}
```

Si vous avez bien lu le chapitre Comment traiter des données de formulaires, vous avez déjà rencontré la fonction `wp_insert_post()`. Elle permet simplement de créer des publications. Par défaut, elle crée des articles.

| Titre                | Auteur  | Catégories    | Étiquettes | Date                                     |
|----------------------|---------|---------------|------------|--|
| New Post — Brouillon |         | Uncategorized | —          | Dernière modification il.y.a 2 secondes  |
| New Post — Brouillon |         | Uncategorized | —          | Dernière modification il.y.a 13 secondes |
| New Post — Brouillon |         | Uncategorized | —          | Dernière modification il.y.a 29 secondes |
| New Post — Brouillon |         | Uncategorized | —          | Dernière modification il.y.a 39 secondes |
| New Post — Brouillon |         | Uncategorized | —          | Dernière modification il.y.a 54 secondes |
| New Post — Brouillon |         | Uncategorized | —          | Dernière modification il.y.a 1 minute    |
| Hello everyone       | vincent | JavaScript    | —          | Publié 15/01/2020                        |
| Hello Galaxy         | vincent | JavaScript    | —          | Publié 15/01/2020                        |
| Hello Universe       | vincent | CSS           | —          | Publié 15/01/2020                        |
| Hello world!         | vincent | WordPress     | —          | Publié 29/02/2019                        |
| Titre                | Auteur  | Catégories    | Étiquettes | Date                                     |

On va se servir un verre d'eau, on revient et bim. Six articles à écrire.

Tout fonctionne correctement. Un nouvel article est créé toutes les dix secondes. S'il se passe plus de temps entre deux visites, plusieurs articles sont créés.

Si vous tenez un blog multi-auteur, vous pouvez spammer vos rédacteurs de travail avec cette méthode ! Attribuez-leur un nouvel article par semaine, et envoyez-leur une notification par mail avec la fonction `wp_mail()` ! Au travail maintenant !

## Annuler des tâches planifiées

Si vous voulez déprogrammer une tâche, vous pouvez utiliser la fonction `wp_unschedule_event( int $timestamp, string $hook, array $args = array() )`, en lui passant le timestamp de la tâche, le nom du hook et le tableau d'argument.

Le souci est qu'il faut non seulement lui passer le nom de la tâche à déprogrammer, mais aussi le timestamp de la prochaine exécution ! Mais heureusement, on a vu que `wp_next_scheduled()` renvoyait le timestamp de la tâche passée en paramètre. Donc nous pouvons l'utiliser pour vérifier que notre tâche est programmée avant de la dé-programmer.

Aussi, un bon moment pour déprogrammer des tâches est la désactivation de l'extension. Faire le ménage derrière soi est toujours une bonne idée.

```
register_deactivation_hook( __FILE__, 'wpcookbook_deactivation' );
/**
 * De-register scheduled hooks on plugin deactivation
 */
function wpcookbook_deactivation(){
    $timestamp = wp_next_scheduled( 'wpcookbook_cron_tasks' );
    if( $timestamp ){
        wp_unschedule_event( $timestamp, 'wpcookbook_cron_tasks' );
    }
}
```

Attention ! Si vous avez passé des arguments à votre fonction de rappel lors de l'appel à `wp_schedule_event()` (via son 4e paramètre), alors il faut absolument passer les mêmes arguments à `wp_unschedule_event()` ! En effet, vous pouvez très bien exécuter une fonction avec tels paramètres toutes les semaines et tels autres paramètres différents tous les mois ! Donc vous devez être précis dans votre appel à `wp_unschedule_event()` et lui passer le bon hook, le bon timestamp ET le bon tableau d'arguments si besoin.

## Programmer un seul événement

Vous pouvez aussi très bien programmer une seule tâche avec `wp_schedule_single_event( int $timestamp, string $hook, array $args = array() )`. Il suffit de lui passer le timestamp auquel vous voulez effectuer la tâche en question, le nom du hook à déclencher et un tableau d'arguments optionnel.

```
wp_schedule_single_event( strtotime( "now + 1 month" ) , 'wpcookbook_single_tasks' );
```

Ici, on programme le hook `wpcookbook_single_tasks` pour se déclencher dans un mois. Attention, si

vous exécutez ce code à chaque chargement de page, cela va faire beaucoup ! Utilisez `wp_next_scheduled()` pour vérifier que la tâche n'est pas déjà prévue.

## Désactiver les tâches planifiées

Il est possible de désactiver les tâches planifiées de WordPress complètement. En ajoutant `define('DISABLE_WP_CRON', true);` dans votre `wp-config.php`, les tâches planifiées ne seront plus vérifiées ni déclenchées après un chargement de page.

Vous pouvez toujours les déclencher manuellement en appelant le fichier `wp-cron.php`, car ce fichier charge WordPress s'il n'est pas déjà chargé.

Attention cependant, car toutes les actions automatiques programmables de WordPress ne fonctionneront plus. Par exemple, la publication programmée d'articles, les sauvegardes automatiques et les mises à jour.

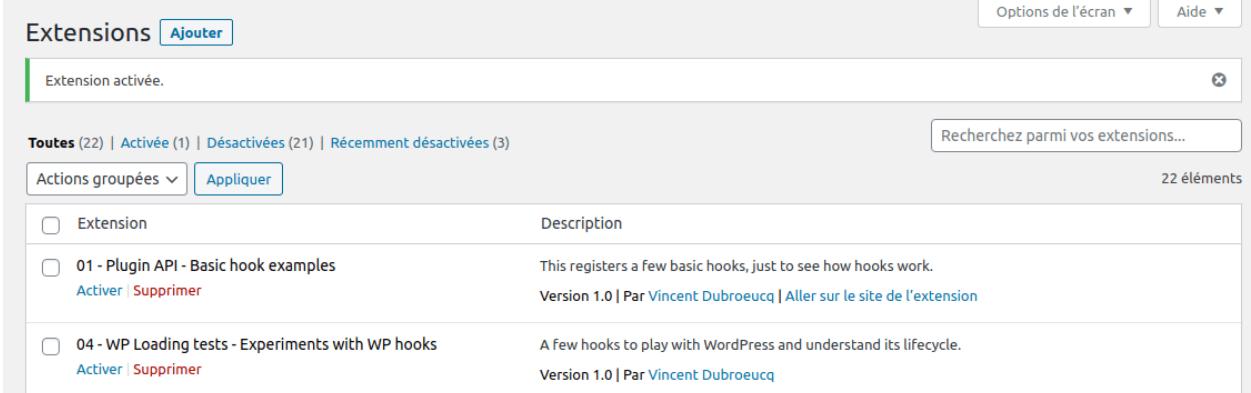
## Ce qu'il faut retenir

- Les tâches planifiées de WordPress ne sont pas très précises au niveau du timing, car elles dépendent des visites sur le site. Un site avec très peu de visites risque de faire son travail un peu en retard.
- L'extension [WP Cron Control](#) est bien pratique pour gérer les tâches planifiées et vérifier que l'on n'a pas accidentellement enregistré plusieurs fois la même tâche.
- On peut ajouter des intervalles d'exécution à l'aide du filtre `cron_schedules`.
- On peut obtenir le timestamp d'une tâche planifiée à l'aide de `wp_next_scheduled()`. Très utile pour vérifier qu'une tâche n'est pas déjà planifiée, ou pour la supprimer.
- `wp_schedule_event()` permet de planifier une tâche récurrente. Attention, elle planifie le déclenchement d'un hook. Il ne faut donc pas oublier d'ajouter une action sur ce hook !
- `wp_schedule_single_event()` permet de programmer une seule tâche.
- `wp_unschedule_event()` permet de dé-programmer une tâche. Attention ! Si vous avez passé des arguments optionnels à `wp_schedule_event()` pour cette tâche, il vous faut passer les mêmes arguments à `wp_unschedule_event()` pour la dé-programmer.

Les tâches planifiées de WordPress sont un outil très puissant et très simple à utiliser. Pensez juste à dé-programmer vos tâches quand vous n'avez plus besoin de les effectuer.

# Comment ajouter des notices dans l'administration de WordPress

Dans WordPress, ou tout autre application web d'ailleurs, les notices sont omni-présentes. A chaque fois que vous installez une extension, l'activez, sauvegardez une page de réglages ou autre, vous avez une jolie notice en haut de votre page d'administration qui va vous informer du succès de votre manœuvre.



The screenshot shows the 'Extensions' screen in the WordPress admin area. At the top, there are buttons for 'Ajouter' (Add), 'Options de l'écran' (Screen Options), and 'Aide' (Help). Below the header, a message says 'Extension activée.' (Extension activated). There are links for 'Toutes' (All) [22], 'Activée' (Active) [1], 'Désactivées' (Deactivated) [21], and 'Récemment désactivées' (Recently deactivated) [3]. A search bar says 'Recherchez parmi vos extensions...' (Search your extensions...). Below the search bar, there are buttons for 'Actions groupées' (Grouped actions) and 'Appliquer' (Apply). On the right, it says '22 éléments' (22 elements). The main list contains two items:

| Extension   | Description   |
|---|---|
| <input type="checkbox"/> 01 - Plugin API - Basic hook examples<br><a href="#">Activer</a>   <a href="#">Supprimer</a>             | This registers a few basic hooks, just to see how hooks work.<br>Version 1.0   Par <a href="#">Vincent Dubroeucq</a>   <a href="#">Aller sur le site de l'extension</a> |
| <input type="checkbox"/> 04 - WP Loading tests - Experiments with WP hooks<br><a href="#">Activer</a>   <a href="#">Supprimer</a> | A few hooks to play with WordPress and understand its lifecycle.<br>Version 1.0   Par <a href="#">Vincent Dubroeucq</a>   |

Un notice classique

Lors de vos développements vous aurez sûrement besoin d'insérer vos propres notices pour vos actions personnalisées. C'est vraiment facile. Voilà la marche à suivre.

## Hooks everywhere

Pas everywhere en fait. Les hooks dont nous avons besoin sont dans le fichier `admin-header.php` dans le dossier `wp-admin` de votre installation.

Il y en a quatre différents :

- `network_admin_notices` : ce hook ne va se déclencher que sur l'administration du réseau
- `user_admin_notices` : ce hook n'est utilisé que par une seule fonction du coeur, pour afficher une notice de confirmation de changement d'email
- `admin_notices` est déclenché sur chaque page d'administration
- `all_admin_notices` : se déclenche dans tous les cas : multisite, administration des utilisateurs, ou page d'administration normale

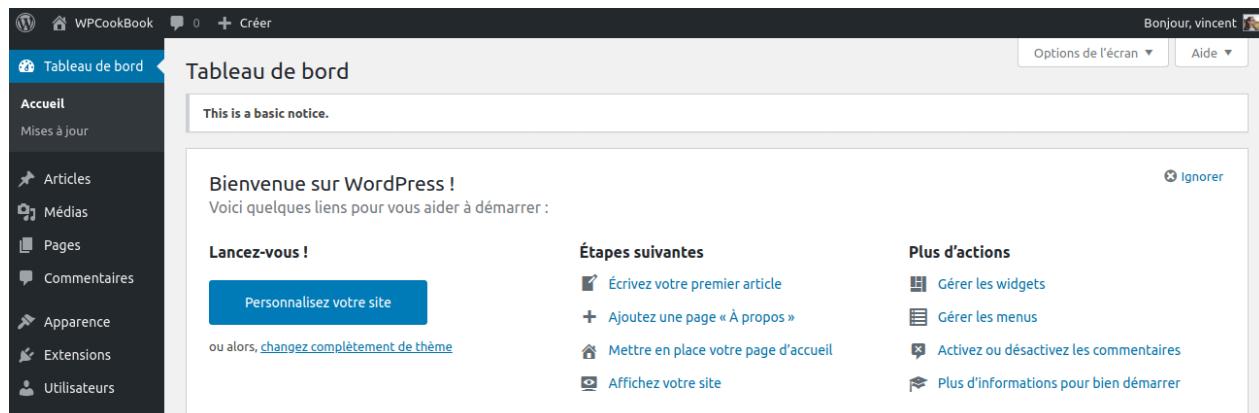
Dans 99% des cas, se hooker sur `admin_notices` suffit. C'est sur celui-ci qu'on va se concentrer.

## Afficher une notice

Le hook `admin_notices` est un hook d'action qui ne passe aucune donnée. Il sert juste à afficher un peu de code HTML. On va donc simplement l'utiliser pour afficher notre petit bout de code HTML, et toute la logique pour gérer l'affichage ou non de la notice doit être contenue dans la fonction de rappel.

```
add_action( 'admin_notices', 'wpcookbook_notices' );
/**
 * Adds a notice in the administration screens
 */
function wpcookbook_notices(){
    ?>
        <div class="notice">
            <p><strong><?php esc_html_e( 'This is a basic notice.', '32-notices' ); ?></strong>
        </p>
    </div>
    <?php
}
```

La fonction ajoute une simple `<div>` avec la classe CSS `notice`, qui est utilisée par le cœur de WordPress.



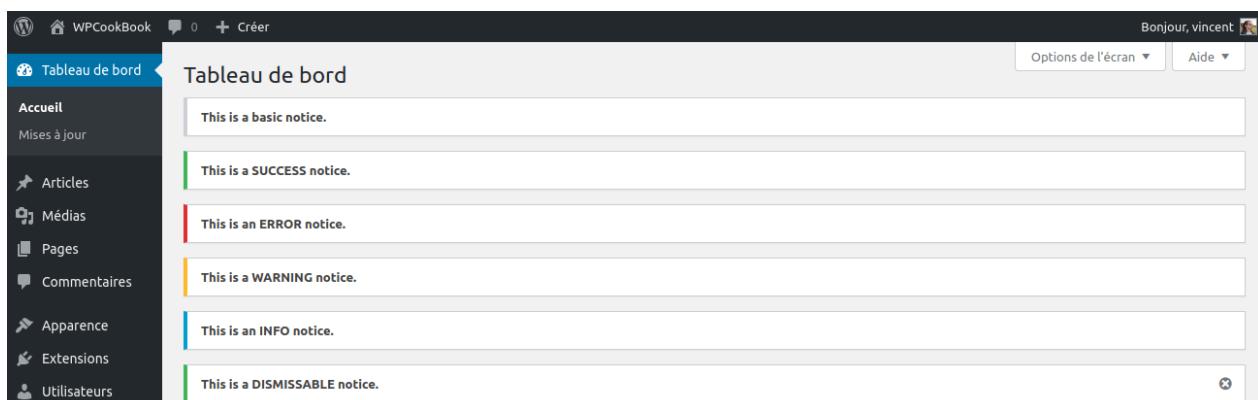
Un notice de base

Pour afficher autre chose que cette notice tristounette, WordPress met à notre disposition plusieurs classes CSS en fonction du type de message que l'on souhaite afficher, et si on veut ou non laisser la possibilité de supprimer la notice : `notice-success`, `notice-error`, `notice-warning`, `notice-info`, et `is-dismissible`.

```

add_action( 'admin_notices', 'wpcookbook_notices' );
/**
 * Adds a notice in the administration screens
 */
function wpcookbook_notices(){
    ?>
        <div class="notice">
            <p><strong><?php esc_html_e('This is a basic notice.', '32-notices' ); ?></strong>
        </p>
        </div>
        <div class="notice notice-success">
            <p><strong><?php esc_html_e('This is a SUCCESS notice.', '32-notices' ); ?></strong>
        </p>
        </div>
        <div class="notice notice-error">
            <p><strong><?php esc_html_e('This is an ERROR notice.', '32-notices' ); ?></strong>
        </p>
        </div>
        <div class="notice notice-warning">
            <p><strong><?php esc_html_e('This is a WARNING notice.', '32-notices' ); ?></strong>
        </p>
        </div>
        <div class="notice notice-info">
            <p><strong><?php esc_html_e('This is an INFO notice.', '32-notices' ); ?></strong>
        </p>
        </div>
        <div class="notice notice-success is-dismissible">
            <p><strong><?php esc_html_e('This is a DISMISSABLE notice.', '32-notices' ); ?></strong>
        </p>
        </div>
    <?php
}

```



Une jolie collection de notices.

Il suffit juste de mettre la bonne classe CSS. Ajouter `is-dismissible` va permettre d'ajouter automatiquement la petite croix pour fermer la notice. Tout cela grâce à un peu de JS magic, déjà incluse par WordPress.

## Contrôler l'apparition des notices

Dans notre exemple, les notices apparaissent à chaque chargement de page. Même celle que l'on peut fermer. Il faut donc contrôler l'affichage de vos notices par un autre moyen.

Selon votre besoin et cas précis, vous allez devoir utiliser des réglages en base de données (en utilisant l'API des Options), des métadonnées utilisateur (comme on a vu au chapitre Comment créer, modifier et gérer les utilisateurs et leurs données) ou des données transitoires (comme on a vu ... ah non, pas encore. Un peu de patience, on y vient !)

Pour travailler sur un exemple plus concret, imaginez que le site accepte les inscriptions utilisateurs et que nous recommandions fortement à nos membres de remplir leur biographie sur leur profil. On pourrait donc afficher une notice à tous les utilisateurs ne l'ayant pas fait.

```
add_action( 'admin_notices', 'wpcookbook_user_notices' );
/**
 * Adds a notice in the administration screens
 */
function wpcookbook_user_notices(){
    $user_id = get_current_user_id();
    $user_description = get_user_meta( $user_id, 'description', true );
    if( empty( $user_description ) ){
        ?>
        <div class="notice notice-warning">
            <p>
                <?php
                    printf(
                        __( 'It looks like your profile description is empty. We highly
recommend you fill it in. You can do so by visiting <a href="%s">your profile page.</a>', '32-
notices' ),
                        esc_url( get_edit_profile_url() )
                    );
                ?>
            </p>
        </div>
        <?php
    }
}
```

Ici, on récupère l'identifiant de l'utilisateur courant, ainsi que sa description qui est stockée dans ses métadonnées et si cette dernière est vide, on lui affiche un avertissement.

Notez au passage la fonction `get_edit_profile_url()` qui prend un identifiant utilisateur optionnel en paramètre et renvoie vers sa page d'édition de profil. Bien pratique.

## Permettre la suppression des notices

On a vu qu'on pouvait simplement faire disparaître les notices en ajoutant la classe `is-dismissible` sur la notice, mais le souci est que malgré ça la notice réapparaît à chaque chargement de page.

Et si notre utilisateur ne voulait pas remplir sa bio ? On va l'embêter comme ça tout le temps, à chaque chargement de page ? Certainement pas ! Il faut donc trouver un moyen d'enregistrer son choix de façon

plus ou moins durable.

Maintenant, il faut définir ce “plus ou moins”. Imaginons que nous voulions rappeler à notre utilisateur qu'il n'a pas rempli sa biographie toutes les semaines. Nous pouvons le faire simplement grâce aux transients.

## Les transients

Les transients sont les données transitoires de WordPress. Simplement, un transient est une donnée temporaire ayant une valeur et une durée d'expiration associée. Le transient peut être stockée en base de données ou en cache.

Ils sont très pratiques pour les données censées expirer ou alors les actions plutôt coûteuses comme les requêtes externes ou les grosses requêtes en base de données. Au lieu d'effectuer cette grosse action à chaque chargement de page, on va stocker son résultat temporairement dans un transient et le réutiliser à chaque fois tant que le transient n'est pas expiré.

## L'API des transients

L'API des transients est très simple. Elle est composée de trois fonctions: `set_transient()`, `get_transient()`, et `delete_transient()`. Les noms sont très explicites, mais on va détailler quand même un minimum :

- `set_transient( string $transient, mixed $value, int $expiration )` permet de créer ou de mettre à jour un transient. On lui passe un nom, une valeur et un entier représentant sa durée de vie en secondes. C'est tout. Si la valeur doit être serialisée avant d'entrer en base, la fonction s'en occupe. Magic.
- `get_transient( string $transient )` récupère la valeur d'un transient. S'il est expiré, il est supprimé et la fonction renvoie `false`.
- `delete_transient( string $transient )` supprime un transient.

Si vous utilisez un multisite, vous avez aussi accès aux fonctions `set_site_transient()`, `get_site_transient()` et `delete_site_transient()`, qui vont faire exactement la même chose que les trois précédentes mais qui vont manipuler des transients disponibles sur tous les sites multisite.

## Programmer un rappel pour notre utilisateur

De retour dans notre fonction `wpcookbook_user_notices()`, on va utiliser un transient pour indiquer si oui ou non il faut afficher la notice.

L'idée est de créer un transient lié à notre utilisateur avec une expiration d'une semaine dès que la notice est affichée. Puis, à chaque chargement de page, on vérifie si ce dernier est expiré ou non. Si oui, alors il est temps d'afficher le rappel ! Peu importe la valeur contenue dans le transient au final.

```
function wpcookbook_user_notices(){
    $user_id = (int) get_current_user_id();
    $name    = "wpcookbook_user_{$user_id}_reminder";
    if( get_transient( $name ) ){
        return;
    }

    $user_description = get_user_meta( $user_id, 'description', true );
    if( empty( $user_description ) ){
        ?>
        <div class="notice notice-warning">
        ...
        </div>
    <?php
    set_transient( $name, true, 20 );
}
}
```

On crée le nom de notre transient (\$name) de manière dynamique de façon à ce qu'il soit unique pour chaque utilisateur. Si on a une valeur pour ce transient (c'est-à-dire si `get_transient()` ne nous renvoie pas `false`), alors il ne faut rien faire. Pas besoin d'aller plus loin.

Par contre, s'il est expiré, cela signifie qu'il est temps d'afficher le rappel. On affiche notre notice et on recrée le transient avec la même valeur d'expiration. C'est reparti pour une semaine.

J'ai donné un délai d'expiration de 20 secondes pour pouvoir tester sans attendre une semaine, évidemment, mais vous pouvez aussi utiliser les constantes magiques de WordPress pour ce faire : `MINUTE_IN_SECONDS`, `HOUR_IN_SECONDS` `DAY_IN_SECONDS`, `WEEK_IN_SECONDS`, `MONTH_IN_SECONDS`, ou `YEAR_IN_SECONDS`.

## Prendre en compte la décision de l'utilisateur

Notre petit rappel est bien moins intrusif que d'afficher la notice à chaque chargement de page.

Mais si l'utilisateur ne voulait VRAIMENT PAS remplir son profil. Il risque d'être ennuyé par les multiples rappels, même s'ils sont espacés d'une semaine. On peut donc prendre en compte son choix en insérant un petit lien `Ne plus me déranger dans la notice`.

```

add_action( 'admin_notices', 'wpcookbook_user_notices' );
/**
 * Adds a notice in the administration screens
 */
function wpcookbook_user_notices(){
    $user_id = (int) get_current_user_id();

    // Check user's decision or transient
    $dismiss = get_user_meta( $user_id, '_wpcookbook_dismiss_notice', true );
    $name    = "wpcookbook_user_${user_id}_reminder";
    if( $dismiss || get_transient( $name ) ){
        return;
    }

    $user_description = get_user_meta( $user_id, 'description', true );
    if( empty( $user_description ) ){
        ?>
        <div class="notice notice-warning is-dismissible">
            <p>
                <?php
                    printf(
                        __( 'It looks like your profile description is empty. We highly
recommend you fill it in. You can do so by visiting <a href="%s">your profile page.</a>. If you
do not wish to enter a description, <a href="%s">click to never be bothered again</a>.', '32-
notices' ),
                        esc_url( get_edit_profile_url() ),
                        '?wpcookbook-dismiss-notice=true'
                    );
                ?>
            </p>
        </div>
        <?php
            set_transient( $name, true, 20 );
        ?
    }
}

add_action( 'admin_init', 'wpcookbook_dismiss_notice' );
/**
 * Records a small metadata if the user chose to ignore our notice.
 */
function wpcookbook_dismiss_notice(){
    if ( isset( $_GET['wpcookbook-dismiss-notice'] ) && 'true' === $_GET['wpcookbook-dismiss-
notice'] ) {
        $user_id = get_current_user_id();
        update_user_meta( $user_id, '_wpcookbook_dismiss_notice', true );
    }
}

```

The screenshot shows the WordPress dashboard with a dark sidebar on the left containing links like Accueil, Articles, Médias, Pages, Commentaires, Apparence, Extensions, Utilisateurs, and Outils. The main content area has a heading 'Tableau de bord' and a message: 'It looks like your profile description is empty. We highly recommend you fill it in. You can do so by visiting [your profile page](#). If you do not wish to enter a description, click to never be bothered again.' Below this, there's a section titled 'Bienvenue sur WordPress !' with a sub-section 'Lancez-vous !' containing a button 'Personnalisez votre site' and text 'ou alors, [changez complètement de thème](#)'. To the right, there are sections for 'Étapes suivantes' (Ecrivez votre premier article, Ajoutez une page « À propos », Mettre en place votre page d'accueil, Affichez votre site) and 'Plus d'actions' (Gérer les widgets, Gérer les menus, Activer ou désactiver les commentaires, Plus d'informations pour bien démarrer). At the top right, it says 'Bonjour, vincent' with a photo placeholder.

L'utilisateur peut maintenant nous faire part de son choix.

L'idée est d'enregistrer une petite `usermeta` contenant sa décision. Avant d'afficher la notice, on va donc vérifier si cette métadonnée utilisateur existe. Si oui, pas besoin d'aller plus loin.

Si non, on peut vérifier notre transient et afficher la notice si besoin. Dans le texte de la notice, notez le lien menant vers la même page, mais avec un paramètre d'URL. Quand l'utilisateur va cliquer ce lien, il va atterrir sur la même page, mais elle aura une URL différente.

On se hooke ensuite sur `admin_init` et on vérifie si notre paramètre d'URL est présent. Si oui, alors on enregistre la petite métadonnée à l'aide de `update_user_meta()`, qui indique qu'il faut ignorer la notice.

C'est vraiment tout simple. On pourrait mieux faire, évidemment. Le clic pourrait déclencher une requête AJAX effectuant la même chose. Cela permettrait de ne pas recharger la page. Aussi, notre lien n'est pas sécurisé. Un nonce permettrait d'authentifier l'action.

Je vous laisse optimiser tout ça ? Maintenant vous êtes plus qu'armés pour le faire seuls ! Si vous avez besoin de réviser, vous pouvez relire le chapitre [Comment traiter les données de formulaire](#). Vous aurez besoin de créer un lien avec un nonce dans les paramètres d'URL. Vous pouvez le faire avec `wp_nonce_url()`. Ou vous pouvez aussi utiliser un élément HTML `<button>` et tout gérer en JavaScript. Comme vous le souhaitez.

## Ce qu'il faut retenir

Les notices sont très simples à créer. Il faut juste utiliser le bon hook.

- Vous pouvez utiliser `network_admin_notices`, `user_admin_notices`, `admin_notices` ou `all_admin_notices` pour afficher des notices.
- Ces hooks permettent juste d'ajouter du code HTML.
- On peut utiliser les classes CSS de WordPress pour personnaliser nos notices et pour qu'elles soient bien intégrées à WordPress : `notice`, `notice-success`, `notice-error`, `notice-warning`, `notice-info`. La classe `is-dismissible` permet d'ajouter le petit bouton de fermeture de la notice. Le clic est géré automatiquement.
- La logique pour afficher ou non la notice et à insérer dans la fonction de rappel. Il faut donc gérer les choix utilisateurs via des transients ou des métadonnées utilisateur.

Et c'est à peu près tout ce qu'il y a à savoir sur les notices WordPress. En ce qui concerne les transients, ce n'est pas bien plus compliqué :

- les transients sont des données temporaires stockées en cache ou en base de donnée, dans la table `wp_options`.
- `set_transient()` permet de créer ou mettre à jour un transient. Il suffit de lui passer le nom du transient, une valeur, et un délai d'expiration. WordPress s'occupe de sérialiser la donnée si besoin est.
- `get_transient()` renvoie la valeur d'un transient ou `false` s'il n'existe pas ou a expiré.
- `delete_transient()` supprime un transient.

Concernant le délai d'expiration passé à `set_transient()`, je tiens à préciser que c'est un délai maximal. Il se peut qu'un transient expire avant ce délai si le cache se vide ou en cas de mise à jour de WordPress (qui nettoie tous les transients).

On peut aussi ne pas passer de délai d'expiration du tout à la fonction ! Dans ce cas, il faut faire le ménage derrière nous et supprimer nos transients avec `delete_transient()`. Aussi, `get_transient()` va supprimer un transient s'il est expiré.

WordPress ne va donc pas automatiquement nettoyer vos transients expirés. C'est donc une bonne idée de faire le ménage à la désinstallation, par exemple.

Voilà, vous savez tout sur les notices ET les transients ! On a fait d'une pierre deux coups !

# Comment personnaliser les tableaux natifs de WordPress

Pour tous nos types de contenus, WordPress nous fournit des tableaux d'administration bien pratiques. Nos articles sont rangés par date de création, et en un coup d'œil on peut voir les auteurs de chaque article, leurs catégories, on peut les visionner, les supprimer, les trier et effectuer une recherche. On peut même en sélectionner plusieurs pour les supprimer en masse.

Mais parfois on peut avoir besoin d'une colonne supplémentaire pour avoir une information cruciale directement sous les yeux, d'un filtre supplémentaire, d'une option de tri supplémentaire ou d'une nouvelle action groupée.

Dans ce chapitre, on va voir comment faire tout cela.

## Ajouter une colonne

Ajouter une colonne est assez simple. Il suffit d'une fonction pour déclarer la nouvelle colonne, et d'une autre pour son contenu.

Pour déclarer une nouvelle colonne, on utilise le filtre `manage_posts_columns`. La fonction de rappel reçoit deux paramètres : `$columns` et `$post_type`. `$columns` est un tableau avec en clé les identifiants des colonnes et un label pour valeur. `$post_type` est le type de contenu pour ce tableau. Dans notre cas, il vaut `post`, puisqu'on travaille sur la table des articles.

Les colonnes disponibles par défaut sont `cb` (la checkbox utilisée pour les actions de masse), `title`, `author`, `categories`, `tags`, `comments`, et `date`.

Il nous suffit d'ajouter un élément à ce tableau pour ajouter notre colonne !

```
add_filter( 'manage_posts_columns', 'wpcookbook_posts_columns', 10, 2 );
/**
 * Registers a new column
 *
 * @param array $columns Array of existing columns.
 * @param string $post_type 'post' in this case.
 * @return array $columns
 */
function wpcookbook_posts_columns( $columns, $post_type ) {
    $columns['thumbnail'] = __( 'Thumbnail', '33-tables' );
    return $columns;
}
```

Pour notre exemple, on va ajouter une petite vignette avec l'image mise en avant s'il y en a une. On enregistre donc une nouvelle colonne `thumbnail` et on lui donne un label adéquat.

The screenshot shows the WordPress admin interface with the 'Articles' page selected. The top navigation bar includes 'Bonjour, vincent' and links for 'Options de l'écran' and 'Aide'. On the left, a sidebar lists various menu items like 'Tous les articles', 'Ajouter', 'Catégories', etc. The main content area displays a table of posts with columns for 'Titre', 'Auteur/autrice', 'Catégories', 'Étiquettes', 'Date', and 'Thumbnail'. The 'Thumbnail' column is correctly positioned at the end of the row headers and the post list. The table has a header section with filters and a footer section with 'Actions groupées' and 'Appliquer' buttons.

Notre colonne apparaît correctement en bout de tableau.

Maintenant, il nous faut remplir notre colonne. Pour ce faire, on utilise le hook `manage_posts_custom_column`.

La fonction de rappel hookée va recevoir deux paramètres : `$column_name` et `$post_id`. `$column_name` permet de savoir sur quelle colonne on se situe, car pour afficher le tableau, WordPress va boucler sur tous les articles à afficher et va déclencher ce hook pour chaque entrée et chaque colonne personnalisée. On dispose aussi de l'identifiant de l'article pour pouvoir aller chercher les données dont on a besoin.

```
add_action( 'manage_posts_custom_column', 'wpcookbook_posts_custom_column', 10, 2 );
/**
 * Displays our custom column data
 *
 * @param string $column_name Name of the current column
 * @param int    $post_id     Post id.
 */
function wpcookbook_posts_custom_column( $column_name, $post_id ) {
    if( 'thumbnail' === $column_name ){
        the_post_thumbnail( 'thumbnail', array( 'style' => 'max-width: 50px; max-height: 50px' ) );
    }
}
```

Ici, on vérifie le nom de la colonne pour ne pas afficher notre vignette partout, et on utilise simplement `the_post_thumbnail()` pour récupérer notre image. Pour l'exemple, j'ai aussi passé un attribut `style` pour les garder dans une taille moins envahissante.

Notez que je ne me suis pas servi de `$post_id`, simplement car on est dans une boucle standard de WordPress, donc la variable globale `$post` est accessible. Les fonctions standards de WordPress utilisables dans les thèmes et dans la boucle (les templates tags) fonctionnent donc normalement. C'est plutôt pratique.

| Titre          | Auteur/autrice | Catégories | Étiquettes | Date                         | Thumbnail |
|----------------|----------------|------------|------------|------------------------------|-----------|
| Hello everyone | vincent        | JavaScript | —          | Publié<br>15/01/2020 à 16h18 |           |
| Hello Galaxy   | vincent        | JavaScript | —          | Publié<br>15/01/2020 à 14h35 |           |
| Hello Universe | vincent        | CSS        | —          | Publié<br>15/01/2020 à 14h34 |           |
| Hello world!   | vincent        | WordPress  | —          | Publié<br>29/12/2019 à 18h10 |           |

Nos images s'affichent correctement.

## Les tableaux des pages, des types de contenus personnalisés et commentaires

Pour l'exemple, nous avons travaillé sur la table des articles, mais tout ce qu'on a vu ici est applicable sur les autres tableaux, comme celui des pages, ou des types de contenus personnalisés. Il suffit de changer le nom du hook, ou d'utiliser un hook dynamique.

### Les tableaux des pages

Pour les tableaux des pages, le procédé est exactement le même. Il suffit de se hooker sur `manage_pages_columns`. Mais attention, car la fonction de rappel ne prend que le paramètre `$post_columns`, et pas de `$post_type`.

Si on veut ajouter exactement la même colonne que pour nos articles sur la liste des pages, il va falloir ajuster un peu la signature de notre fonction.

```

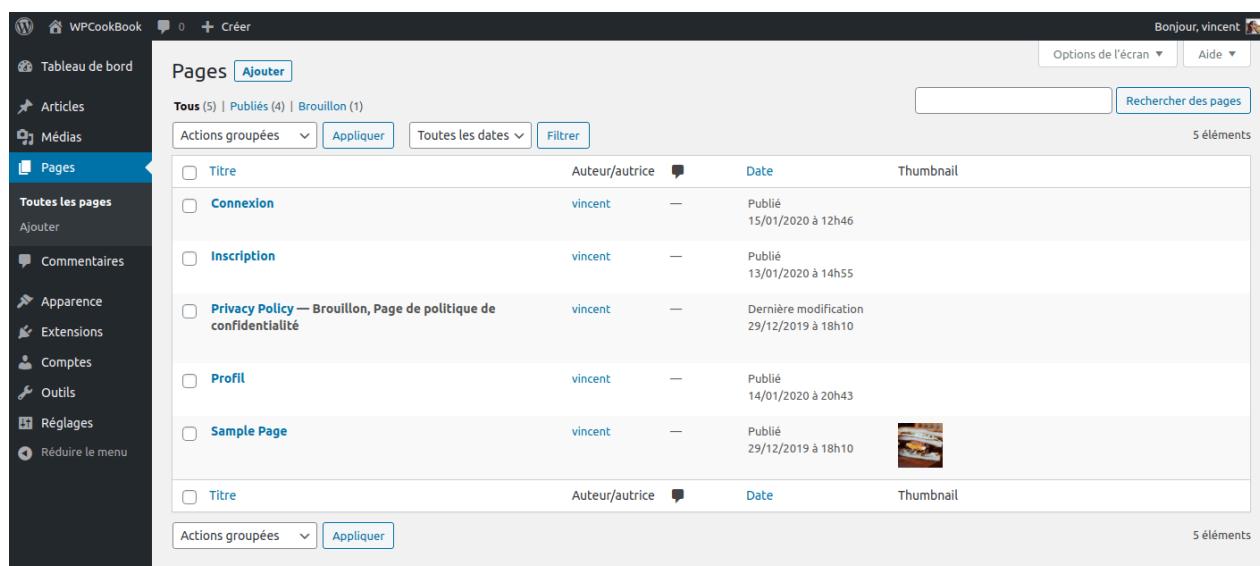
add_filter( 'manage_posts_columns', 'wpcookbook_posts_columns', 10, 2 );
add_filter( 'manage_pages_columns', 'wpcookbook_posts_columns', 10, 1 ); // Only one param !
/**
 * Registers a new column
 *
 * @param array $columns Array of existing columns.
 * @param string $post_type 'post' on posts page.
 * @return array $columns
 */
function wpcookbook_posts_columns( $columns, $post_type = 'page' ) {
    $columns['thumbnail'] = __( 'Thumbnail', '33-tables' );
    return $columns;
}

```

Notez ici qu'un seul paramètre sera passé à la fonction `wpcookbook_posts_columns` par le hook `manage_pages_columns`. Pour rendre le second paramètre de la fonction optionnel, on lui passe une valeur par défaut, simplement.

Pour gérer le contenu des colonnes personnalisées des listes de pages, on utilise le hook `manage_pages_custom_column`. Ce hook fonctionne exactement comme celui pour les articles, donc ici, on peut réutiliser notre fonction de rappel sans souci.

```
add_action( 'manage_pages_custom_column', 'wpcookbook_posts_custom_column', 10, 2 );
add_action( 'manage_posts_custom_column', 'wpcookbook_posts_custom_column', 10, 2 );
/**
 * Displays our custom column data
 *
 * @param string $column_name Name of the current column
 * @param int    $post_id     Post id.
 */
function wpcookbook_posts_custom_column( $column_name, $post_id ) {
    if( 'thumbnail' === $column_name ){
        the_post_thumbnail( 'thumbnail', array( 'style' => 'max-width: 50px; max-height: 50px' ) );
    }
}
```



The screenshot shows the WordPress admin interface for managing pages. The left sidebar is dark-themed and includes links for 'Tableau de bord', 'Articles', 'Médias', 'Pages' (which is selected), 'Toutes les pages', 'Ajouter', 'Commentaires', 'Apparence', 'Extensions', 'Comptes', 'Outils', 'Réglages', and 'Réduire le menu'. The main area has a light background and displays a table titled 'Pages'. The table has columns for 'Titre' (Title), 'Auteur/autrice' (Author), 'Date' (Date), and 'Thumbnail'. There are five rows in the table, each corresponding to a page: 'Connexion' (author vincent, date 15/01/2020), 'Inscription' (author vincent, date 13/01/2020), 'Privacy Policy — Brouillon, Page de politique de confidentialité' (author vincent, date 29/12/2019), 'Profil' (author vincent, date 14/01/2020), and 'Sample Page' (author vincent, date 29/12/2019). A thumbnail image is shown for the 'Sample Page' row. At the bottom of the table, there are buttons for 'Actions groupées' (Grouped actions) and 'Appliquer' (Apply).

Notre colonne image fonctionne aussi sur la liste de nos pages, maintenant.

## Les types de contenu personnalisés

Pour les types de contenus personnalisés, on peut aussi utiliser `manage_posts_columns` et `manage_posts_custom_column`, exactement comme pour les listes d'articles ! Simplement parce que ces deux hooks sont déclenchés pour tous les types de publications autres que les pages.

Si vous activez la petite extension créée dans le chapitre Comment créer un type de contenu personnalisé, vous remarquerez que notre colonne `Thumbnail` fonctionne correctement par défaut sur la liste de nos snippets !

The screenshot shows the WordPress dashboard with the 'Tableau de bord' (Dashboard) selected in the sidebar. The main area is titled 'Extraits de code' (Code Snippets) with a blue 'Ajouter' (Add) button. A search bar and a 'Rechercher des extraits de code' (Search for code snippets) button are at the top right. Below is a table with columns: Titre, Étiquettes, Date, and Thumbnail. One item is listed: 'test' (Publié 06/01/2021 à 9h37). At the bottom left is another 'Actions groupées' (Grouped actions) dropdown with an 'Appliquer' (Apply) button.

| Titre | Étiquettes | Date                        | Thumbnail |
|-------|------------|-----------------------------|-----------|
| test  | —          | Publié<br>06/01/2021 à 9h37 |           |

Par défaut, la colonne image fonctionne aussi sur TOUS les types de contenu personnalisés.

Dans nos fonctions de rappel utilisées pour ajouter notre colonne sur le tableau des articles, nous n'avons jamais vérifié le paramètre `$post_type` fourni ou le type du `$post`. On pourrait donc insérer un peu de logique dans cette fonction pour afficher notre colonne uniquement pour les articles par exemple.

Pour affecter uniquement les types de contenus spécifiques, on dispose aussi des hooks `manage_{$post_type}_posts_columns` pour ajouter des colonnes, et `manage_{$post_type}_posts_custom_column` pour le contenu des colonnes, où `$post_type` est l'identifiant du type de contenu, simplement.

```

add_filter( 'manage_wpcookbook-snippets_posts_columns', 'wpcookbook_snippets_posts_columns',
10, 1 );
/**
 * Registers a new column ONLY on wpcookbook-snippets
 *
 * @param array $columns Array of existing columns.
 * @return array $columns
 */
function wpcookbook_snippets_posts_columns( $columns ) {
    $columns['extra'] = __( 'Extra information', '33-tables' );
    return $columns;
}

add_action( 'manage_wpcookbook-snippets_posts_custom_column',
'wpcookbook_snippets_custom_column', 10, 2 );
/**
 * Displays our custom column data on wpcookbook-snippets table
 *
 * @param string $column_name Name of the current column
 * @param int $post_id Post id.
 */
function wpcookbook_snippets_custom_column( $column_name, $post_id ) {
    if( 'extra' === $column_name ){
        _e( 'Display any extra information here.', '33-tables' );
    }
}

```

Ici, on ajoute uniquement notre colonne supplémentaire sur les tableaux des contenus de type wpcookbook-snippets.

| Titre | Étiquettes | Date                        | Thumbnail | Extra information                   |
|-------|------------|-----------------------------|-----------|-------------------------------------|
| test  | —          | Publié<br>06/01/2021 à 9h37 |           | Display any extra information here. |

Notre nouvelle colonne apparaît ici, mais pas sur les listes des pages ou articles.

## Les commentaires

Pour les pages de commentaires, on ne dispose bizarrement pas d'un hook du type `manage_comments_columns` pour ajouter une colonne dans le tableau. Mais on peut tout de même en ajouter un, grâce au hook `manage_{$screen->id}_columns`, où `$screen_id` est le slug de la page en question sur laquelle il y a un tableau. Ce qui implique que potentiellement, tous les tableaux sont filtrables de cette façon ! Par contre, on dispose bien d'un hook `manage_comments_custom_column` pour le contenu des colonnes.

Imaginons que sur le devant du site, on dispose d'une fonctionnalité permettant de "liker" un commentaire, et qu'on dispose d'une métadonnée qui compte ce nombre de likes. On pourrait l'afficher dans les tableaux d'administration comme ceci :

```
add_filter( 'manage_edit-comments_columns', 'wpcookbook_comments_columns', 10, 1 );
/**
 * Adds a custom column to comments screen.
 *
 * @param array $columns Default columns
 * @return array $columns
 */
function wpcookbook_comments_columns( $columns ){
    $columns['likes'] = __( 'Comment likes', '33-tables' );
    return $columns;
}

add_action( 'manage_comments_custom_column', 'wpcookbook_comments_custom_column', 10, 2 );
/**
 * Adds content for our custom columns
 *
 * @param string $column_name Name of the current column
 * @param int     $comment_id Comment ID.
 */
function wpcookbook_comments_custom_column( $column_name, $comment_id ){
    if( 'likes' === $column_name ){
        $likes = get_comment_meta( $comment_id, 'comment_likes', true );
        echo (int) $likes;
    }
}
```

Les tables des commentaires aussi sont personnalisables.

## Modifier l'ordre des colonnes

Modifier l'ordre des colonnes se fait simplement sur le même hook que celui sur lequel on ajoute nos colonnes. Puisqu'on dispose d'un tableau contenant l'ensemble des colonnes dans notre fonction de rappel, on peut manipuler ce tableau comme bon nous semble.

```
add_filter( 'manage_posts_columns', 'wpcookbook_posts_columns', 10, 2 );
add_filter( 'manage_pages_columns', 'wpcookbook_posts_columns', 10, 1 );
/**
 * Registers a new column
 *
 * @param array $columns Array of existing columns.
 * @param string $post_type 'post' on posts page.
 * @return array $columns
 */
function wpcookbook_posts_columns( $columns, $post_type = 'page' ) {
    $position = array_search( 'title', array_keys( $columns ) ) + 1;
    $thumbnail = array( 'thumbnail' => __( 'Thumbnail', '33-tables' ) );
    $columns = array_merge(
        array_slice( $columns, 0, $position ),
        $thumbnail,
        array_slice( $columns, $position )
    );
    return $columns;
}
```

Ici, on veut insérer notre colonne `Thumbnail` juste après la colonne Titre. Donc, on repère la position de la colonne `title` dans le tableau `$columns`, et on découpe ce tableau en deux à cette position grâce à `array_slice`, pour y glisser notre colonne.

Attention cependant, car si vous utilisez une extension qui modifie les tableaux, votre fonction pourrait ne pas trouver la colonne que vous cherchez. Pour éviter cela, on peut aussi raisonner en terme d'index, c'est-à-dire, ajoutez une colonne en n-ième position, plutôt qu'après telle ou telle colonne spécifique qui pourrait ne pas être présente.

Notre colonne est bien déplacée.

## Rendre une colonne triable

Dans les divers tableaux, vous avez sans doute remarqué qu'on peut aussi trier le contenu selon certaines colonnes, mais que toutes les colonnes ne sont pas triables. Nos nouvelles colonnes ne le sont pas, par exemple.

Sur notre page d'administration des commentaires, cela peut être utile de pouvoir trier les commentaires en fonction de leur nombre de likes. Pour ça, on utilise le hook `manage_{$screen_id}_sortable_columns`, où `$screen_id` est le slug de la page, comme pour le hook utilisé pour ajouter notre colonne.

```
add_filter( 'manage_edit-comments_sortable_columns', 'wpcookbook_comments_sortable_columns',
10, 1 );
/**
 * Adds our 'likes' tabel to the list of sortable columns
 *
 * @param array $sortable_columns Array of default sortable columns
 * @return array $sortable_columns
 */
function wpcookbook_comments_sortable_columns( $sortable_columns ) {
    $sortable_columns['likes'] = 'likes';
    return $sortable_columns;
}
```

Le hook passe un seul paramètre à notre fonction, qui est un tableau des colonnes filtrables. Les clés sont les noms des colonnes, et les valeurs sont les paramètres `orderby` qui vont être passés à la requête de contenu.

Dans notre exemple, on indique que la colonne `likes` est filtrable, et que le paramètre `orderby` à utiliser pour la requête est `likes`.

Si vous cliquez sur le titre de la colonne pour trier, vous pouvez voir les paramètres de requête utilisés dans l'URL. Dans mon cas, cela donne <https://wpcookbook.local/wp-admin/edit-comments.php>

`orderby=likes&order=asc`. Si vous changez la valeur `orderby` associée à notre colonne, vous pourrez voir que le paramètre utilisé dans l'URL change aussi. Utiliser `$sortable_columns['likes'] = 'toto'`; dans notre fonction va donner <https://wpcookbook.local/wp-admin/edit-comments.php?orderby=toto&order=asc>.

Mais pour le moment, notre tri ne fonctionne pas car WordPress ne sait pas comment interpréter la valeur `likes` pour le paramètre `orderby`. Il faut donc lui expliquer.

Les arguments de requête pour les commentaires sont filtrables via le hook `comments_list_table_query_args`. On peut donc y greffer notre logique.

```
add_filter( 'comments_list_table_query_args', 'wpcookbook_comments_query_args', 10, 1 );
/**
 * Filters the list of comments to allow for ordering by comment_likes
 *
 * @param array $args Query arguments passed to get_comments()
 * @return array $args
 */
function wpcookbook_comments_query_args( $args ){
    if( is_admin() ){
        $screen = get_current_screen();
        if( 'edit-comments' === $screen->base && 'likes' === $args['orderby'] ){
            $args['orderby'] = 'meta_value_num';
            $args['meta_key'] = 'comment_likes';
        }
    }
    return $args;
}
```

Ici, on vérifie que l'on est bien sur le bon écran avant de filtrer, et on le fait en indiquant qu'il faut trier par valeur numérique (`meta_value_num`) de la métadonnée ayant pour clé `comment_likes`. On pourrait aussi définir une logique plus complexe avec une `meta_query`.

Le fonctionnement de la requête est similaire à une celui d'une `WP_Query` que l'on a étudié précédemment au chapitre Comprendre la boucle de WordPress. Je vous invite à consulter la documentation sur les `WP_Comment_Query` ([https://developer.wordpress.org/reference/classes/wp\\_comment\\_query/](https://developer.wordpress.org/reference/classes/wp_comment_query/)).

The screenshot shows the WordPress dashboard with the 'Commentaires' (Comments) page selected. The sidebar on the left has a dark theme with various menu items like 'Tableau de bord', 'Articles', 'Médias', 'Pages', 'Commentaires' (which is highlighted in blue), 'Extraits de code', 'Apparence', 'Extensions', 'Comptes', 'Outils', 'Réglages', and 'Réduire le menu'. The main content area is titled 'Commentaires' and shows a list of 4 comments. The columns are 'Auteur/autrice', 'Commentaire', 'En réponse à', 'Envoyé le', and 'Comment likes'. The first comment is from 'vincent' with the text 'This reply is liked a lot!', showing 4 likes. The second comment is from 'A WordPress Commenter' with the text 'Hi, this is a comment. To get started with moderating, editing, and deleting comments, please visit the Comments screen in the dashboard. Commenter avatars come from Gravatar.', also showing 4 likes. The third comment is from 'vincent' with the text 'This is an example reply', showing 1 like. The fourth comment is from 'Auteur/autrice' with the text 'Commentaire', showing 0 likes. At the bottom, there are buttons for 'Actions groupées' and 'Appliquer'.

On peut maintenant trier les commentaires par likes.

Pour cet exemple, j'ai artificiellement donné des valeurs de likes aux commentaires, à l'aide de `update_comment_meta( $comment_id, 'comment_likes', $value )`.

## Rendre une colonne du tableau des articles ou pages filtrable

Pour les tableaux des articles, on peut procéder exactement de la même manière pour rendre une colonne filtrable. On ajoute la colonne filtrable et le paramètre `orderby` en se greffant sur le hook `manage_edit-post_sortable_columns` (`edit-post` étant l'identifiant de l'écran), et on intercepte les paramètres de la requête en se greffant sur `pre_get_posts`.

```

add_filter( 'manage_edit-post_sortable_columns', 'wpcookbook_posts_sortable_columns', 10, 1 );
/**
 * Adds our 'likes' table to the list of sortable columns
 *
 * @param array $sortable_columns Array of default sortable columns
 * @return array $sortable_columns
 */
function wpcookbook_posts_sortable_columns( $sortable_columns ){
    $sortable_columns['likes'] = 'likes';
    return $sortable_columns;
}

add_action( 'pre_get_posts', 'wpcookbook_posts_orderby', 10, 1 );
/**
 * Updates the posts query in the admin
 *
 * @param WP_Query $query
 */
function wpcookbook_posts_orderby( $query ) {
    if( ! is_admin() || ! $query->is_main_query() ) {
        return;
    }

    $screen = get_current_screen();
    if( 'edit-post' === $screen->id && 'likes' === $query->get( 'orderby' ) ){
        $query->set( 'orderby', 'meta_value' );
        $query->set( 'meta_key', 'post_likes' );
        $query->set( 'meta_type', 'numeric' );
    }
}

```

Pour cet exemple, j'ai aussi ajouté une colonne `likes` sur nos articles, affichant la metadonnée `post_likes`. Et vu qu'il n'y a pas d'interface sur le devant du site pour modifier cette metadonnée, je l'ai artificiellement attribuée pour chaque article à l'aide de `update_post_meta( $post_id, 'post_likes', rand( 1, 100 ) )`.

## Ajouter une action pour les articles.

Dans le tableau des articles, on peut aussi ajouter des liens d'actions juste sous les titres, parmi les liens Modifier ou Supprimer. Il suffit de se hooker sur `post_row_actions`. Le filtre nous fournit deux paramètres `$actions` et `$post`. Le premier contient la liste des actions déjà enregistrées, et le second l'objet `post` en question.

```
add_filter( 'post_row_actions', 'wpcookbook_post_row_actions', 10, 2 );
/**
 * Adds new actions on the post page.
 *
 * @param array $actions List of available row actions.
 * @param WP_Post $post Current post
 * @return array $actions
 */
function wpcookbook_post_row_actions( $actions, $post ) {
    $duplicate_url = add_query_arg( array(
        'action' => 'duplicate',
        'post_id' => (int) $post->ID,
        '_wpnonce' => wp_create_nonce( 'duplicate-post-' . (int) $post->ID ),
    ), admin_url( 'admin-post.php' ) );

    $actions['duplicate'] = sprintf(
        '<a href="%s">%s</a>',
        esc_url( $duplicate_url ),
        esc_html__( 'Duplicate post', '33-tables' )
    );
    return $actions;
}
```

Dans cet exemple, on va créer un lien pour dupliquer un article. On va simplement traiter cette action en pointant le lien vers `admin-post.php`, exactement comme au chapitre Comment traiter des données de formulaires. Sur cette url, à l'aide de `add_query_arg`, on y ajoute notre action, l'identifiant du contenu à dupliquer, et un nonce créé avec l'action correspondante. Ensuite, on ajoute notre lien au tableau des actions sous la clé `duplicate`, et on retourne le tableau des actions.

The screenshot shows the WordPress dashboard with the 'Articles' section selected. The left sidebar shows 'Tous les articles' and other navigation options like 'Ajouter', 'Catégories', and 'Étiquettes'. The main area displays a list of articles with columns for 'Titre', 'Thumbnail', 'Auteur/autrice', and 'Catégories'. For the article 'Hello everyone', there is an additional column labeled 'Actions' containing links: 'Modifier', 'Modification rapide', 'Corbeille', 'Afficher', and 'Duplicate post'. The 'Duplicate post' link is highlighted, indicating it has been added.

Notre lien est bien ajouté.

Mais si on clique sur notre lien, on aura une page blanche sur admin-post.php. Il faut maintenant implémenter le traitement de notre action.

Pour cette action, l'utilisateur doit absolument être connecté, et on veut déclencher notre traitement uniquement pour cette action, donc on va se hooker sur admin\_post\_duplicate.

```
add_action( 'admin_post_duplicate', 'wpcookbook_handle_row_actions' );
/**
 * Handles our new post action link
 */
function wpcookbook_handle_row_actions(){
    $post_id = ! empty( $_GET['post_id'] ) ? (int) $_GET['post_id'] : false;
    $referer = wp_get_referer();
    check_admin_referer( 'duplicate-post-' . $post_id );

    if( ! $post_id ){
        $redirect_url = add_query_arg( 'error', 'invalid-id', $referer );
        wp_safe_redirect( $redirect_url );
        exit;
    }

    if( ! current_user_can( 'edit_post', $post_id ) ){
        $redirect_url = add_query_arg( 'error', 'unauthorized', $referer );
        wp_safe_redirect( $redirect_url );
        exit;
    }

    // Try to fetch the post
    $post = get_post( $post_id, 'ARRAY_A' );

    if( ! $post ){
        $redirect_url = add_query_arg( 'error', 'no-post', $referer );
        wp_safe_redirect( $redirect_url );
        exit;
    }

    unset( $post['ID'], $post['guid'] );
    $new_post = wp_insert_post( $post );

    if( ! $new_post ){
        $redirect_url = add_query_arg( 'error', 'failed', $referer );
        wp_safe_redirect( $redirect_url );
        exit;
    }

    $redirect_url = add_query_arg( 'success', 'true', $referer );
    wp_safe_redirect( $redirect_url );
    exit;
}
```

Notre action et notre nonce seront vérifiés par `check_admin_referer()`. La fonction `check_admin_referer( int|string $action = -1, string $query_arg = '_wpnonce' )` prend deux paramètres : une action `$action`, et une nonce `$nonce`. L'action passée en paramètre doit correspondre au paramètre `action` de la requête. Ici, cela correspond au paramètre d'URL du lien. Le second argument est le nom du nonce que l'on a donné. Il vaut `_wpnonce` par défaut, et c'est ce nom que

l'on a aussi donné au nonce sur notre lien.

Puis on vérifie si l'identifiant de l'article est valide. Si non, on redirige vers la page d'où vient l'utilisateur, en ajoutant un paramètre d'url error. Je vous laisse implémenter la petite notice du message d'erreur en vous aidant du chapitre Comment ajouter des notices dans l'administration de WordPress ok ?

Ensuite, on essaie d'aller chercher le contenu en question à l'aide de `get_post()` en lui passant `ARRAY_A` comme second argument pour que la fonction me retourne l'article sous forme de tableau associatif. En cas de souci, on redirige.

Quand on a notre article dans son tableau associatif, on lui enlève son identifiant et son `guid`, et on le passe dans `wp_insert_post()` pour créer un nouvel article. Si on lui avait laissé son identifiant, alors `wp_insert_post()` aurait simplement mis à jour l'article original.

Enfin, on redirige vers la page d'où l'utilisateur venait, avec un petit paramètre pour lui indiquer que tout va bien.

La fonctionnalité est toute simple et directe, mais cela fonctionne. Attention par contre, elle ne copie pas les `post_meta` et autres données ! C'est juste une implémentation très basique pour l'exemple !

| Titre          | Thumbnail | Auteur/autrice | Catégories | Étiquettes | Date                      | Likes |
|----------------|-----------|----------------|------------|------------|---------------------------|-------|
| Hello everyone |           | vincent        | JavaScript | —          | Publié 15/01/2020 à 16h18 | 71    |
| Hello everyone |           | vincent        | JavaScript | —          | Publié 15/01/2020 à 16h18 | 0     |

L'article est bien dupliqué.

Remarquez que l'article a le même titre, le même slug suffixé d'un -2, et pas d'image mise en avant. Je vous laisse vous amuser pour compléter cette fonctionnalité si vous le souhaitez !

Pour les types de contenu hiérarchiques, comme les pages, le hook à utiliser est `page_row_actions`, simplement.

## Ajouter des actions de masse (bulk actions)

Pour les actions de masse, le procédé est le même, sauf que ces actions sont traitées sur la même page et donc il y a un processus défini sur lequel se greffer.

Pour ajouter notre bulk action, on se hooke sur `bulk_actions-$screen_id`, et on ajoute notre action parmi celles fournies à la fonction de rappel. `$screen_id` correspond à l'identifiant de l'écran sur lequel on veut ajouter notre action. Pour l'exemple, on va ajouter une action sur la page listant les articles.

```

add_filter( 'bulk_actions-edit-post', 'wpcookbook_posts_bulk_actions' );
/**
 * Adds a bulk action
 *
 * @param array $bulk_actions Bulk actions already registered
 * @return array $bulk_actions
 */
function wpcookbook_posts_bulk_actions( $bulk_actions ) {
    $bulk_actions['bulk_duplicate'] = __( 'Duplicate', '33-tables' );
    return $bulk_actions;
}

```

La fonction prend un seul paramètre, qui est un tableau des actions déjà enregistrées. Il suffit de lui ajouter un élément avec un identifiant d'action pour clé et un label associé.

Pour traiter notre formulaire, on utilise le hook `handle_bulk_actions-$screen_id`. Attention, c'est un filtre et pas une action ! La fonction de rappel reçoit en premier paramètre l'url vers laquelle l'utilisateur va être redirigé, puis l'action en cours, et enfin un tableau contenant les IDs des éléments sélectionnés pour cette action.

```

add_filter( 'handle_bulk_actions-edit-post', 'wpcookbook_handle_bulk_duplicate', 10, 3 );
/**
 * Handles our bulk duplicate action
 *
 * @param string $redirect_to URL to redirect to after bulk action processing
 * @param string $doaction Current bulk action
 * @param array $items Array of post ids to process
 * @return string $redirect_to
 */
function wpcookbook_handle_bulk_duplicate( $redirect_to, $doaction, $items ) {
    if ( $doaction !== 'bulk_duplicate' ) {
        return $redirect_to;
    }
    foreach ( $items as $post_id ) {
        $post = get_post( $post_id, 'ARRAY_A' );
        unset($post['ID'], $post['guid']);
        $new_post = wp_insert_post( $post );
    }
    $redirect_to = add_query_arg( 'bulk-duplicate', 'success', $redirect_to );
    return $redirect_to;
}

```

Ici, on vérifie simplement que l'on est bien sur la bonne action. Puis on parcourt le tableau `$items` qui contient les identifiants des articles à dupliquer, et on applique la même logique que pour notre action précédente. En cas de succès, on ajoute un petit paramètre à l'url de redirection et on la retourne. Par contre, pour garder l'exemple simple, je ne traite aucune erreur. Je vous laisse améliorer ce petit bout de code, l'objectif étant simplement de vous montrer où vous hooker.

The screenshot shows the WordPress admin interface for the 'Articles' section. On the left, a sidebar lists various menu items. The 'Articles' item is currently selected. The main area displays a list of posts with columns for 'Thumbnail', 'Auteur/autrice', and 'Catégories'. Two posts are selected: 'Hello everyone' and 'Hello Galaxy'. A context menu is open over the first post, with 'Duplicate' highlighted. Other options in the menu include 'Actions groupées', 'Modifier', and 'Mettre à la corbeille'.

Notre action fonctionne.

## Ajouter des filtres

Par défaut, à côté des actions groupées, WordPress nous permet de filtrer par date et par catégorie. Il est aussi possible d'ajouter ici nos propres filtres.

Pour cela, on se hooke sur `restrict_manage_posts`. La fonction de rappel hookée prends deux paramètres : `$post_type` et `$which`. `$post_type` est assez explicite et correspond au type de contenu sur lequel on se situe. `$which` est une chaîne de caractère représentant quelle barre de filtre on veut modifier. Il peut valoir 'top' ou 'bottom'. Sur les pages de listing des médias, ce sera 'bar'.

Dans notre cas, `$which` vaudra forcément 'top' car la barre de filtre se trouve en haut. Mais vous pouvez ajouter toute sorte de contrôles et boutons supplémentaires en vous hookant sur `manage_posts_extra_tablenav`. La fonction de rappel prendra un seul paramètre `$which` et vous pourrez y insérer le contrôle que vous souhaitez.

```

add_action( 'restrict_manage_posts', 'wpcookbook_likes_filter', 10, 2 );
/**
 * Adds a filter to our post page
 *
 * @param string $post_type The post type
 * @param string $which The location of the filters currently being displayed. Can be 'top'
 * or 'bottom' (or 'bar' for media list)
 */
function wpcookbook_likes_filter( $post_type, $which ) {
    if( 'post' === $post_type && 'top' === $which ){
        $choices = array(
            '50' => __( '50 likes or more', '33-tables' ),
            '100' => __( '100 likes or more', '33-tables' )
        );
        $options = sprintf( '<option value="">%s</option>', esc_html__( 'Any number of likes', '33-tables' ) );
        foreach ( $choices as $value => $label ) {
            $options .= sprintf( '<option value="%s">%s</option>', esc_attr( $value ), esc_html( $label ) );
        }
        $select = sprintf(
            '<label for="likes" class="screen-reader-text">%s</label>
            <select id="likes" name="likes">%s</select>',
            esc_html__( 'Filter by number of likes', '33-tables' ),
            $options
        );
        echo $select;
    }
}

```

Dans notre fonction, on vérifie simplement que l'on est sur la bonne page et barre de filtres, puis on insère un champ de type `<select>`. La valeur de ce champ sera directement envoyée quand l'utilisateur soumettra le formulaire à l'aide du bouton Filtrer.

| Titre          | Thumbnail | Auteur/autrice | Catégories | Étiquettes | Date                      | Likes |
|----------------|-----------|----------------|------------|------------|---------------------------|-------|
| Hello everyone |           | vincent        | JavaScript | —          | Publié 15/01/2020 à 16h18 | 71    |

Notre filtre apparaît bien avant le bouton Filtrer.

La valeur est bien envoyée et vous pouvez le voir dans les paramètres d'URL, mais le filtre ne fait rien pour le moment. Il faut, comme pour le tri, se hooker sur `pre_get_posts` pour venir ajuster la requête juste avant qu'elle ne soit faite.

```

add_action( 'pre_get_posts', 'wp_cookbook_filter_admin_query', 10, 1 );
/**
 * Use our filter value to filter the posts
 *
 * @param WP_Query $query The WP Query
 */
function wp_cookbook_filter_admin_query( $query ){
    $screen = get_current_screen();
    if( ! is_admin() || ! $query->is_main_query() || ( 'edit-post' !== $screen->id ) ){
        return;
    }

    $likes = ! empty( $_GET['likes'] ) ? (int) $_GET['likes'] : null;
    if( ! $likes ){
        return;
    }

    $query->set( 'meta_query', array(
        array(
            'key'      => 'post_likes',
            'value'    => $likes,
            'compare' => '>=',
            'type'     => 'NUMERIC',
        )
    ) );
}

```

On vérifie s'il y a une valeur pour notre filtre, et si oui, on va chercher uniquement les articles ayant plus de likes que la valeur de notre filtre. Pour rappel, nous avions stockés le nombre de likes dans une métadonnée. Il convient donc d'ajouter une `meta_query` à la requête.

## Ce qu'il faut retenir

Les tableaux par défaut regorgent de hooks sur lesquels on peut se greffer pour y ajouter nos éléments d'interface. On n'a pas étudié tous les hooks disponibles, mais pour 95% des besoins, vous savez comment maintenant comment faire !

- Pour ajouter une colonne, on se greffe sur les hooks `manage_posts_columns`, `manage_pages_columns` pour les pages (attention au nombre de paramètres !), ou `manage_${post_type}_columns` pour les types de contenu personnalisés. Attention car `manage_posts_columns` se déclenche sur tous les types de contenu non-hierarchiques comme les articles. Plus généralement, on peut aussi se hooker sur `manage_${screen_id}_columns` pour ajouter des colonnes, où `screen_id` est l'identifiant de l'écran sur lequel il y a un tableau.
- Pour ajouter du contenu dans nos colonnes, on utilise `manage_posts_custom_column`, `manage_pages_custom_column`, `manage_comments_custom_column` ou `manage_${post_type}_posts_custom_column`.
- Modifier l'ordre des colonnes se fait sur le même hook que pour déclarer les colonnes. Puisqu'on a accès à un tableau regroupant toutes les colonnes, on peut le modifier directement.
- Pour rendre les colonnes triables, le hook à utiliser est `manage_${screen_id}_sortable_columns`. Ensuite, il faut se hooker sur `pre_get_post` pour modifier la requête pour qu'elle prenne en compte le nouveau paramètre d'URL. Pour les

commentaires, c'est sur `comments_list_table_query_args`.

- Ajouter un lien dans les actions sous les titres des éléments se fait simplement à l'aide du hook `post_row_actions` ou `page_row_actions`. A nous de gérer l'action liée à ce lien via `admin-post.php`, par exemple.
- Ajouter une action de groupe se fait sur `bulk_actions-${screen_id}`, et la traiter se fait sur `handle_bulk_actions-${screen_id}`.
- Ajouter un filtre se fait sur `restrict_manage_posts`. Attention, car ce hook se déclenche sur les tableaux des articles, pages et types de contenu personnalisés. Il faut donc vérifier le type de contenu pour éviter d'afficher le filtre sur les pages non désirées. Comme pour les actions, il faut traiter notre nouveau filtre via `pre_get_posts`.
- On peut ajouter toute sorte de contrôles personnalisés avant et après notre tableau en se hookant sur `manage_posts_extra_tablenav`.

Les tableaux natifs n'ont (presque) plus de secrets pour vous maintenant !

## That's a wrap !

Vous voici à la conclusion de ce livre ! Félicitations !

Vous avez maintenant toutes les bases nécessaires pour poursuivre votre apprentissage du développement pour WordPress :

- Vous comprenez comment fonctionnent les hooks, savez les trouver et en ajouter à vos développements.
- Vous avez une vue d'ensemble de comment WordPress fonctionne en interne.
- Vous savez internationaliser vos développements.
- Vous avez les notions indispensables pour sécuriser vos développements.
- Vous comprenez comment fonctionnent les thèmes, les thèmes enfants et les modèles de pages
- Vous savez déclarer des zones de widgets et des emplacements de menu.
- Vous savez personnaliser vos menus
- Vous savez faire vos propres requêtes de contenu.
- Vous savez gérer les rôles et capacités de WordPress.
- Vous savez créer des réglages pour vos thèmes et vos extensions.
- Vous savez manipuler les types de contenus personnalisés, et les taxonomies personnalisées.
- Vous savez traiter tout type de formulaire.
- Vous savez créer et manipuler les utilisateurs.
- Vous savez programmer vos actions récurrentes.
- Vous savez afficher des notices.
- Vous savez stocker des données temporaires en cache ou base, via les transients de WordPress
- Vous savez personnaliser les tableaux dans l'administration
- Vous savez faire tout ça de façon sécurisée !

C'est énorme ! Encore bravo !

Pourtant, ce n'est que le début de votre apprentissage. Il y a plein d'aspects sur lesquels on n'a pas insisté. D'autres dont on n'a même pas parlé (API REST ? Le nouvel éditeur ?).

WordPress est une bécane vieille de 15 ans. Un seul tome de 450 pages ne suffit pas pour en détailler tout le fonctionnement. Moi-même j'ai appris plein de choses en écrivant ce livre, en creusant dans la documentation de WordPress ou dans son code source. Et j'en apprends encore tous les jours quand je travaille sur les sites de mes clients ! Parfois, leur besoin me demande de la recherche et je déterre quelques gemmes pendant le process ! C'est magique !

J'espère vraiment que ce guide vous a été utile, que vous avez appris plein de choses, et que vous avez gagné la confiance nécessaire pour vous lancer ! Allez-y ! Faites votre thème perso ou une petite extension ! Publiez-la sur le répertoire officiel ! Peu importe le temps que cela prendra, faites ça bien, selon les standards de WordPress détaillés dans ce guide, de façon sécurisée et en pensant d'abord à l'utilisateur et son expérience.

Si vous avez la moindre question ou remarque ou si vous avez une suggestion à faire, n'hésitez pas à m'envoyer un email à [vincent@vincentdubroeucq.com](mailto:vincent@vincentdubroeucq.com). Je lis tous mes mails et je réponds à tout le monde. Pas toujours du tac au tac, mais bon.

Je suis toujours ok pour recevoir du feedback, des corrections de typo (il y en aura !), ou des suggestions de nouvelles recettes. Qui sait, peut-être que j'aurais assez d'idées pour un tome 2 !

Si ce guide vous a aidé, un petit tweet est bienvenu ! Cela m'aiderait beaucoup ! Et s'il vous a été utile, il le

sera peut-être pour d'autres ! Aussi, n'hésitez pas à m'envoyer un mail et me laisser un petit témoignage pour m'expliquer en quoi ce livre vous a aidé !

Merci encore pour votre lecture et votre soutien !

Lancez-vous !

## Thank you !

Je voudrais remercier mon épouse Adélaïde, qui m'a supporté — au sens français et anglais du terme (support = soutenir) — pendant l'écriture de ce guide, qui m'a relu, corrigé, conseillé sur le ton, conseillé sur chaque mail que j'ai écrit pour mes abonnés, qui m'a encouragé quand je recevais un feedback positif et que mes yeux brillaient comme un gamin emerveillé, qui a toléré que je travaille (parfois beaucoup) plus chaque jour y compris certains weekends, et bien plus. Merci à mes enfants aussi !

Un grand merci à Julio Potier <https://secupress.fr> pour m'avoir soutenu et encouragé dès que je lui annoncé mon idée de livre ! À fond dedans, direct. Merci pour ses conseils ultra pertinents, ses retweets, toute la pub et les recommandations pour ce guide qu'il a pu faire, sa relecture technique détaillée et ses propositions d'améliorations ! Merci pour ses questions qui m'ont fait repenser des pans entiers de certains de mes chapitres ! À minuit, évidemment. Me donner de bonnes idées à 9h du matin ce serait trop facile !

Merci à toutes les personnes avec lesquelles j'ai échangé durant l'écriture du livre et qui m'ont aidé d'une manière ou d'une autre, feedback, retweets, etc. Merci à Alex Bortolotti de [WPMarmite](#) pour les discussions très enrichissantes, la relecture et les suggestions pertinentes ! Merci à Lycia Diaz (<https://lycia-diaz.com/>) pour nos échanges très intéressants et son enthousiasme pour le projet ! Merci à Daniel Roch (<https://www.seomix.fr>) pour les discussions et remarques pertinentes, à Maxime Bernard-Jacquet (<https://capitainewp.io>) pour son feedback, à Sylvain Dorémus (<https://www.dragonjoker.org>) pour la relecture, à Jérémie Allard (<https://jeremy-allard.com>) pour son soutien et ses conseils marketing avisés !

Merci à toutes les personnes qui ont participé à la prévente ! Sans vous je ne serai peut-être pas allé au bout ! Merci pour votre soutien depuis le début, pour les encouragements, l'enthousiasme et le feedback reçu par mail, pour vos retweets, pour vos questions qui m'ont aidé à réorganiser mon contenu mais aussi ne rien oublier d'évident, car mon premier plan avait quelques manques flagrants !

Un grand merci à tous les abonnés à ma newsletter qui me lisent régulièrement et me posent des questions, ainsi qu'à ceux qui me suivent sur les réseaux sociaux, Twitter et Linkedin principalement, et qui partagent mon contenu !

Et merci à toi, lecteur, de m'avoir fait confiance pour t'enseigner comment développer pour WordPress ! J'espère que la dégustation a été agréable, et que tu t'es régale !

A tous, merci !

## L'auteur



C'est moi !

Je m'appelle Vincent Dubroeucq, je suis designer et développeur WordPress. Orateur, formateur et blogueur. Je développe pour WordPress depuis quelques années déjà en publiant des thèmes sur le répertoire officiel et en freelance pour les clients.

J'adore WordPress, je trouve que c'est un outil de publication formidable et d'une simplicité de prise en main inégalable. Mais il a aussi ses défauts, ses limites et ses pans de code pas très jolis. Mais que ce soit d'un point de vue utilisateur ou développeur, c'est un super outil qui offre des possibilités presque sans limite.

Il est extensible à l'infini et fournit une quantité d'outils internes impressionnante, répondant à énormément de besoins. Donc il revient à nous, développeurs, d'exploiter au mieux ces API internes pour profiter au maximum de WordPress et garantir la compatibilité et la sécurité de nos développements.

C'est ce que j'ai voulu transmettre à travers ce guide et ce que j'essaie de mettre en avant sur mon blog : restez simples et comprenez WordPress pour mieux exploiter tout son potentiel et vous simplifier la vie.

Je souhaite aider un maximum de développeurs qui souhaitent apprendre à développer pour WordPress à le faire de la meilleure façon possible, efficacement, rapidement et de façon sécurisée, en exploitant tout le potentiel des outils et APIs que WordPress met à notre disposition.

J'espère vous avoir convaincu !

Si vous voulez me joindre, pour une question ou pour faire coucou, vous pouvez m'envoyer un mail à [vincent@vincentdubroeucq.com](mailto:vincent@vincentdubroeucq.com), ou me suivre sur [Twitter](#).

Vous pouvez aussi vous abonner sur mon site <https://vincentdubroeucq.com> pour recevoir la newsletter !

N'hésitez pas !

Encore merci et à bientôt !

Vincent Dubroeucq

WPCookbook, Copyright 2021 Vincent Dubroeucq