Plug-in Timber pour WordPress

Templating avec Twigg

TIMBER C'EST QUOI ?

Disponible sous forme de plugin ou de composant à intégrer directement dans son thème via Composer, Timber va apporter une petite surcouche vous permettant d'utiliser du templating et bien séparer le code PHP de vos pages, du HTML.

Sous WordPress :

```php
<?php
  get_header();

  // case : display a category
  if (is_category()) {
        $title = "Catégorie &bullet; ".single_tag_title( '', false);
  }
  // case : display a tag
  elseif (is_tag()) {
        $title = "Mot clé &bullet; ".single_tag_title( '', false);
  }
  else {
        $title = 'Le Blog &bullet; dernières actus';
  }
?>

  <div class="section blog">
        <div class="wrapper flex-container">
            <div class="f3 blog__posts">
                <h1 class="blog__title"><?php echo $title; ?></h1>

                <?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
                    <div class="post">
                        <h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a>

                        <div class="post__meta">
                            <span class="post__date">Le <?php the_time('d F Y'); ?></span
                            <span class="post__categories">Thème : <?php the_category();
                            <span class="post__tags"> Tags : <?php the_tags('',' ',''); ?
                        </div>

                        <div class="post__thumbnail">
                            <a href="<?php the_permalink(); ?>"><?php the_post_thumbnail(
                        </div>
```

<= Classic

L'idée est de simplifier et améliorer la création des thèmes WP en accentuant bien la séparation entre la forme et les fonctionnalités. On a donc plus ce mélange entre HTML et PHP et des fonctionnalités à l'instar du Rails ou du Node par exemple.

Donc on a d'un côté des fichiers PHP contenant les classes PHP du thème et de l'autre des fichiers twig rédigés avec le markup Timber et les langages de balisage traditionnels tels que le HTML, le CSS, etc.

LE THEMING AVEC TIMBER

Une fois Timber en place, vos fichiers de template deviennent des Controlleurs : c'est à dire qu'ils ne gèrent que le code.

Votre HTML ira du coup dans un fichier de template au format .twig . Twig est un système de templating très connu et très simple à utiliser.

Plus de PHP du coup dans les templates ! les variables sont appelées de cette forme {{variable}} :

```php
<?php

$context = Timber::get_context();

$template = 'archive.twig';

// Define title
if (is_tag()) {
  $context['title'] = single_tag_title( '', false );
}
elseif (is_category()  {
  $context['title'] = 'Catégorie &bullet; '.single_cat_title( '', false );
}
elseif(is_post_type_archive()) {
  $context['title'] = post_type_archive_title( '', false );
  $template = 'archive-' . get_post_type() . '.twig';
}
else {
  $context['title'] = 'Le Blog';
}

// Pagination
$context['pagination'] = Timber::get_pagination();

// Posts
$context['posts'] = Timber::get_posts();


Timber::render($template, $context);
```

```twig
{% extends "base.twig" %}

{% block content %}
  <div class="section blog">
    <div class="wrapper flex-container">
      <main class="blog__main">
        <h1 class="blog__title">{{title}}</h1>

        <div class="blog__posts">
        {% for post in posts %}
          {% include 'teaser.twig' %}
        {% endfor %}
        </div> <!-- blog__posts -->

        {% include 'navigation.twig' %}

      </main> <!-- blog__main -->

      <div class="blog__sidebar">
        {{sidebar}}
      </div> <!-- blog__sidebar -->

    </div> <!-- wrapper -->
  </div>
{% endblock %}
```

Les templates vont nous permettre de séparer le code PHP du code HTML/XML/Text, etc.

Seulement, pour faire du HTML de présentation, on a toujours besoin d'un peu de code dynamique : faire une boucle pour afficher toutes les annonces de notre plateforme,

créer des conditions pour afficher un menu différent pour les utilisateurs authentifiés ou non, etc.

Pour faciliter ce code dynamique dans les templates, le moteur de templates Twig offre son pseudo-langage à lui.

La syntaxe est plus concise et plus claire. Pour afficher une variable,{{ mavar }} suffit, alors qu'en PHP il faudrait écrire <?php echo $mavar; ?>.

Il y a quelques fonctionnalités en plus, comme l'héritage de templates (nous le verrons). Cela serait bien entendu possible en PHP, mais il faudrait coder soi-même le système et cela ne serait pas aussi esthétique.

Il sécurise vos variables automatiquement : plus besoin de se soucier de htmlentities(),addslashes()

Description	Exemple Twig	Équivalent PHP
Afficher une variable	`Pseudo : {{ pseudo }}`	`Pseudo : <?php echo $pseudo; ?>`
Afficher l'index d'un tableau	`Identifiant : {{ user['id'] }}`	`Identifiant : <?php echo $user['id']; ?>`
Afficher l'attribut d'un objet, dont le getter respecte la convention `$objet->getAttribut()`	`Identifiant : {{ user.id }}`	`Identifiant : <?php echo $user->getId(); ?>`
Afficher une variable en lui appliquant un filtre. Ici, « upper » met tout en majuscules :	`Pseudo en majuscules : {{ pseudo|upper }}`	`Pseudo en lettre majuscules : <?php echo strtoupper($pseudo); ?>`
Afficher une variable en combinant les filtres. « striptags » supprime les balises HTML. « title » met la première lettre de chaque mot en majuscule. Notez l'ordre d'application des filtres, ici `striptags` est appliqué, puis `title`.	`Message : {{ news.texte|striptags|title }}`	`Message : <?php echo ucwords(strip_tags($news->getTexte())); ?>`
Utiliser un filtre avec des arguments. Attention, il faut que `date` soit un objet de type `Datetime` ici.	`Date : {{ date|date('d/m/Y') }}`	`Date : <?php echo $date->format('d/m/Y'); ?>`

Les filtres utiles :

Filtre	Description	Exemple Twig
`upper`	Met toutes les lettres en majuscules.	`{{ var|upper }}`
`striptags`	Supprime toutes les balises XML.	`{{ var|striptags }}`
`date`	Formate la date selon le format donné en argument. La variable en entrée doit être une instance de `Datetime`.	`{{ date|date('d/m/Y') }}` `Date d'aujourd'hui : {{ "now"|date('d/m/Y') }}`
`format`	Insère des variables dans un texte, équivalent à `printf`.	`{{ "Il y a %s pommes et %s poires"|format(153, nb_poires) }}`
`length`	Retourne le nombre d'éléments du tableau, ou le nombre de caractères d'une chaîne.	`Longueur de la variable : {{ texte|length }}` `Nombre d'éléments du tableau : {{ tableau|length }}`

Twig et la sécurité :

Dans tous les exemples précédents, vos variables ont déjà été protégées par Twig ! Twig applique par défaut un filtre sur toutes les variables que vous affichez, afin de les protéger de balises HTML malencontreuses.

Ainsi, si le pseudo d'un de vos membres contient un «<» par exemple, lorsque vous écrivez {{ pseudo }}celui-ci est échappé, et le texte généré est en réalité «mon<pseudo» au lieu de «mon<pseudo», ce qui poserait problème dans votre structure HTML.

Très pratique ! Et donc à savoir : inutile de protéger vos variables en amont, Twig s'occupe de tout en fin de chaîne !Et dans le cas où vous voulez afficher volontairement une variable qui contient du HTML (JavaScript, etc.), et que vous ne voulez pas que Twig l'échappe, il vous faut utiliser le filtre rawcomme ceci :
{{ ma_variable_html|raw }}.

Avec ce filtre, Twig désactive localement la protection HTML, et affiche la variable en brut, quel que soit ce qu'elle contient.

Installer via Composer :

```
cd /mon-site/wp-content/themes/mon-theme

composer require timber/timber
```

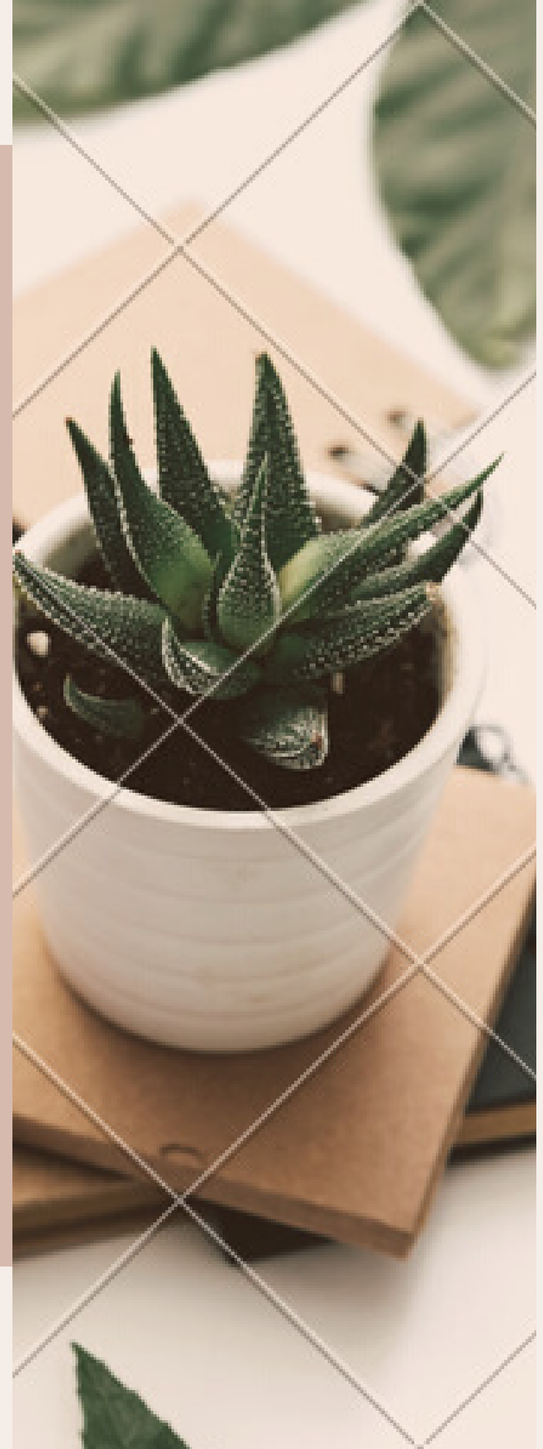
Une fois installé via composer vous verrez apparaitre un dossier vendor dans votre thème :

et vous n'aurez qu'à ajouter l'include vers composer dans votre functions.php ainsi qu'ajouter l'appel de la classe Timber.

```php
<?php

// Composer
require_once(__DIR__ . '/vendor/autoload.php');

// Timber
$timber = new \Timber\Timber();
```



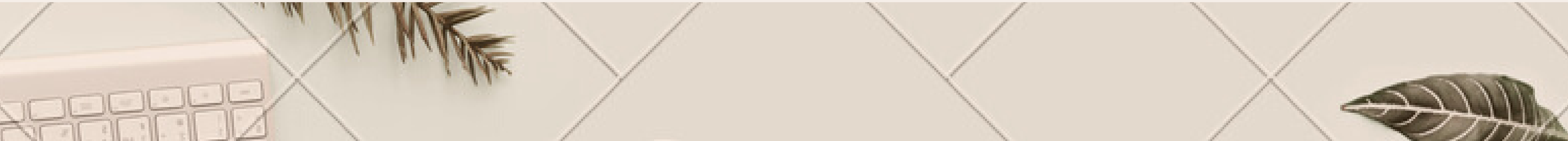
Comparaison des modèles Blade et Twig dans Laravel

Blade et Twig fournissent les fonctionnalités les plus importantes; héritage de modèle, sections, sortie d'échappement et syntaxe propre.

Blade fournit une syntaxe simple et rapide, mais n'ajoute pas (beaucoup) de fonctionnalités supplémentaires.

Twig va encore plus loin et ajoute une couche supplémentaire pour offrir plus de sécurité et des fonctionnalités supplémentaires.

Le choix dépend principalement de vos préférences personnelles. Si vous développez principalement pour Laravel, Blade serait probablement bon. Si vous utilisez également de nombreux autres frameworks, Twig pourrait être un meilleur choix.



Mais c'était sans compter sur la nouvelle version de Blade de Novembre 2019

Blade a reçu de nombreuses mises à jour depuis la rédaction de cet article il y a 4 ans.

Certaines différences subsistent, mais la principale raison de choisir Twig plutôt que Blade serait que vous / votre entreprise avez plus d'expérience dans Twig (en raison d'autres cadres).

Si vous utilisez principalement Laravel, je resterais avec Blade.

Blade est le moteur de template par défaut pour Laravel (depuis Laravel 2 en 2011). La syntaxe est à l'origine inspirée de la syntaxe ASP.net Razor et fournit un moyen plus propre d'écrire vos modèles.

Mais la syntaxe n'est qu'une partie, le principal avantage de l'utilisation de Blade au lieu de PHP simple est de faciliter la réutilisation des modèles et des modèles fractionnés.

Exemple Laravel :

```
@extends('layouts.master')

@section('content')
  @foreach ($users as $user)
      <p>This is user </p>
  @endforeach
@endsection
```

Fondamentalement, la syntaxe Blade est un wrapper mince pour PHP, pour fournir une syntaxe propre.

Tout ce que vous pouvez faire avec PHP, vous pouvez le faire avec Blade (et vice versa).

Vous pouvez également facilement mélanger du PHP simple dans vos modèles Blade.

Exemple Twigg:

```
{% extends "layouts.master" %}

{% block content %}
    {% for user in users %}
        <p>This is user {{ user.id }}</p>
    {% endfor %}
{% endblock %}
```

Twig est en bac à sable.

Vous ne pouvez pas utiliser n'importe quelle fonction PHP par défaut, vous ne pouvez pas accéder à des choses en dehors du contexte donné et vous ne pouvez pas utiliser du PHP simple dans vos modèles.

Cela se fait par conception, cela vous oblige à séparer votre logique métier de vos modèles.

Sources :

https://barryvdh.nl/laravel/twig/2015/08/22/comparing-blade-and-twig-templates-in-laravel/

https://www.geekpress.fr/timber/

https://blog.julien-maury.com/veille/timber-le-nouveau-markup-wp/

https://openclassrooms.com/fr/courses/3619856-developpez-votre-site-web-avec-le-framework-symfony/3621582-le-moteur-de-templates-twig