Laravel : Générer un fichier PDF avec laravel-dompdf

Auteur	Domaine	Technologies
Wilo Ahadi	Développement d'app. web	Laravel, PHP

Découvrez comment créer un fichier PDF à partir d'un code HTML ou d'une vue (template Blade) avec le package laravel-dompdf dans un projet Laravel.

Sommaire

Laravel-dompdf, qu'est-ce ?

Le package laravel-dompdf est un wrapper pour Laravel de la librairie Dompdf qui permet de convertir le code HTML en PDF (Portable Document Format).

Il génère des fichiers PDF en prenant en charge les fonctionnalités suivantes :

- La plupart des propriétés CSS 2.1 et CSS 3, les feuilles de style internes et externes
- La plupart des attributs HTML 5

- Les images (GIF, PNG, JPG, …) et offre un support pour SVG
- Les tableaux (lignes, colonnes, bordures de tableau, style de cellule, …)
- La police d'écriture (font)
- …

Installer laravel-dompdf

Voici la procédure pour importer, intégrer et configurer le package laravel-dompdf dans un projet Laravel :

1. importation

Pour importer laravel-dompdf avec ses dépendances, ouvrez la console (invite de commandes) à la racine du projet, exécutez la commande *composer* suivante :

```
composer require barryvdh/laravel-dompdf
```

```
λ composer require barryvdh/laravel-dompdf
Using version ^0.8.6 for barryvdh/laravel-dompdf
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 5 installs, 0 updates, 0 removals
  - Installing sabberworm/php-css-parser (8.3.0): Downloading (100%)
  - Installing phenx/php-svg-lib (v0.3.3): Downloading (100%)
  - Installing phenx/php-font-lib (0.5.1): Downloading (100%)
  - Installing dompdf/dompdf (v0.8.5): Downloading (100%)
  - Installing barryvdh/laravel-dompdf (v0.8.6): Downloading (100%)
dompdf/dompdf suggests installing ext-imagick (Improves image processing performance)
dompdf/dompdf suggests installing ext-gmagick (Improves image processing performance)
Writing lock file
Generating optimized autoload files
```

2. Intégration

Laravel-dompdf importé, chargeons-le dans l'application en ajoutant le service provider au tableau **$providers** et la façade (« PDF » pour faire court) au tableau **$aliases** dans le fichier de configuration **config/app.php** :

```
// ...
    'providers' => [
        // ...,
        // Service Provider DomPDF
        Barryvdh\DomPDF\ServiceProvider::class
    ],

    'aliases' => [
        // ...,
        // Façade DomPDF
        "PDF" => Barryvdh\DomPDF\Facade::class
    ],
// ...
```

3. Configuration

Pour créer (copier) le fichier de configuration **config/dompdf.php** où vous pouvez modifier les options par défaut de dompdf, exécutez la commande *artisan* suivante :

```
php artisan vendor:publish --provider="Barryvdh\DomPDF\ServiceProvider"
```

Quelques options qu'on y trouve :

- orientation (*string*) : la disposition en portrait ("portrait") ou en paysage ("landscape")
- default_paper_size (*string*) : la taille du papier ("a3", "a4", …)
- dpi (*int*) : « Dots per Inch », le nombre de points ou de pixels qui seront rendus dans un pouce sur l'interface
- default_font (*string*) : La police par défaut à utiliser ("sans-serif", "courier", "helvetica", "dejavu sans", …)
- ...

Nous pouvons aussi modifier ces options dans le code avant de générer le PDF avec la méthode *setOptions()* :

```
PDF::setOptions([
    "defaultFont" => "Courier",
    "defaultPaperSize" => "a4",
    "dpi" => 130
]);
```

Utiliser laravel-dompdf

Maintenant que laravel-dompdf est installé et configuré, nous pouvons créer une nouvelle instance Barryvdh\DomPDF\PDF en indiquant le contenu du fichier PDF par :

1. Une vue (template Blade) avec ses données :

```
use App\Post;
use PDF;
// ...
public function getPostPdf (Post $post)
{
    // L'instance PDF avec une vue : resources/views/posts/show.blade.php
    $pdf = PDF::loadView('posts.show', compact('post'));
}
// ...
```

2. Le chemin vers un fichier :

```
$pdf = PDF::loadFile(public_path("documents/fichier.html"));
```

3. Une chaîne de caractères HTML :

```
$pdf = PDF::loadHTML("Mon contenu HTML ici");
```

De cet instance PDF, nous pouvons forcer le navigateur à lancer le téléchargement du fichier PDF généré :

```
use App\Post;
use PDF;
// ...
public function getPostPdf (Post $post)
{
    // L'instance PDF avec la vue resources/views/posts/show.blade.php
    $pdf = PDF::loadView('posts.show', compact('post'));

    // Lancement du téléchargement du fichier PDF
    return $pdf->download(\Str::slug($post->title).".pdf");

}
// ...
```

Ou l'afficher dans le navigateur :

```
return $pdf->stream();
```

Ou encore l'enregistrer :

```
$pdf->save(public_path("storage/documents/fichier.pdf"));
```

Astuces

1. Nous pouvons combiner les méthodes de la manière suivante :

```
return PDF::loadView('posts.show', compact('post'))
            ->setPaper('a4', 'landscape')
            ->setWarnings(false)
            ->save(public_path("storage/documents/fichier.pdf"))
            ->stream();
```

2. Pour le support UTF-8, ajouter le Metatag au template :

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
```

3. Pour faire un saut de page, nous pouvons utiliser la propriété CSS page-break-before ou page-break-after :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Ma page</title>
</head>
<body>
<h1>Titre de la page 1</h1>
<div>Contenu de la page 1</div>
```

```
<!-- Saut de page -->
<div style="page-break-after: always;" ></div>

<h1>Titre de la page 2</h1>
<div>Contenu de la page 2</div>
</body>
</html>
```