In this tutorial, I'll take you through an example on how to create a Laravel package in just 7 steps. So, let's go ahead and dive into it.

1. Create our folder for our new package.

Let's create a new folder called **packages** inside the root of our project.

We are going to use this packages folder instead of the vendor folder because files inside of our vendor folder should only be managed by composer. Once we have published our package we can then include it in our vendor folder using composer.

Next, we will create a folder inside of packages which will be the name of our package **calculator**.

Lastly we will add one last folder called **src** which is where the code for our package will live.

So, our folder structure will be **packages/calculator/src**

Let's go ahead and name our package in our project by initializing a composer.json file.

2. Name our package and initialize it with composer.

Inside your command prompt navigate to the folder with your package name. In our case: **packages/calculator**, and run the following command:

```
composer init
```

This will initialize the current folder as a composer package and it will ask you a series of questions to setup your package. It will ask for the name, description, author, and a little more info about your package. After you have finished this onscreen setup you will have a new file in your package folder called composer.json.

The composer.json file will look similar to the following:

```
{
    "name": "w3public/calculator",
    "description": "This is a normal calculator package so that we can
calculate summation, substraction, multiplication, and division",
    "authors": [
        {
            "name": "Bablu Ahmed",
            "email": "bablukpik@gmail.com"
        }
    ],
    "minimum-stability": "dev",
    "require": {}
}
```

Great! we have initialized our package we need to tell our main app composer.json file to load it.

3. Load our package from main application composer.json file

With every Laravel application there is a main composer.json file in the root of every new application. This is your main application composer file and this is where we define all our app dependencies.

Let's go ahead our newly created package via the **PSR-4** autoload block:

```
"autoload": {
        "psr-4": {
            "W3public\\Calculator\\": "packages/calculator/src/"
        }
    },
```

In the above code, `"W3public\\Calculator\\"` is called namespace for the package.

Then we need composer to run the autoloader to autoload our package. To do this we run the following command:

```
composer dump-autoload
```

4. Add our package service provider.

The service provider is the main entry inside of our package. This is where our package is loaded or booted. In the root of our application, let's create a ServiceProvider with an artisan command via this command line:

```
php artisan make:provider CalculatorServiceProvider
```

This will create a new file located at -
app/Providers/CalculatorServiceProvider.php

Let's move this file into our package folder at -**packages /calculator/src/CalculatorServiceProvider.php**

Don't forget to change your namespace to be your -
vendor_name\package_name\folder_name

I will describe details later, since the CalculatorServiceProvider.php file is directly inside **src** directory, our namespace will be as follows:

namespace W3public\Calculator;

As you can see in our new Service Provider class we have 2 methods boot() and register(). The boot() method is used to boot any routes, event listeners, or any other functionality you want to add to your package. The register() method is used to bind or inject any classes or functionality into the app container.

Next, we should inform Laravel that we have a service provider inside our **src** directory to add the new service provider got to **config/app.php** at the bottom of the **providers[]** array like this:

```
'providers' => [
W3public\Calculator\CalculatorServiceProvider::class,
],
```

Awesome! Our service provider is loaded and our package is ready to go! But we don't have any functionality yet. Let's tackle that by adding a route file for our package.

5. Add our package route file

Let's start by creating a new **routes/web.php** inside our package **src** directory, and add the following code:

```
Route::get("calculator", function(){
    echo "Hello from the calculator package!";
});
```

Then let's include our web.php file inside the boot() method of our Service Provider:

```
public function boot()
{
// include __DIR__.'/routes/web.php';
$this->loadRoutesFrom(__DIR__.'/routes/web.php');
}
```

And now visit **http://localhost:8000/calculator** route.

How awesome is that! We just called a route loaded from our package! Instead of outputting data inside of a route closure let's create a new controller.

6. Creating our package controller.

Let's create a new controller by running the following artisan command:

```
php artisan make:controller CalculatorController
```

Then let's move that controller from **app/Http/Controllers/CalculatorController.php** to

packages /calculator/src/Http/Controllers/CalculatorController.php

Next, I will create 4 methods like addition(), subtraction(), division(), and multiplication() inside of this controller. So the contents of the file will look like this:

```
<?php
namespace W3public\Calculator\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class CalculatorController extends Controller
{

public function addition($a, $b)
{
return $a+$b;
}

public function subtraction($a, $b)
{
return $a-$b;
}

public function division($a, $b)
{
return $a/$b;
}

public function multiplication($a, $b)
{
return $a*$b;
}

}
```

Again, remember to set your namespace for package controller - v**endor_name\package_name\directory_path**

This is because you have set the namespace in the main app's composer.json at psr-4 of autoload like this-

"W3public\\Calculator\\": "packages/calculator/src/"

Look, "W3public\Calculator" is covering "packages/calculator/src/" So the namespace will be like this -

namespace W3public\Calculator\Http\Controllers;

Finally, let's add a few more routes to our **routes/web.php** file:

```
Route::get('sum/{a}/{b}',
'W3public\Calculator\Http\Controllers\CalculatorController@addition');
```

```
Route::get('sub/{a}/{b}',
'W3public\Calculator\Http\Controllers\CalculatorController@subtraction')
;

//OR we can set a route group of the namespace

Route::group(['namespace'=>'W3public\Calculator\Http\Controllers'],funct
ion(){
    Route::get('sum/{a}/{b}','CalculatorController@addition');
    Route::get('sub/{a}/{b}','CalculatorController@subtraction');
    Route::get('div/{a}/{b}','CalculatorController@division');
    Route::get('mul/{a}/{b}','CalculatorController@multiplication');

});
```

Now navigate to **http://localhost:8000/sum/5/2** and we will end up with the result of addition.

Next, if we want to load a few views from our package, well that's easy enough. Let's do that in the next step.

7. Adding views for our package.

Let's create a new folder called **views** inside our package **src** directory. Then let's create a new file and call it **index.blade.php**. Inside the file let's add the following markup:

```
<!doctype html>
<html lang="en">
<head>
    <title>Calculator</title>
</head>
<body>
    <h2 style="text-align: center">Your Result: <span style="font-
weight:normal">{{ $result }}</span></h2>
</body>
</html>
```

Next, let's register where these views should be loaded from by adding a line of code to the boot method of Package Service Provider:

```
public function boot()
{
$this->loadViewsFrom(__DIR__.'/views', 'calculator');
}
```

In the above code, **calculator** is a package name but you can set any string. Later from the controller we will call view as follows:

package_name::view_name

For Example: return view('calculator::index', compact('result'));

Lastly, let's call the view from our calculator controller as follows:

```
public function addition($a, $b)
    {
        $result = $a + $b;
        return view('calculator::index', compact('result'));
    }

public function subtraction($a, $b)
    {
        $result = $a - $b;
        return view('calculator::index', compact('result'));
    }
```

If we load up the following URL http://localhost:8000/sum/5/2 in our browser we'll end up with the result of summation to the view.

Finally we have done it!

That's how to create a Laravel package. If you want you can also do more about how to publish your views and a few other advanced topics on publishing your package. Now let's do this

Publishing Views for our package

If we want to customize markup or any other things we have to publish views, for publishing to the application's resources/views/vendor directory, we may use the service provider's **publishes** method. The publishes method accepts an array of package view paths and their desired publish locations like this -

```
public function boot()
{
   $this->publishes([
      __DIR__.'/views' => resource_path('views/vendor/calculator'),
   ]);
}
```

Now run the **vendor:publish** Artisan command then the **index.blade.php** file will be copied to the specified publish location.

Submitting the package to the packagist.org

If you want to submit your package to **packagist.org** so that you can install and uninstall or manage the package with **composer** you'll need to do the following steps:

- Firstly we need **Package Discovery** for automatically add **Service Providers** to the providers array. So add the following things to the **packages/calculator/composer.json** file:

```
"extra": {
   "laravel": {
      "providers": [
         "W3public\\Calculator\\CalculatorServiceProvider"
      ]
   }
},
```

- Submit your package to Github with Tag (i.e, https://github.com/w3public/calculator)
- Submit your package Github URL to Packagist.
- Add the API key of Packagist to Github

For more details:
https://stackoverflow.com/questions/53990368/how-to-add-api-key-of-packagist-to-github-using-webhook

And then you'll be able to include your package inside of your main project composer.json file and run composer install/require to install your package in your application.

Created by Bablu Ahmed

Website: www.w3public.com

Email: bablukpik@gmail.com

Skype: bablukpik

Address: Dhaka, Bangladesh.