LARAVELISTA

A COLLECTION OF
LARAVEL TUTORIALS

MARIO BAŠIĆ

A Collection of Laravel Tutorials

Mario Bašić

This book is for sale at http://leanpub.com/laravelista-collection

This version was published on 2018-05-02

Contents

Getting Started 1

Elementary Laravel 2

Laravel on Windows with Homestead 29

Laravel on Windows with Laragon 51

Configure Laravel 5 for Shared Hosting 64

Getting Started

Homestead or Laragon, pick one and start building.

Elementary Laravel

Only the basics in the simplest way possible. Everything you need to create a website.

View Source Code

Source code for this tutorial is available here[1].

Installation

We have to install Laravel before we can start using it, so let's do that now.

Published at: **17. October, 2016.**

Welcome to my new course called *Elementary Laravel*. In this course, we will build a simple *business* website with Laravel 5.3. The idea is to teach you how to build a website with Laravel by using the least amount of steps necessary and provide you with references to expand the knowledge you obtain from this course.

Don't bother yourself with questions like: "Is this the best practice?" or "Should I be doing this-this way?" etc. Everybody starts somewhere and I think that this is the best starting point for learning Laravel. By the end of this course, you will have a working *business* website and basic knowledge about Laravel.

Requirements

These are the tools that you will need for this tutorial:

- PHP[2]
- Composer[3]
- A text editor or an IDE. *I suggest using Atom*[4].
- A Web browser. *Chrome*[5] *preferred.*

Installation

If you haven't already, download the Laravel installer using Composer:

[1] https://github.com/laravelista/elementary-laravel
[2] http://php.net/
[3] https://getcomposer.org/
[4] https://atom.io/
[5] https://www.google.com/chrome/index.html

Installing Laravel Installer

```
composer global require "laravel/installer"
```

To create a fresh Laravel installation we will use this command:

Creating a fresh Laravel installation

```
laravel new website
```

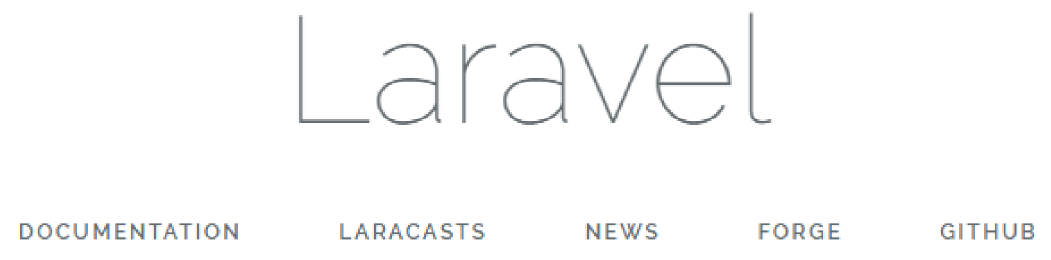
This command will create a directory called `website` containing a fresh Laravel installation.

To see how our newly installed Laravel application looks like we will use the built-in Local development server. To start a development server at `http://localhost:8000` run this command:

Starting a development server

```
php artisan serve
```

Now open your browser to that URL and you will see this screen:



Laravel installed

View changes in this commit

d49ab3224cee877d9a201a0cf973e07988317600[6].

[6]https://github.com/laravelista/elementary-laravel/commit/d49ab3224cee877d9a201a0cf973e07988317600

Improve your skills!

Instead of using the built-in PHP development server, you should really learn how to install and configure Homestead[7]. If you are on Windows, I already have a course on Homestead on Windows and Laragon on Windows. If you are on a Mac, take a look at Valet[8].

You now have Laravel installed and ready to go.

Routing

Routes are the entry points to your application, so it is only logical to learn about them first.

Published at: **17. October, 2016.**

Routes are the main entry points for your Laravel application. With routes, we define URLs that are accessible on our website.

If you need an about page, you probably want it to be accessible at `/about` URL. Routes define what URLs are accessible and what happens when a route is triggered. Think of them as an index for your website. When you want to locate something, you just have to take a look at the routes file.

URL structure

For our *business* website, we will have a structure like this:

- `GET /` - Our home page.
- `GET /about` - The about page.
- `GET /contact` - Contact page that has a contact form.
- `POST /contact` - When contact form is submitted, data will be sent to this route.

If you don't know already what `GET` and `POST` mean, think of them this way. `GET` is used for getting a web page. `POST` is used to send data to a web page. While this isn't the exact definition, for the purpose of this tutorial it will do.

Improve your skills!

If you are thinking of starting a career in web development, you should learn more about HTTP methods: Method definitions[9], HTTP Methods: GET vs. POST[10] and Using HTTP Methods for RESTful Services[11].

[7]https://laravel.com/docs/5.3/homestead
[8]https://laravel.com/docs/5.3/valet
[9]https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
[10]http://www.w3schools.com/tags/ref_httpmethods.asp
[11]http://www.restapitutorial.com/lessons/httpmethods.html

Defining our routes

To define the routes for our website, in your text editor open the routes file `routes/web.php`.

You should see the default route which you can see if you visit `http://localhost:8000` in your browser and a comment describing this routes file.

Since we want to keep the route `/` for our home page, we only need to change the view that is being returned from that route closure. Change the line `return view('welcome');` to `return view('home');`.

Now to create the rest of our routes, add this bellow that route:

Adding routes to the routes file

```
Route::get('about', function() {
    return view('about');
});

Route::get('contact', function() {
    return view('contact');
});

Route::post('contact', function() {
    //
});
```

Save the changes and open your browser to `http://localhost:8000`. You should get an error now saying `View [home] not found.`. This means that Laravel has not found the `home` view file that we specified in our `/` route. Our next step is to create that view file and any other view file that we specified in our routes file.

View changes in this commit

4f504843665c9e1d0cd8d41e0eb60259daaeaabc[12].

Views

Views contain the HTML served by your application and separate your application logic from your presentation logic.

[12] https://github.com/laravelista/elementary-laravel/commit/4f504843665c9e1d0cd8d41e0eb60259daaeaabc

Published at: **24. October, 2016.**

Views are files located in `resources/views` with `.blade.php` extension. Views contain HTML which is served by your application.

For our *business* website we need to create the views that we have specified in our routes file.

Create View Files

Create the following files in the `resources/views` directory:

- `home.blade.php`
- `about.blade.php`
- `contact.blade.php`

If you try to view the home page now, you will see a blank page. That is also true for all other routes that we have defined.

Bootstrap Home Page

Good, now we will add HTML to our `resources/views/home.blade.php` view. We will be using Bootstrap[13] to quickly get started with some basic page design. Open the file and add the following inside it:

Bootstrapping Bootstrap

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Home page</title>

    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7\
/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3\
RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
  </head>
  <body>
    <h1>Hello, world!</h1>
```

[13]http://getbootstrap.com

```
    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.\
js"></script>
    <!-- Latest compiled and minified JavaScript -->
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.mi\
n.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGN\
IcPD7Txa" crossorigin="anonymous"></script>
  </body>
</html>
```

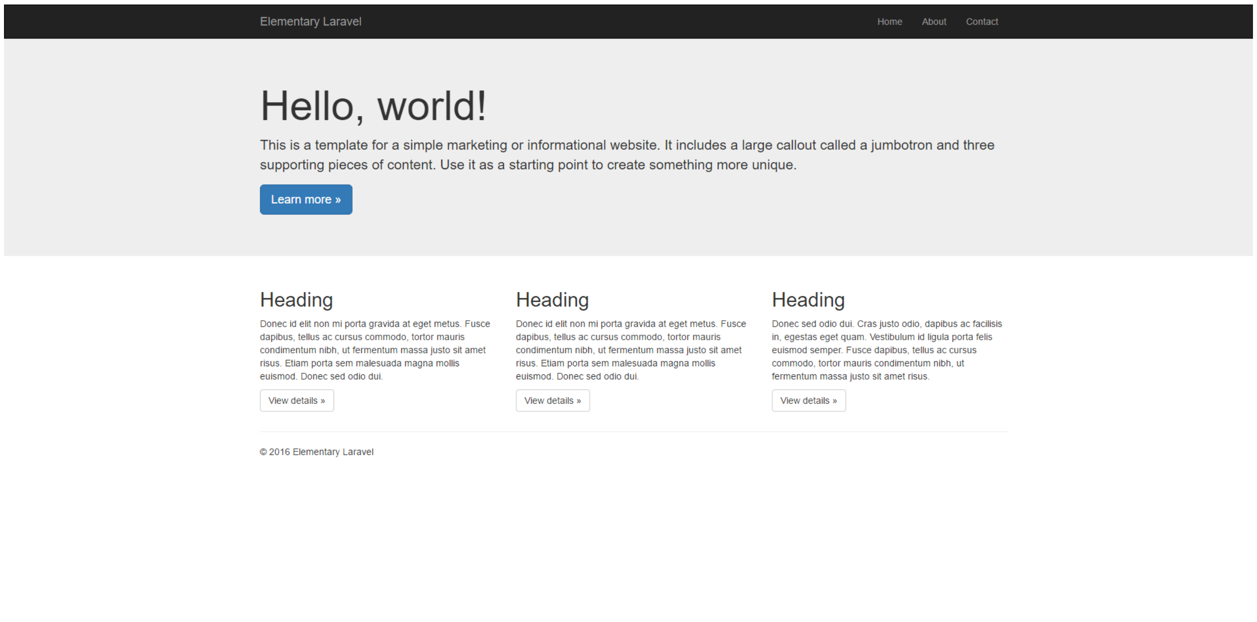
Save the changes and open your browser to `/`. *Hit F5 to refresh the page if needed.* You should see `Hello, world!` displayed on the page. **Excellent!** Now you can use Bootstrap on this page to create your home page.

Improve your skills!

Bootstrap is the fastest way to get started with a new website. It contains almost everything needed to design a website and it also looks good by default. I suggest that you read: *Getting started*, *CSS* and *Components* pages from its website[14].

This is how I have made the home page to look using Bootstrap. I have used a Jumbotron[15] example as a starting point.

[14]http://getbootstrap.com/
[15]http://getbootstrap.com/examples/jumbotron/



Hello, world!

This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron and three supporting pieces of content. Use it as a starting point to create something more unique.

Heading

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Heading

Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Heading

Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.

© 2016 Elementary Laravel

Home page

Feel free to change the page however you like it.

View changes in this commit

8a5128154e3cc0db6f02780029538ca3ec18f52b[16].

Use Helpers

There are a few things that we can do to improve our page. First, if you look at the `home.blade.php` file[17] you can notice that for defining links to our other routes we use `<a href="/about">About</a>`. While this will work, it is much better to use a Laravel helper function `url('/about')` which creates a full URL to the route.

Change all links so that they use the Laravel `url` helper. For example, change `<a href="/">Home</a>` to `<a href="{{ url('/') }}">Home</a>`. Do this for all links.

View changes in this commit

2df2d0ae2172affda92c9142753c44814b651752[18].

Before you start copying the HTML from our home page to other pages, ask yourself *Is there a way to reuse sections of our homepage on other pages to avoid copying the code*?

[16]https://github.com/laravelista/elementary-laravel/commit/8a5128154e3cc0db6f02780029538ca3ec18f52b
[17]https://github.com/laravelista/elementary-laravel/blob/8a5128154e3cc0db6f02780029538ca3ec18f52b/resources/views/home.blade.php
[18]https://github.com/laravelista/elementary-laravel/commit/2df2d0ae2172affda92c9142753c44814b651752

As you may have noticed `{{ }}` is used to echo a value in the view. There will be more talk about this in the next tutorial.

Blade templates

Blade is a templating engine provided with Laravel and unlike other templating engines it does not restrict you from using plain PHP code in your views.

Published at: **02. November, 2016.**

In my opinion one of the best parts of Laravel is the Blade templating engine. It already comes with Laravel and has everything you need and more. It enables you to work with templates and layouts, display data, use control structures, include subviews, use stacks, inject services in views and if that is still not enough you can easily extend it to do whatever you desire.

The purpose of this tutorial is not to teach you everything that Blade can do, but to teach you about templates, layouts, sections, subviews and basic data presentation.

Improve your skills!

Learn more about Blade templates by reading the documentation[19]. It is very important to know what you can do with it.

In our previous tutorial I have left you with a question to ask yourself: *Is there a way to reuse sections of our home page on other pages to avoid copying the code*?

The answer is yes, there is and it is called Blade templates.

Layouts

When building your templates you should start from the most outer shell and those are the HTML tags `html`, `head` and `body`. We will extract a part of the HTML from our `resources/views/home.blade.php` file that is common for all other pages and place it in `resources/views/layouts/default.blade.php`.

What I want you to do now is to move this:

[19]https://laravel.com/docs/5.3/blade

Extracting a layout page

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Home page</title>

    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7\
/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3\
RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
    <style>
        body {
            padding-bottom: 20px;
        }
        .navbar {
            margin-bottom: 0px;
            border-radius: 0;
        }
    </style>
  </head>
  <body>


    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.\
js"></script>
    <!-- Latest compiled and minified JavaScript -->
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.mi\
n.js" integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGN\
IcPD7Txa" crossorigin="anonymous"></script>
  </body>
</html>
```

to a new file in `resources/views/layouts/default.blade.php`. So that you `home.blade.php` file now only contains:

Content of the home page file

```
<nav class="navbar navbar-inverse">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle collapsed" data-toggle="collaps\
e" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="{{ url('/') }}">Elementary Laravel</a>
    </div>
    <div id="navbar" class="navbar-collapse collapse">
        <ul class="nav navbar-nav navbar-right">
          <li><a href="{{ url('/') }}">Home</a></li>
          <li><a href="{{ url('/about') }}">About</a></li>
          <li><a href="{{ url('/contact') }}">Contact</a></li>
        </ul>
    </div><!--/.navbar-collapse -->
  </div>
</nav>

<!-- Main jumbotron for a primary marketing message or call to action -->
<div class="jumbotron">
  <div class="container">
    <h1>Hello, world!</h1>
    <p>This is a template for a simple marketing or informational website. It in\
cludes a large callout called a jumbotron and three supporting pieces of content\
. Use it as a starting point to create something more unique.</p>
    <p><a class="btn btn-primary btn-lg" href="#" role="button">Learn more »</a>\
</p>
  </div>
</div>

<div class="container">
  <!-- Example row of columns -->
  <div class="row">
    <div class="col-md-4">
      <h2>Heading</h2>
      <p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus\
 ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit\
 amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.\
 </p>
      <p><a class="btn btn-default" href="#" role="button">View details »</a></p>
    </div>
    <div class="col-md-4">
      <h2>Heading</h2>
      <p>Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus\
 ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit\
 amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.\
 </p>
      <p><a class="btn btn-default" href="#" role="button">View details »</a></p>
   </div>
    <div class="col-md-4">
      <h2>Heading</h2>
      <p>Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas e\
get quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus\
 ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit\
 amet risus.</p>
      <p><a class="btn btn-default" href="#" role="button">View details »</a></p>
    </div>
  </div>

  <hr>

  <footer>
    <p>© 2016 Elementary Laravel</p>
  </footer>
</div> <!-- /container -->
```

Now that we have created our first layout template in `layouts/default.blade.php`, we must tell our view `home.blade.php` to extend upon that layout. We do that by adding

Extending a page with a layout

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

at the top of the file `home.blade.php`. There is one more step before we continue. We have to tell our layout where to display the HTML from the page we want.

Sections

Sections are used to tell the layout where we want the content of a section to be displayed.

Go to our `layouts/default.blade.php` file and just bellow the opening `body` tag place the following:

Defining a section for content

```
@yield('content')
```

and now in `home.blade.php` wrap all content bellow `@extends('layouts.default')` in a section block:

Populating content section

```
@section('content')
    {{-- Place wrapped content instead of this Blade comment --}}
@stop
```

If you take a look at `http://localhost:8000` you should see that everything looks the same. That is the point, but can you notice how much cleaner our home view looks now?

Subviews

We can make it even cleaner by using subviews to extract our navigation to a subview which we will include in our default layout template.

Create a new file in `resources/views/layouts/partials/navbar.blade.php` and move our `navbar` from `home.blade.php` to that file.

Creating a navbar partial

```
<nav class="navbar navbar-inverse">
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle collapsed" data-toggle="colla\
pse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
          <span class="sr-only">Toggle navigation</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a class="navbar-brand" href="{{ url('/') }}">Elementary Laravel</a>
      </div>
      <div id="navbar" class="navbar-collapse collapse">
          <ul class="nav navbar-nav navbar-right">
            <li><a href="{{ url('/') }}">Home</a></li>
            <li><a href="{{ url('/about') }}">About</a></li>
            <li><a href="{{ url('/contact') }}">Contact</a></li>
        </ul>
      </div><!--/.navbar-collapse -->
    </div>
  </nav>
```

Good, now we have to tell our layout to include that subview. Go to `layouts/default.blade.php` and just above `@yield('content')` place the following:

Including navbar partial

```
@include('layouts.partials.navbar')
```

Save the changes and if you look at the browser and hit refresh it should still look the same, but our home view file is even cleaner now. Awesome!

There are more things that you can move into subviews if you think that you can benefit from it, but for this tutorial, this seems fine to me.

Data presentation

> The name of this chapter is misleading at best, I know.

If you have been following along, you may have noticed that in out default layout file the `title` tag is hardcoded to be `Home page`, but what about our other pages? Should the title bar not hold some other value instead of `Home page`?

There is a simple way of achieving this. Go to `layouts/default.blade.php` and replace the value of `title` to be:

Defining a place for title section

```
@yield('title', 'Elementary Laravel')
```

This tells our template to place a section from our view called `title` here, but if it cannot find it then place the default value of `Elementary Laravel`.

Now to set the title in our view file, go to `home.blade.php` and create a new section just bellow the `@extends('layouts.default')` and above the `@section('content')`:

Creating a title section

```
@section('title', 'Home page')
```

If you take a look at `http://localhost:8000` you should see that everything looks the same. Great job!

Wrapping things up

Now we have a way to populate our other views `about.blade.php` and `contact.blade.php`. Copy the following to those files and change the title accordingly:

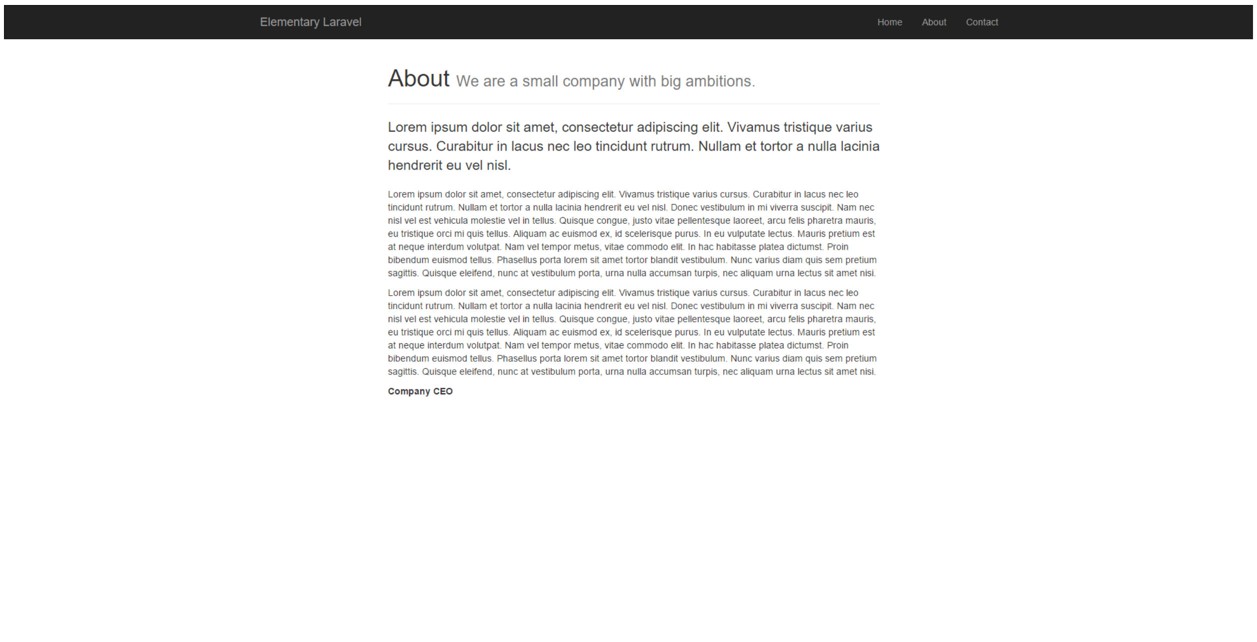
Page skeleton

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

@section('title', 'About page')

@section('content')
    {{-- Place content instead of this Blade comment --}}
@stop
```

I will quickly populate those pages with some Bootstrap. You can modify them however you want or you can copy the code from what I have done:

`about.blade.php`[20]

[20] https://raw.githubusercontent.com/laravelista/elementary-laravel/00f0293ad4668f08f11ba67c890f892bddb579d8/resources/views/about.blade.php



About We are a small company with big ambitions.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus tristique varius cursus. Curabitur in lacus nec leo tincidunt rutrum. Nullam et tortor a nulla lacinia hendrerit eu vel nisl.

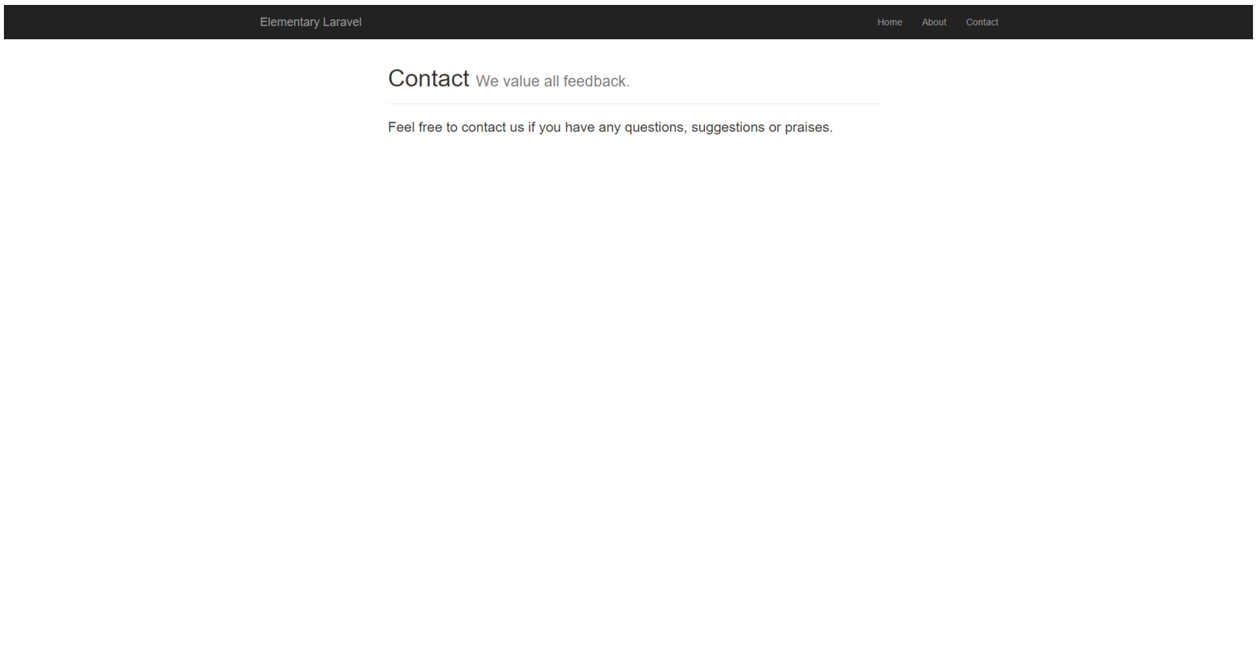
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus tristique varius cursus. Curabitur in lacus nec leo tincidunt rutrum. Nullam et tortor a nulla lacinia hendrerit eu vel nisl. Donec vestibulum in mi viverra suscipit. Nam nec nisl vel est vehicula molestie vel in tellus. Quisque congue, justo vitae pellentesque laoreet, arcu felis pharetra mauris, eu tristique orci mi quis tellus. Aliquam ac euismod ex, id scelerisque purus. In eu vulputate lectus. Mauris pretium est at neque interdum volutpat. Nam vel tempor metus, vitae commodo elit. In hac habitasse platea dictumst. Proin bibendum euismod tellus. Phasellus porta lorem sit amet tortor blandit vestibulum. Nunc varius diam quis sem pretium sagittis. Quisque eleifend, nunc at vestibulum porta, urna nulla accumsan turpis, nec aliquam urna lectus sit amet nisi.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus tristique varius cursus. Curabitur in lacus nec leo tincidunt rutrum. Nullam et tortor a nulla lacinia hendrerit eu vel nisl. Donec vestibulum in mi viverra suscipit. Nam nec nisl vel est vehicula molestie vel in tellus. Quisque congue, justo vitae pellentesque laoreet, arcu felis pharetra mauris, eu tristique orci mi quis tellus. Aliquam ac euismod ex, id scelerisque purus. In eu vulputate lectus. Mauris pretium est at neque interdum volutpat. Nam vel tempor metus, vitae commodo elit. In hac habitasse platea dictumst. Proin bibendum euismod tellus. Phasellus porta lorem sit amet tortor blandit vestibulum. Nunc varius diam quis sem pretium sagittis. Quisque eleifend, nunc at vestibulum porta, urna nulla accumsan turpis, nec aliquam urna lectus sit amet nisi.

Company CEO

About page

`contact.blade.php`[21]



Contact We value all feedback.

Feel free to contact us if you have any questions, suggestions or praises.

Contact page

[21] https://raw.githubusercontent.com/laravelista/elementary-laravel/00f0293ad4668f08f11ba67c890f892bddb579d8/resources/views/contact.blade.php

View changes in this commit

00f0293ad4668f08f11ba67c890f892bddb579d8[22].

We now have a home and about pages complete. On our contact page, we surely want to have a contact form, which sends an email upon successful validation.

Forms

Forms are an important part of any web application. I'll show you a quick and easy way to create a form with helpers.

Published at: **11. November, 2016.**

We will now create a contact form. We will use normal HTML with some Laravel helpers in order to provide a better user experience.

Improve your skills!

To improve upon the form that we will create in this tutorial I advise you to take a look at Forms & HTML[23] package. I have an in-depth tutorial about it called Laravel Forms & HTML so be sure to check it out.

Create the form

We will use basic Bootstrap styling to design the form. We will require from the user to enter his:

- name
- email
- message (*comment* - We will cover at a later point)

Improve your skills!

It is very important to know how to build and design forms with Bootstrap, so I recommend reading the documentation[24] about it.

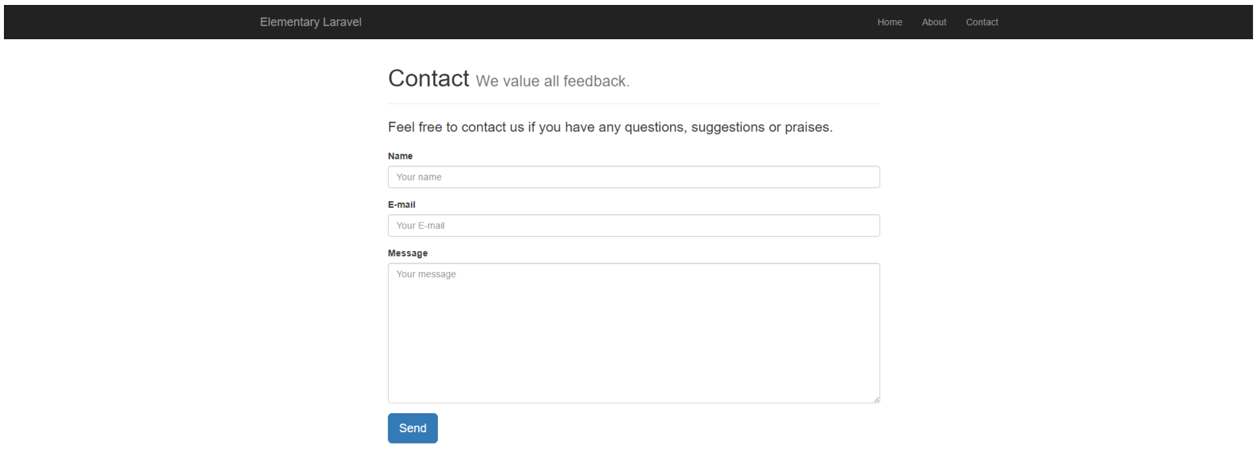
In our `contact.blade.php` you will see a Blade comment `{{-- Contact form goes here --}}`. Replace that line with the following:

[22]https://github.com/laravelista/elementary-laravel/commit/00f0293ad4668f08f11ba67c890f892bddb579d8
[23]https://laravelcollective.com/docs/5.3/html
[24]http://getbootstrap.com/css/#forms

Creating a contact form

```
<form method="POST" action="{{ url('/contact') }}">
    {{ csrf_field() }}
    <div class="form-group">
        <label for="name">Name</label>
        <input id="name" type="text" class="form-control" name="name" value="{{ \
old('name') }}" placeholder="Your name">
    </div>
    <div class="form-group">
        <label for="email">E-mail</label>
        <input id="email" type="email" class="form-control" name="email" value="\
{{ old('email') }}" placeholder="Your E-mail">
    </div>
    <div class="form-group">
        <label for="comment">Message</label>
        <textarea rows="10" id="comment" class="form-control" name="comment" pla\
ceholder="Your message">{{ old('comment') }}</textarea>
    </div>
    <button type="submit" class="btn btn-primary btn-lg">Send</button>
</form>
```

I will explain the helpers used here in the following chapter, but for now, save the changes and open your browser to `http://localhost:8000/contact`. You will see that the page looks like this now:



Contact We value all feedback.

Feel free to contact us if you have any questions, suggestions or praises.

Contact with form

We have specified that we want to `POST` the data from the form to `/contact` URL. If you remember our routes file from a few tutorials ago, we have a route for that method:

Adding a route for posting contact form

```
Route::post('contact', function() {
    //
});
```

If you press `Send` on the form now, you will get a blank page. That is because we are not returning anything from our route that handles form submission.

View changes in this commit

315057e9e97d0d34cde9a683cd00bd5e2dedfdff[25].

Helpers used

We have used two Laravel helpers in this form.

Improve your skills!

To learn more about all helpers that come with Laravel visit the documentation for Helpers[26].

`old()`

The `old` function retrieves an old input value flashed into the session. This will be very helpful in our next tutorials where we will tackle validation. What this does is it keeps the value that the user entered in the input field so that if the validation fails, the input entered by the user is preserved. He does not need to type it again.

`csrf_field()`

The `csrf_field` function generates an HTML `hidden` input field containing the value of the CSRF token. Laravel automatically generates a CSRF "token" for each active user session managed by the application. This token is used to verify that the authenticated user is the one actually making the requests to the application. Read more about this here[27].

A quick recap of this tutorial:

[25]https://github.com/laravelista/elementary-laravel/commit/315057e9e97d0d34cde9a683cd00bd5e2dedfdff
[26]https://laravel.com/docs/5.3/helpers
[27]https://laravel.com/docs/5.3/csrf

- I've shown you how to build a contact form
- You have learned about `old` and `crsf_field` helpers
- You have been given a lot of documentation to read in order to improve your skills
- The form can be submitted and we are returned a blank screen

What we are still missing is *Validation*. Are we going to blindly believe our users, that they have entered a valid email address or that they have entered all fields that we require? Hell no! This is where Laravel shines, the **Validation**.

Validation

Validation consists of two parts, validating the data from the user and displaying the errors messages back to the user..

Published at: **28. November, 2016.**

Out of the box, Laravel comes loaded with validation options[28] and it is also very easy and quick to implement it however you want. In the previous tutorial, we have left things off at a contact form. We have created a contact form which submits its data to the URL we specified `POST /contact` and it is being handled by our route.

Scenarios

This is what we want to happen when the user submits the contact form:

If the data entered passes validation, our application should:

- send us an email
- redirect the user to `/contact` page
- display success message to the user

If the data entered does not pass validation, our application should:

- redirect the user to the contact form **with old input**
- display errors messages telling the user what he did wrong

Validating data

Back to our `routes/web.php` file. Locate the route:

[28]https://laravel.com/docs/5.3/validation

Empty contact POST route

```
Route::post('contact', function() {
    //
});
```

First we have to tell our route to use dependency injection to inject the `Request` like so:

Injecting Request dependency

```
use Illuminate\Http\Request;
Route::post('contact', function(Request $request) {
    // place code here
});
```

Now we can access the `request`, meaning that we can validate the data inside it. We will manually build our `Validator` but we will also use the *Automatic Redirection* feature to automatically handle the redirection and error processing if the validation fails.

Place this code inside our POST route:

Validating Request data

```
\Validator::make($request->all(), [
    'name' => 'required|string',
    'email' => 'required|email',
    'comment' => 'required|string'
])->validate();

// normal code execution with successful validation.
// send email or do whatever you want here,
// redirect user back and notify him of our success
```

Now let me explain. From the Validation documentation[29]:

"If you would like to create a validator instance manually but still take advantage of the automatic redirection offered by the `ValidatesRequest` trait, you may call the `validate` method on an existing validator instance. If validation fails, the user will automatically be redirected or, in the case of an AJAX request, a JSON response will be returned."

As you can see, we are specifying parameter names and validation rules for each.

[29] https://laravel.com/docs/5.3/validation#automatic-redirection

Improve your skills!

To understand and know which other validation rules exist, read the documentation on Available Validation Rules[30].

View changes in this commit

dbc88e6ccae9f8c8b0c92e9b19edb7e3cefd8949[31].

Displaying errors

If you try to submit the form now with no data, you will be redirected back to out `/contact` page and it will seem like nothing happened, but in fact, the validation was triggered and it failed because it did not pass our validation rules. `required` validation rule means that the field under inspection must have some data in it. If you populate all fields in our contact form correctly and submit the form you should get a blank page again.

The smart thing to do here is to provide the user some information on why the validation has failed. In the Validation documentation under Displaying The Validation Errors[32] is this snippet:

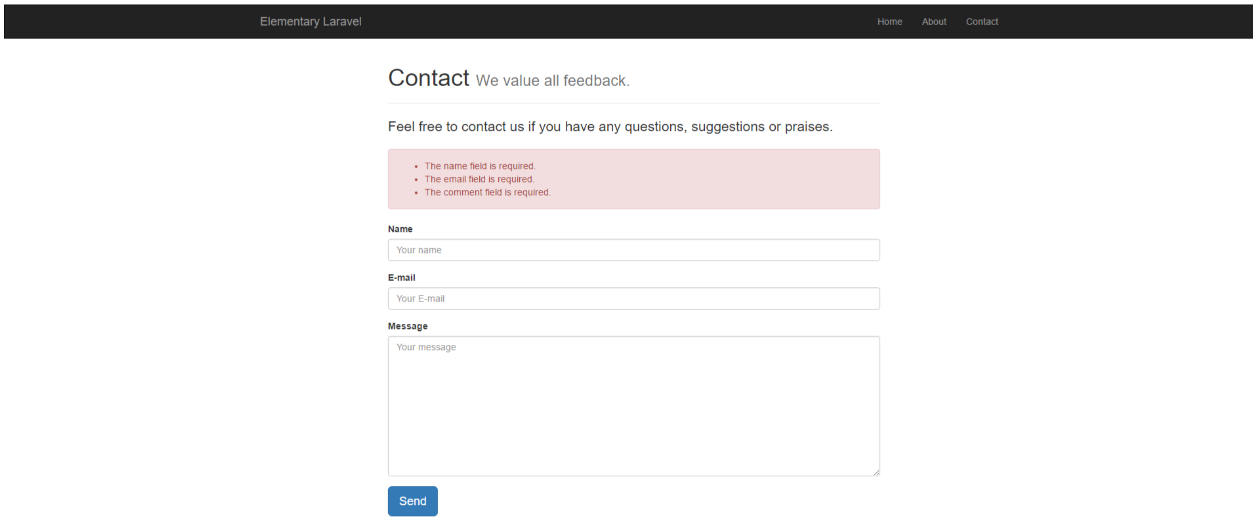
Displaying validation errors

```
@if (count($errors) > 0)
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

This snippet loops through all the errors (if any) in the session and displays them in an unordered list (Twitter Bootstrap styling, but you can modify it however you want). Place this snippet just above the `form` tag in `resources/views/contact.blade.php`.

Now if you try to submit the form with no data, you should get a page which looks like this:

[30] https://laravel.com/docs/5.3/validation#available-validation-rules
[31] https://github.com/laravelista/elementary-laravel/commit/dbc88e6ccae9f8c8b0c92e9b19edb7e3cefd8949
[32] https://laravel.com/docs/5.3/validation#quick-displaying-the-validation-errors



Contact We value all feedback.

Feel free to contact us if you have any questions, suggestions or praises.

- The name field is required.
- The email field is required.
- The comment field is required.

Name

Your name

E-mail

Your E-mail

Message

Your message

Failed Validation

Try messing around with different values to see how it works.

View changes in this commit

62b5b9d7defe9366a97e69c1772371ddfa786350[33].

The validation is now working. We are successfully validating the data and displaying errors to the user. We still have to process what happens if the validation is successful.

Follow the happy path

We want to redirect the user to the contact form (empty) and display a success message signaling that everything went ok.

In `resources/views/contact.blade.php`, just bellow our validation code place the following:

[33]https://github.com/laravelista/elementary-laravel/commit/62b5b9d7defe9366a97e69c1772371ddfa786350

Redirecting to the contact page with a success message

```
return redirect('/contact')->with([
    'success_message' => 'Your message has been sent!'
]);
```

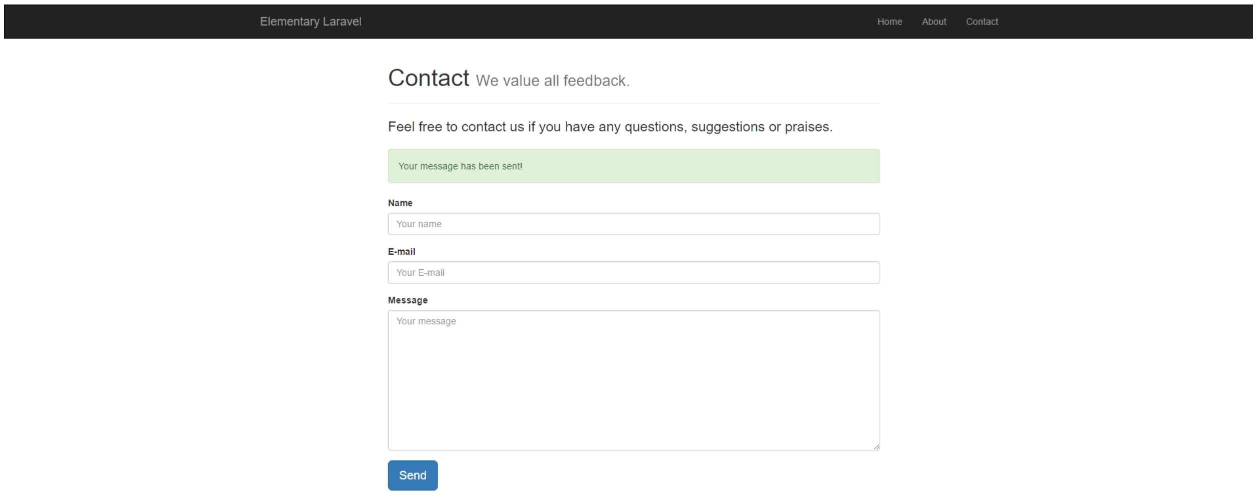
This code redirects the user to the `/contact` page. It also flashes session data with a variable called `success_message`. That variable will be available on our page.

Now to catch that variable on our contact page, we have to add this block of code to the place where we want it to be displayed:

Displaying the success message

```
@if (session('success_message'))
    <div class="alert alert-success">
        {{ session('success_message') }}
    </div>
@endif
```

Add this code just above the `form` tag in `contact.blade.php`. If you populate the form now with data and submit it, you will be presented with this nice little green alert box:



Contact We value all feedback.

Feel free to contact us if you have any questions, suggestions or praises.

Your message has been sent!

Success Message

View changes in this commit

8c4b2c8d534a7d101951fae2f04e6b2cce129fae[34].

Our contact form is now working, the only thing left is actually sending the email :)

Mail

Laravel provides a clean and simple API over the popular SwiftMailer library, allowing you to quickly get started sending mail.

Published at: **11. December, 2016.**

Laravel 5.3 comes with a new feature called Mailables[35] where each type of email sent by your application is represented as a "mailable" class. In the previous tutorial we have hooked our contact form with validation and upon successful validation, presented the user with a "success" message. In this tutorial, we will simulate sending an actual email from our contact form.

Mail & Local Development

Improve your skills!

Laravel comes with drivers for many local and cloud-based services for sending emails. Check the documentation[36] to see how to use a specific driver.

Since we are in the development phase in our application, we don't want to actually send emails to live email addresses. To avoid doing so we will use the **Log Driver**.

Go to your local `.env` file and set a key/value for `MAIL_DRIVER=log`. Comment out all other keys that start with `MAIL_`. By doing so, all emails sent from our application will be written in the log file and not actually sent.

Mailables

Mailabes are stored in `app/Mail`.

Generate a new Mailable class

To create a new Mailable enter this command:

[34]https://github.com/laravelista/elementary-laravel/commit/8c4b2c8d534a7d101951fae2f04e6b2cce129fae
[35]https://laravel.com/docs/5.3/mail
[36]https://laravel.com/docs/5.3/mail

Creating a Mailable

```
php artisan make:mail FeedbackReceived
```

This command will create a new file `app/Mail/FeedbackReceived.php`.

View changes in this commit

0469b561877cd6eb80667264d10480ade0792b26[37].

There are a few things that we need to configure in our new Mailable class:

- Sender
- View
- Data

Configuring the Sender

First, we need to configure who the email is going to be “from”. We do that by setting the `from` method inside the `build` method of the `FeedbackReceived` class.

Configuring the sender

```
public function build()
{
    return $this
        ->from('you@company.com')
        ->view('emails.contact');
}
```

You can change the from field to anything you want or which represents your business.

Configuring the View

In the code above, we have configured the sender and specified which template should be used when rendering the email’s contents. We will now create a blank template file as specified.

Create a new folder in `resources/views` called `emails` and inside it create a file called `contact.blade.php`. Place the following code inside:

[37]https://github.com/laravelista/elementary-laravel/commit/0469b561877cd6eb80667264d10480ade0792b26

Writing Email body

```
<h1>Thank you for contacting us! Your message has been received.</h1>
```

Setting the Data

So far, we are only sending the generic confirmation message to the user who has submitted the contact form. It would be nice if we could address the user by his name and display the message that he has sent us.

To do so, we have to set public properties on our `FeedbackReceived` class for *name* and *comment*:

Setting class properties

```
public $name;
public $comment;

public function __construct($name, $comment)
{
    $this->name = $name;
    $this->comment = $comment;
}
```

Once the data has been set to a public property, it will be automatically available in our view as a variable. Let's modify our view template to include these variables:

Expanding the Email body with comment from Class property

```
<h1>Thank you for contacting us {{ $name }}! Your message has been received.</h1>

<p>{{ $comment }}</p>
```

Sending Mail

To send the actual email that we have configured in the previous chapter, we have to open the file `routes/web.php` and replace the TODO comment in `Route::post('contact')` with the following:

Sending Mail

```
Mail::to($request->get('email'))->send(new FeedbackReceived($request->get('name'\
), $request->get('comment')));
```

Don't forget to add the use statements above the route:

Adding use statements

```
use App\Mail\FeedbackReceived;
use Illuminate\Support\Facades\Mail;
```

Now if you populate the contact form with data that passes validation, the email will be logged in `storage/logs/laravel.log`.

Check that log file to see if the email is logged there.

View changes in this commit

300445c0dc40d7e3099ee5602acfc98979b9a85a[38].

Congratulations! This marks the completion of the course Elementary Laravel. Thank you for reading this far.

By completing this course you will have a basic "elementary" understanding on how to use Laravel to create simple websites. The code for this course is available on Github and you are more than welcome to fork it, improve it and contribute to it.

[38]https://github.com/laravelista/elementary-laravel/commit/8c4b2c8d534a7d101951fae2f04e6b2cce129fae

Laravel on Windows with Homestead

If you are getting started with Laravel and are using Windows, this is the right starting point for you. We will cover everything from installing PHP & Git to using Homestead virtual machine and creating your first blank Laravel application.

Prepare for modern PHP applications

In this introductory tutorial, we will be installing the absolute basic software that is required to run modern PHP frameworks like Laravel.

Published at: **12. March, 2016.**

So, you have heard about Laravel and want to learn how to use it? Are you using Windows? Then this is the right starting point for you. In this tutorial, we will install some basic software that you will need on your PC.

These are the tools that you will be needing:

- Git + Git Bash[39] `(2.7.0)`
- PHP[40] `(7.0.3 - VC14 x64 Thread Safe)`

  I'm running Microsoft Windows 10 x64, that is why I'm using the x64 version of PHP. If you are on an x86 (32bit) Windows then you should use an x86 version of PHP.

Install PHP on Windows

First, we need to download PHP zip file from the website mentioned above. Unzip that file a place its content in `C:\tools\php` directory. Now inside that folder, you will find a file called `php.ini-development`. Copy/paste that file and rename it to `php.ini`.

There are somethings that we will need to enable inside that file, so grab your favorite text editor (I prefer Sublime Text, but you can use notepad as well) and open that file:

- On line `:368` change `max_execution_time = 30` to `max_execution_time = 300`. *You will thank me for this later.*

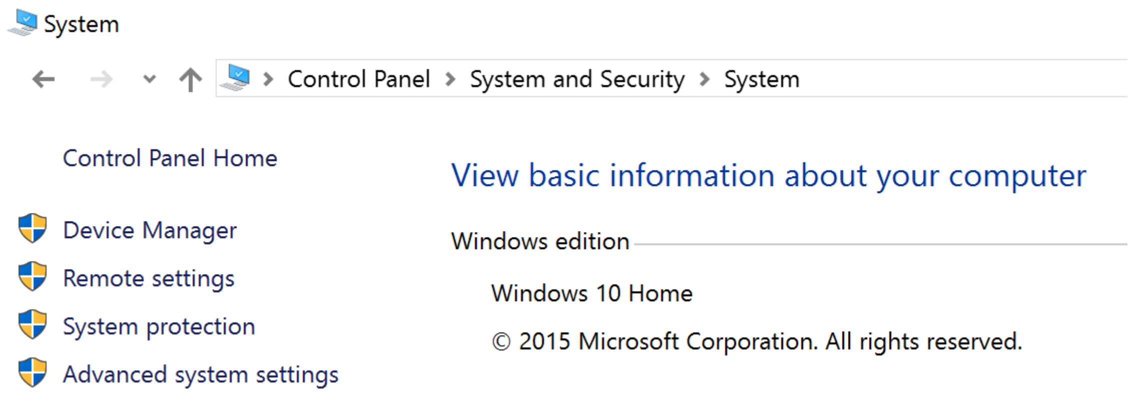
[39] http://www.git-scm.com/
[40] http://windows.php.net/

- On line `:724` uncomment `; extension_dir = "ext"` (Remove `;` from the start of the line)
- On line `:837` under section **Dynamic Extensions** you will find a list of extension. We need to uncomment a few of those:
  - extension=php_curl.dll
  - extension=php_fileinfo.dll
  - extension=php_gd2.dll
  - extension=php_mbstring.dll
  - extension=php_mysqli.dll
  - extension=php_openssl.dll
  - extension=php_pdo_mysql.dll
  - extension=php_pdo_sqlite.dll

Now we have configured PHP for Laravel and other modern PHP applications.

We still have to tell Windows where to find this PHP installation. And we do so by adding the absolute path of the folder we installed PHP to our System Path Environment Variable.

Open Control Panel and go to `Control Panel\System and Security\System\Advanced system settings\Advanced\Environment Variables` and under `System variables` locate `Path` press *Edit*. Here add a new value pointing to your PHP installation folder that contains `php.ini` file. In my case, I would add `C:\tools\php-7.0.3-Win32-VC14-x64`. Press *Ok* to all and close open windows.



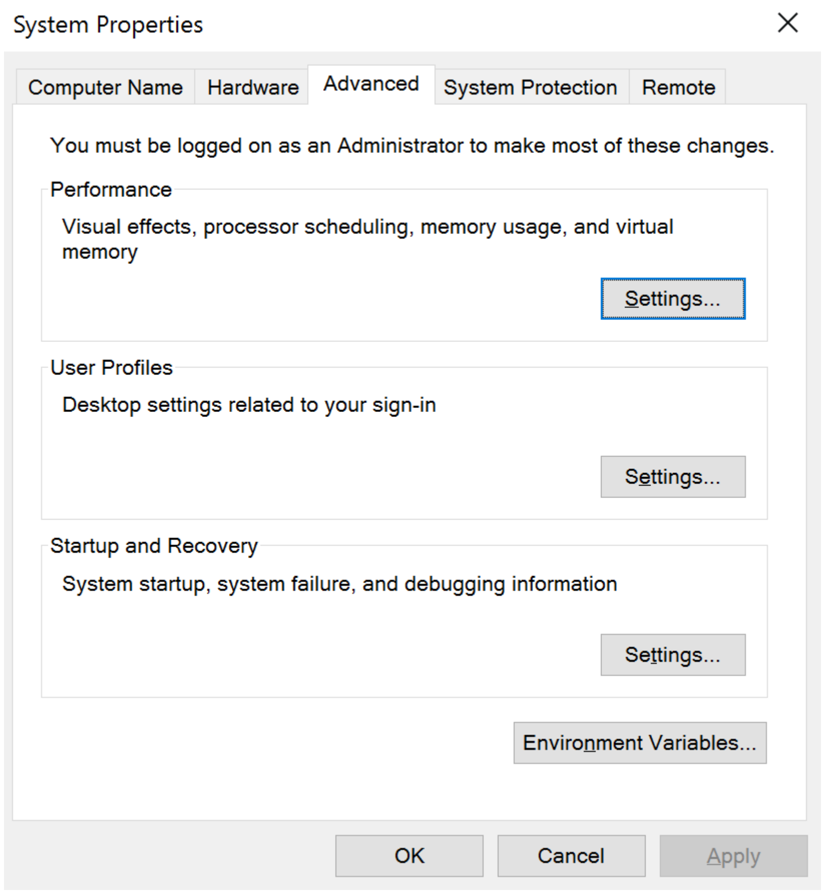
View basic information about your computer

Windows edition

Windows 10 Home

© 2015 Microsoft Corporation. All rights reserved.

Control Panel System



You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

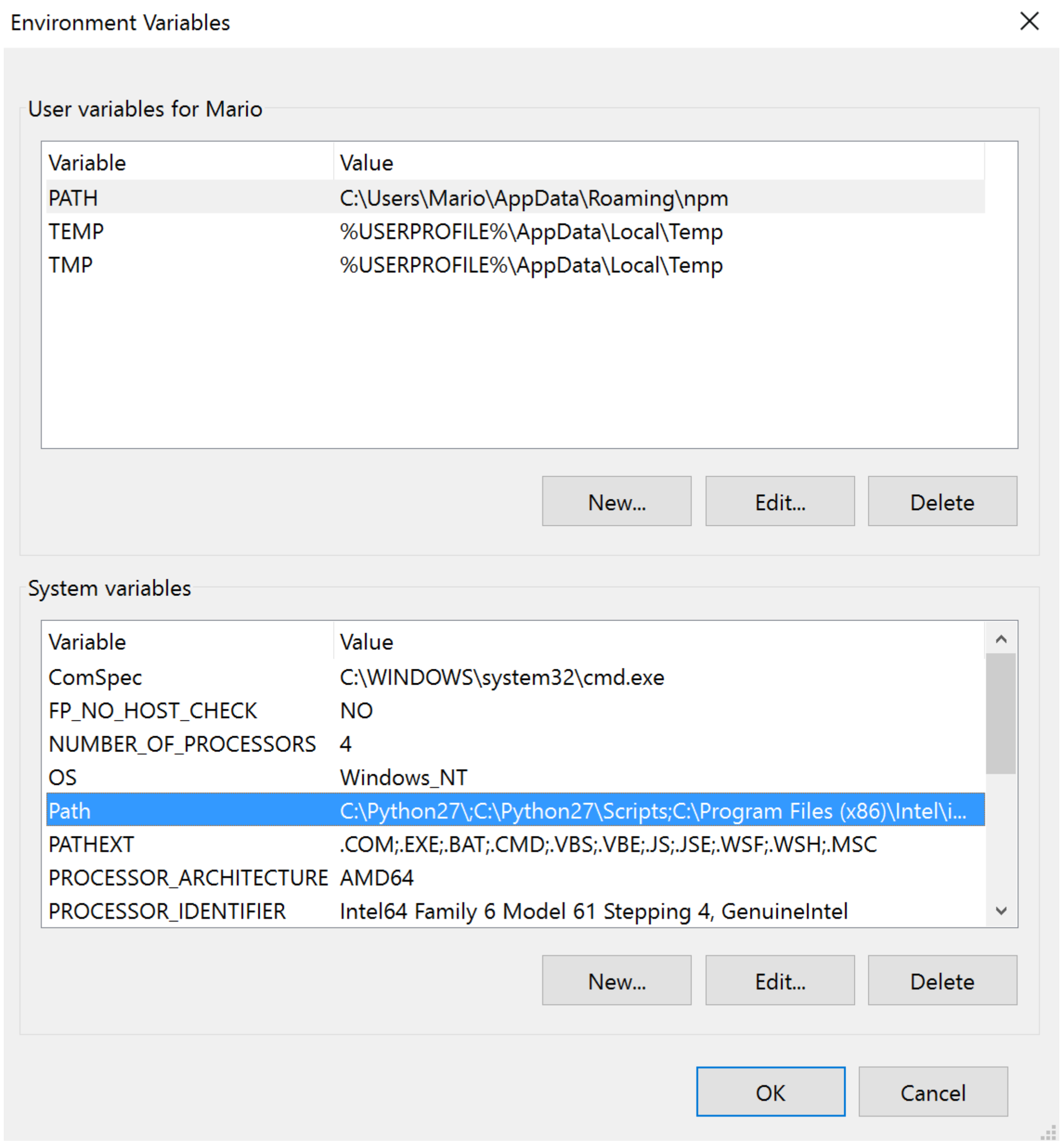
User Profiles

Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

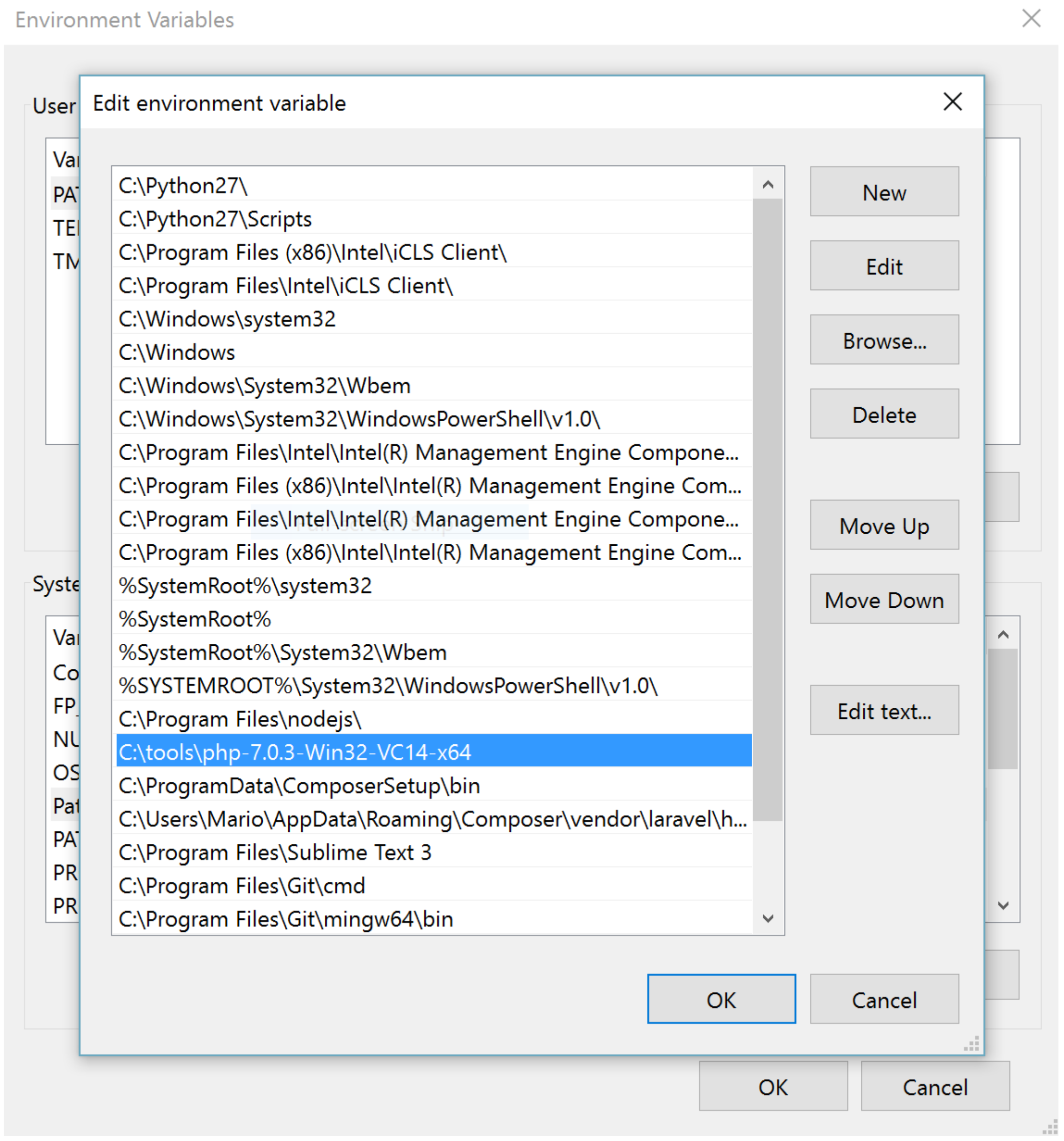
Advanced system properties



Variable	Value
PATH	C:\Users\Mario\AppData\Roaming\npm
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_CHECK	NO
NUMBER_OF_PROCESSORS	4
OS	Windows_NT
Path	C:\Python27\;C:\Python27\Scripts;C:\Program Files (x86)\Intel\i...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 61 Stepping 4, GenuineIntel

Environment variables



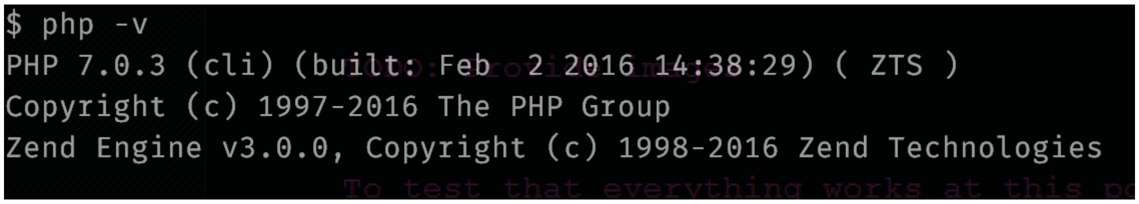
```
C:\Python27\
C:\Python27\Scripts
C:\Program Files (x86)\Intel\iCLS Client\
C:\Program Files\Intel\iCLS Client\
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Program Files\Intel\Intel(R) Management Engine Compone...
C:\Program Files (x86)\Intel\Intel(R) Management Engine Com...
C:\Program Files\Intel\Intel(R) Management Engine Compone...
C:\Program Files (x86)\Intel\Intel(R) Management Engine Com...
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
C:\Program Files\nodejs\
C:\tools\php-7.0.3-Win32-VC14-x64
C:\ProgramData\ComposerSetup\bin
C:\Users\Mario\AppData\Roaming\Composer\vendor\laravel\h...
C:\Program Files\Sublime Text 3
C:\Program Files\Git\cmd
C:\Program Files\Git\mingw64\bin
```

Path system variables

To test that everything works at this point. Open Command Prompt `cmd` and type `php -v`. You should get something like this:

Checking PHP version

```
$ php -v
PHP 7.0.3 (cli) (built: Feb  2 2016 14:38:29) ( ZTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```



```
$ php -v
PHP 7.0.3 (cli) (built: Feb  2 2016 14:38:29) ( ZTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

php version output

Great, now we can move on. If you are having problems at this point, leave me a comment bellow and I will help you out.

Install Composer

Go to the download page for Composer[41] and download the Windows installer. It will install the latest version and configure everything on your system.

You can verify that everything is working by typing `composer --version` in the terminal.

Checking Composer version

```
$ composer --version
Composer version 1.0-dev (72cd6afdfce16f36a9fd786bc1b2f32b851e764f) 2015-12-28 1\
7:35:19
```

Install Git and Git Bash

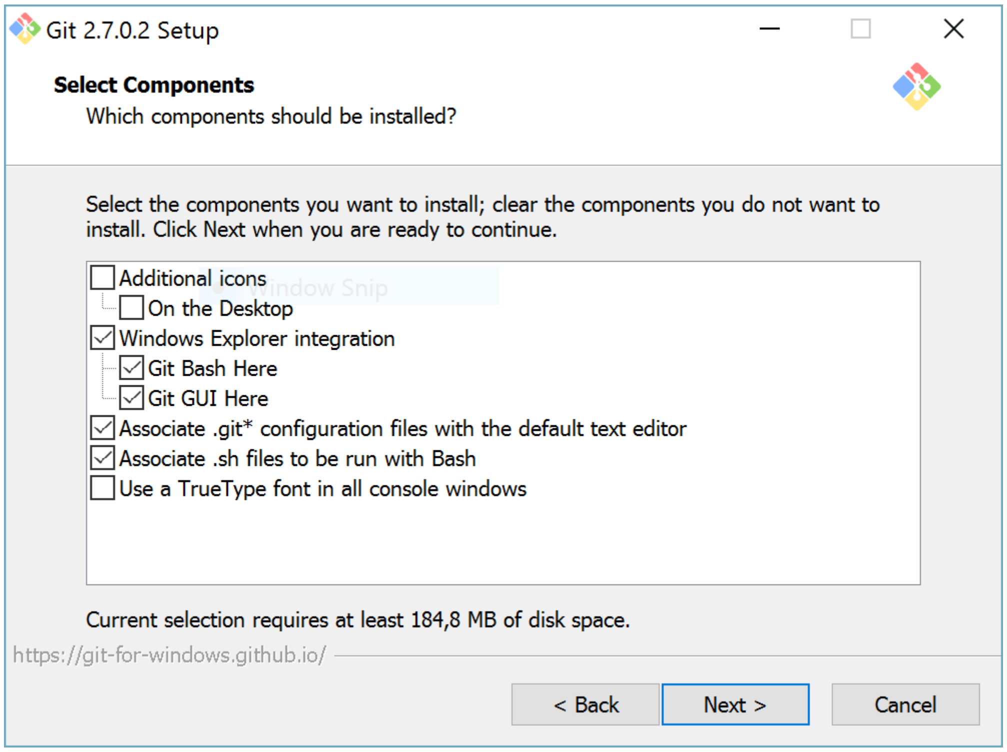
If you are building a modern PHP application or planning to use Laravel you should really learn how to use Git, because without it, over time you will get yourself in a big mess.

The installation is pretty simple, just download the latest version of Git[42] and complete the installation with these options:

- Use MinTTY (the default terminal of MSys2)
- Checkout WIndows-style, commit Unix-style line endings

[41]https://getcomposer.org/download/
[42]http://www.git-scm.com/download/win

- Use Git and optional Unix tools from the Windows Command Prompt
- Enable file system caching



Select Components

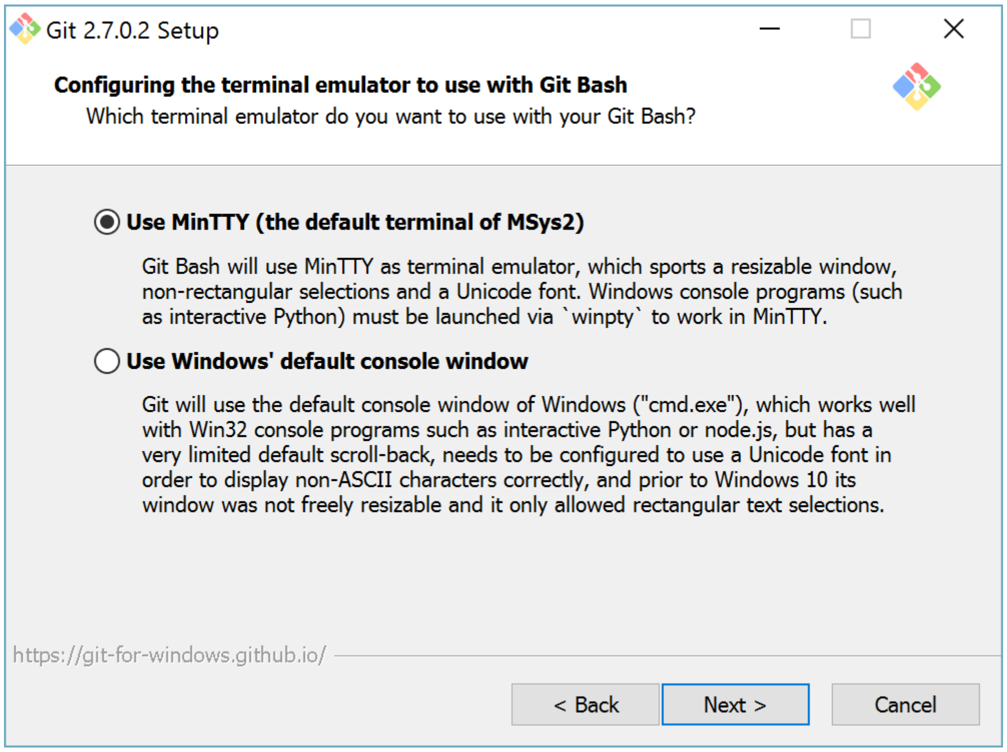
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [ ] Additional icons
  - [ ] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows

Current selection requires at least 184,8 MB of disk space.

Git Components


Git 2.7.0.2 Setup

Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?

[✓ **Use MinTTY (the default terminal of MSys2)**]

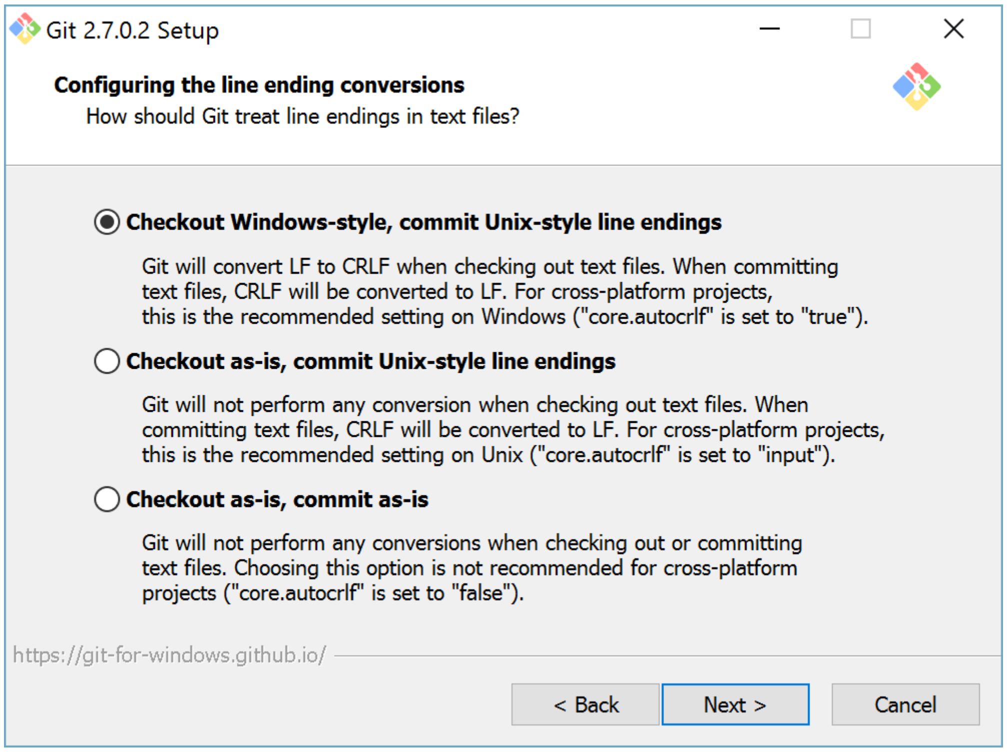
Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

Use Windows' default console window

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

https://git-for-windows.github.io/

Git Terminal Emulator


Git 2.7.0.2 Setup

Configuring the line ending conversions
How should Git treat line endings in text files?

● **Checkout Windows-style, commit Unix-style line endings**

Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

○ **Checkout as-is, commit Unix-style line endings**

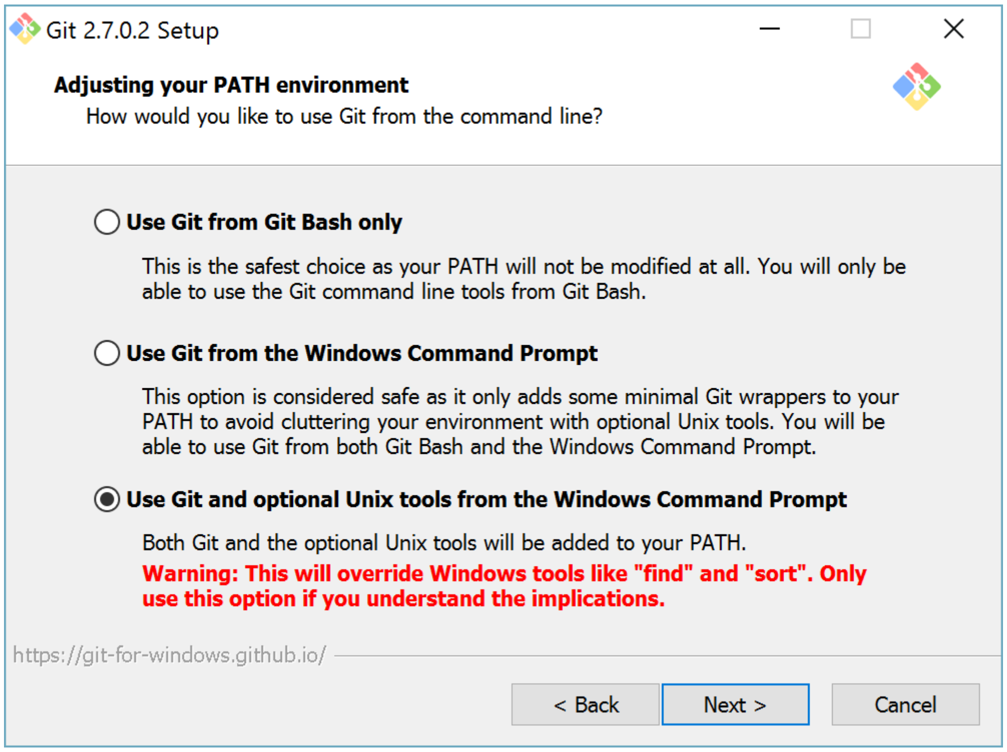
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

○ **Checkout as-is, commit as-is**

Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

https://git-for-windows.github.io/

Git Line ending Checkout style


Git 2.7.0.2 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

( ) **Use Git from Git Bash only**

This is the safest choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

( ) **Use Git from the Windows Command Prompt**

This option is considered safe as it only adds some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from both Git Bash and the Windows Command Prompt.

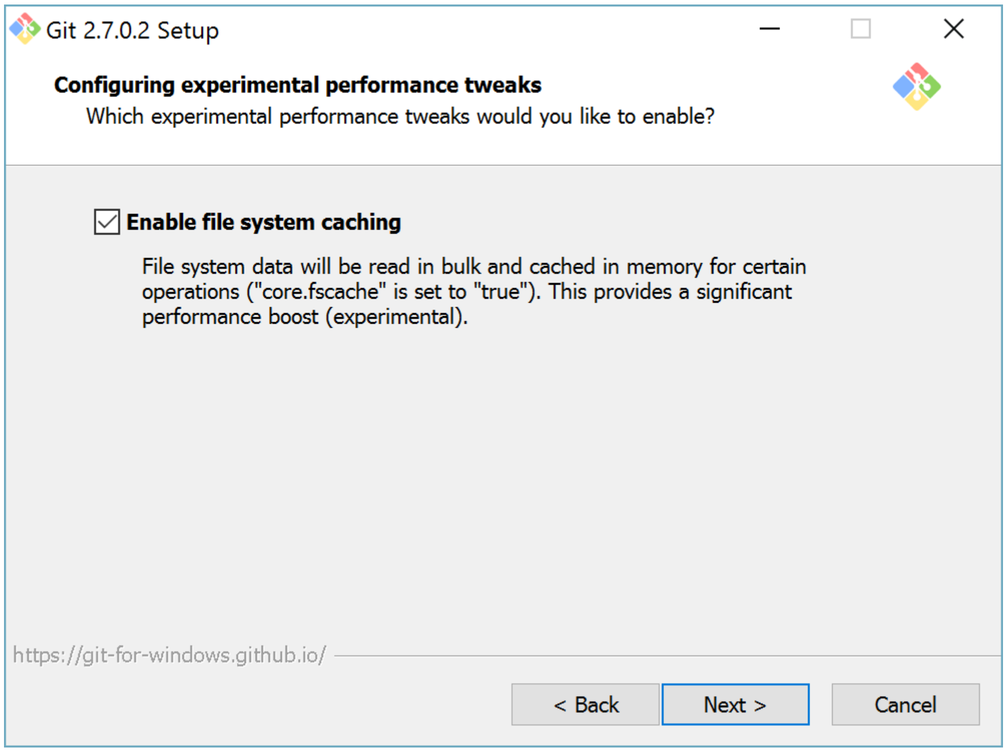
[✓ **Use Git and optional Unix tools from the Windows Command Prompt**]

Both Git and the optional Unix tools will be added to your PATH.

Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

https://git-for-windows.github.io/

Git Environment variables



Git 2.7.0.2 Setup

Configuring experimental performance tweaks

Which experimental performance tweaks would you like to enable?

☑ **Enable file system caching**

File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost (experimental).

https://git-for-windows.github.io/

Git File System Caching

To test that everything is working, find **Git Bash** under Programs, right-click on it and click **Run as Administrator**. Type `git --version` and you should get:

Checking Git version

```
$ git --version
git version 2.7.0.windows.2
```

You can customize the appearance of the terminal by going to options and changing the font family, font size, transparency, full screen and many other options.

> I'm using Fira Code, 11pt, medium transparency, scrollbar turned off and xterm-256color terminal.

Install Node.js

You will be needing node.js to install NPM modules and use Elixir[43].

Go to node.js website[44] and download the latest stable version **(v5.5.0)**. The installation is pretty straight forward, just follow the installer. You can verify your installation by typing `node -v`.

[43] https://laravel.com/docs/5.2/elixir
[44] https://nodejs.org/en/

Repositories location & text editor

Because of path length limitation on Windows, I suggest that you place all of your repositories in the root of your drive `C:\repositories`. *This solves many issues with* `npm`

For you text editor or IDE I suggest using Sublime Text 3[45] or PHPStorm[46]. However, you are free to use anything you want.

You are now ready to move on to the next tutorial.

Important!

When I say “use the terminal” or “type in terminal” in future tutorials, that means to use **Git Bash** program we installed in this tutorial. Everything you do from this point on in terminal, you should be done in **Git Bash** console.

Getting started with Homestead

Your own local virtual server for running PHP applications with lots of extra software in case your projects requires it.

Published at: **12. March, 2016**.

What is Homestead and why all the fuss about it?

Taken from official Laravel documentation[47] on Homestead:

> Laravel Homestead is an official, pre-packaged Vagrant box that provides you a wonderful development environment without requiring you to install PHP, HHVM, a web server, and any other server software on your local machine.

In the simplest way; everything you need (development server related) to start working on your Laravel application is already included in Homestead. You just need to set it up and you are good to go.

Virtualbox provides you with an ability to manage virtual machines. Vagrant is used for automating the virtual machine creation process. Homestead is a Vagrant box. You tell Vagrant to use the Homestead box to create a virtual machine using Virtualbox and voila everything is up and running.

[45]http://www.sublimetext.com/3
[46]https://www.jetbrains.com/phpstorm/
[47]https://laravel.com/docs/5.2/homestead#introduction

Let’s start from the start :)

Important!

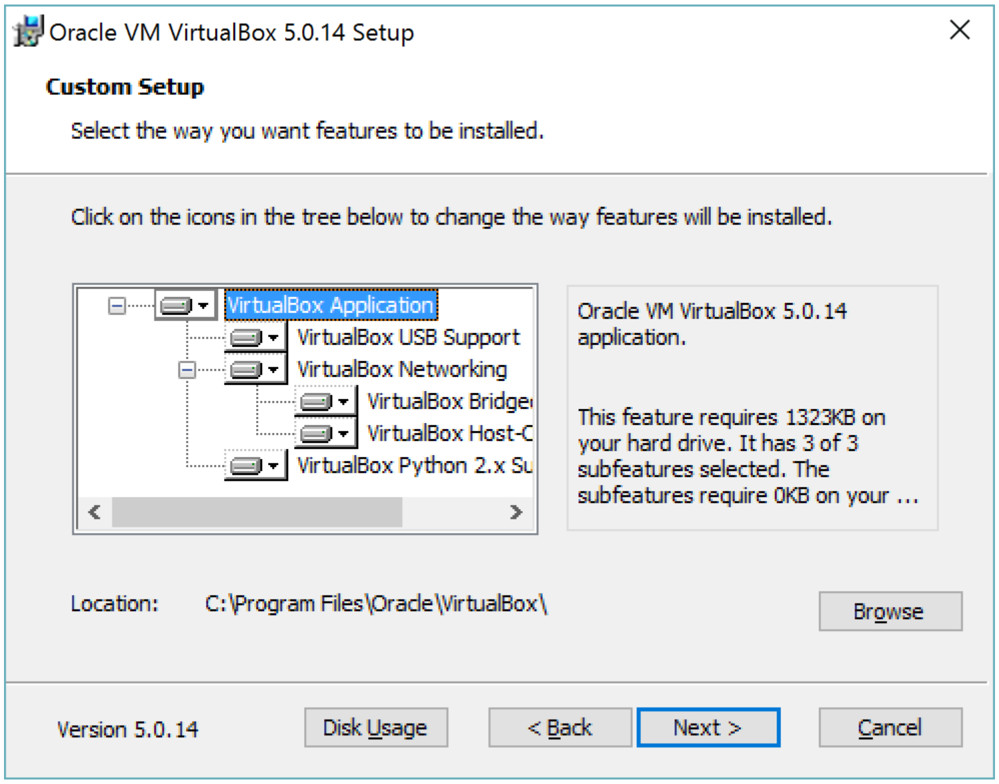
When I say “use the terminal” or “type in terminal”, that means to use **Git Bash** program we installed in the previous tutorial. Everything you do from this point on in terminal, you should be doing in **Git Bash** console.

Install Virtualbox

Visit the official Virtualbox download page and download the latest version (At the time of writing this tutorial the latest version is **5.0.14**). Once downloaded run the setup.

> For reference I’m running Microsoft Windows 10 x64.

Leave all the defaults on this step.


Oracle VM VirtualBox 5.0.14 Setup

Custom Setup

Select the way you want features to be installed.

Click on the icons in the tree below to change the way features will be installed.

- VirtualBox Application
  - VirtualBox USB Support
  - VirtualBox Networking
    - VirtualBox Bridge
    - VirtualBox Host-C
  - VirtualBox Python 2.x Su

Oracle VM VirtualBox 5.0.14 application.

This feature requires 1323KB on your hard drive. It has 3 of 3 subfeatures selected. The subfeatures require 0KB on your ...

Location: C:\Program Files\Oracle\VirtualBox\

Features

And complete the setup by pressing next to everything as usual :) This will disable your network connection for a few seconds so keep that in mind if you are doing something online like reading this tutorial.

Also, be sure to download and install VirtualBox Extension Pack from the same download page. *The extension Pack version must match Virtualbox version.*

Now that we have installed Virtualbox and the Extension Pack we will proceed to Vagrant, but before we do, be sure to restart your PC.

Install Vagrant

Go to the Vagrant download page[48] and download the latest version (**1.8.1**).

Once downloaded, complete the setup by pressing next to everything.

To test that everything is working run `vagrant -v` from the terminal.

Checking Vagrant version

```
$ vagrant -v
Vagrant 1.8.1
```

Install The Homestead Vagrant Box

Run the following command in the terminal to download the latest Homestead box:

Adding Homestead vagrant box

```
vagrant box add laravel/homestead
```

This command should take some time depending on your download speed.

You now have Vagrant installed.

Install Homestead

To install Homestead clone the repository in your Home (~/) directory with the following command:

Cloning Homestead repository

```
cd ~

git clone https://github.com/laravel/homestead.git
```

Navigate to that directory and run `bash init.sh` to create necessary files. You should get an output similar to this:

[48] https://www.vagrantup.com/downloads.html

Initializing Homestead

```
$ ./init.sh

Homestead initialized!
```

This means that the `Homestead.yaml` file has been placed in the `~/.homestead` hidden directory along with two other files (You can open those files with any text editor to see what they are for).

Now open `Homestead.yaml`. This file is the file in which you will be making any future changes. You can:

- change virtual machine settings
- set your SSH key
- add folders
- add Nginx sites
- add databases and more…

For start, change `folders` to point to the directory where you keep your repositories, like so:

Setting the path to your repositories folder

```
folders:
    - map: C:/repositories
      to: /home/vagrant/repositories
```

> There will be more talk about this file later when we will create a blank Laravel application.

Daily usage

In order to avoid navigating to `C:\repositories\homestead` directory every time you want to start the virtual machine, you can add a simple Bash alias to your Bash profile.

Go to `~/` and check if you have a hidden file there called `.bash_profile`. If you don't have that file create it and place the following inside it:

Adding Bash aliases and functions

```
# Some shortcuts for easier navigation & access
alias ..="cd .."
alias vm="ssh vagrant@127.0.0.1 -p 2222"

# Homestead shortcut
function homestead() {
    ( cd /c/repositories/homestead && vagrant $* )
}
```

Save the file and restart your terminal for the changes to take effect.

Now is the time to finally start Homestead. **Now, you must open Git Bash (terminal) as an administrator (Right click -> run as administrator).** Start homestead by typing `homestead up` in your terminal. Vagrant will boot the virtual machine and automatically configure your shared folders and Nginx sites.

Your first Laravel application

Learn how to create a blank Laravel application and serve it locally with Homestead on your PC for development purposes.

Published at: **12. March, 2016.**

To sum things up, so far we have installed the necessary software on our host PC, Virtualbox, Vagrant and got Homestead up and running. Now we will create a blank Laravel application, configure Homestead to serve it and change the `hosts` file so that we have a custom domain for our application.

This process you will have to do every time you create a new Laravel application so try to remember it.

Repositories root directory

As mentioned in the previous tutorial you need to have a repositories folder where you keep all your repositories/applications. In `.homestead/Homestead.yaml` you have a line:

Repositories folder location

```
folders:
    - map: C:/repositories
      to: /home/vagrant/repositories
```

Navigate to that folder and follow the instructions bellow.

Install Laravel via installer

First, we have to download the Laravel installer by typing the command bellow in terminal:

Installing Laravel installer

```
composer global require "laravel/installer"
```

This command will download the Laravel installer and create an executable that you can call upon.

To be able to use `laravel` installer from the terminal we first have to add composer `bin` directory to our path. We do that by going to `Control Panel\System and Security\System\Advanced system settings\Advanced\Environment Variables` and under `System variables` locate `Path` press *Edit*. Here add a new value pointing to the composer `bin` directory. *For reference mine is `C:\Users\Mario\AppData\Roaming\Composer\vendor\bin`.*

To create a directory containing a fresh Laravel installation with all dependencies installed use this command in your repositories root directory:

Creating a new Laravel project

```
laravel new myblog
```

This method of installation is much faster than installing via Composer.

Now you have a directory called `myblog` and inside it, your first Laravel application. **Here are a few tips when working on Windows**. Navigate to your app and open it using Sublime Text like so:

Opening the folder using Sublime Text

```
cd myblog

subl .
```

If your system can't find `subl` you need to add it to your path. The procedure is the same as the above. Add `C:\Program Files\Sublime Text 3` to your `Path`.

Now open the file called `.env`. This file contains all configuration options for your application and by default is not included in version control.

- change `DB_HOST=127.0.0.1` to your Homestead machine IP address. By default that is `192.168.10.10`. This will enable you to run migrations and tinker with your application from the host PC without the need to ssh into Homestead.
- change `DB_DATABASE=homestead` to something more meaningful like `myblog`.

That's it! Now we have to tell Homestead about our brand new application.

Add application to Homestead

To add a new Nginx site to Homestead we need to open `~/.homestead/Homestead.yaml` and add a new site for `myblog.app` and create a database `myblog`:

Adding application to Homestead

```
sites:
    - map: myblog.app
      to: /home/vagrant/repositories/myblog/public

databases:
    - myblog
```

Save the changes and type `homestead provision` in the terminal (remember you must run Git Bash as an Administrator every time your work with Homestead). This command will preserve all changes: sites, databases, custom modifications and update it with new sites and databases. **Very useful**.

There is one more step before you can access your brand new blog and that is adding our custom domain `myblog.app` to point to our Homestead machine IP `192.168.10.10`.

The Hosts file

The `hosts` file will redirect requests for your Homestead sites into your Homestead machine. On Windows, it is located at `C:\Windows\System32\drivers\etc\hosts`. The lines you add to this file will look like the following:

Updating hosts file

```
192.168.10.10  myblog.app
```

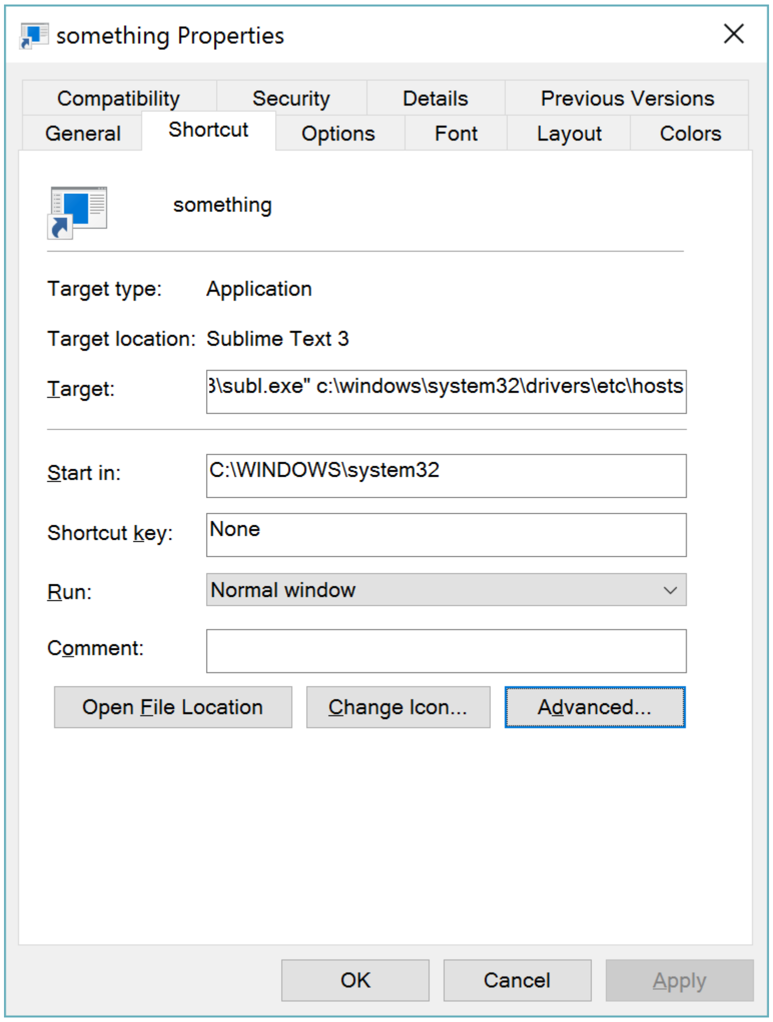
There are a few glitches here. In order to save changes to the `hosts` file, you must open it as an administrator. That means opening the terminal as an administrator navigating to `C:\Windows\System32\drivers\etc` and opening the file `hosts` using Sublime Text. *You will be doing that a lot.*

So to save you and myself time, I have created a shortcut for that.

The shortcut

Go to Desktop and create a new shortcut (right click -> new -> shortcut). When asked for the location of the item paste `"C:\Program Files\Sublime Text 3\subl.exe" c:\windows\system32\drivers\etc\hosts` and press Next. Now type the name for this shortcut `Edit Hosts` and press Finish.

Now right click on the shortcut and go to `Properties -> Shortcut -> Advanced` and check the box saying `Run as Administrator`. This opens the `hosts` file as an administrator so that you can save changes.



Target type: Application

Target location: Sublime Text 3

Target: 3\subl.exe" c:\windows\system32\drivers\etc\hosts

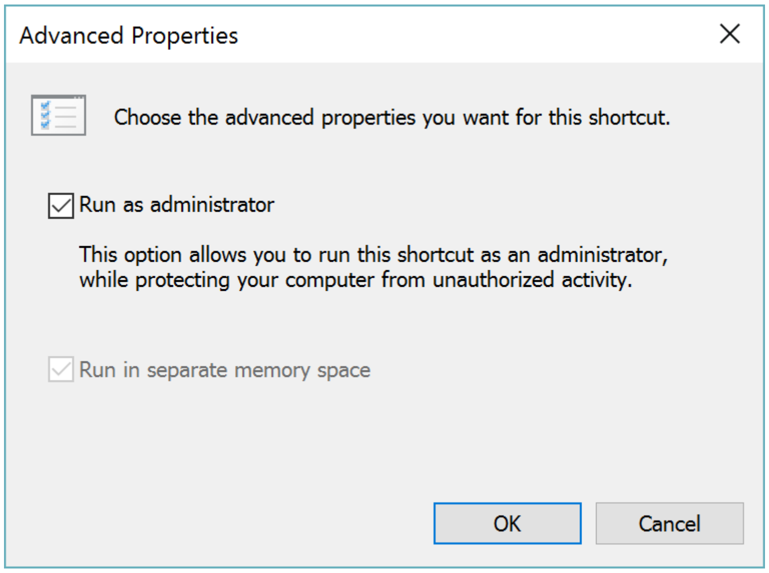
Start in: C:\WINDOWS\system32

Shortcut key: None

Run: Normal window

Comment:

Advanced properties



Choose the advanced properties you want for this shortcut.

[✓] Run as administrator

This option allows you to run this shortcut as an administrator, while protecting your computer from unauthorized activity.

[✓] Run in separate memory space

Run as Administrator

Cut that shortcut (Ctrl+X or right click -> cut) and using the File Explorer navigate to `C:\ProgramData\Microsoft\Wi` `Menu\Programs` and paste it there. It will require you to confirm that you are an administrator.

Once you have done all of this, it is easy now to edit the `hosts` file. Press the Windows key on your keyboard and start typing `Edit Hosts`. The shortcut that we have just created will show. Press Enter and add this line at the bottom:

Updating hosts file

```
192.168.10.10  myblog.app
```

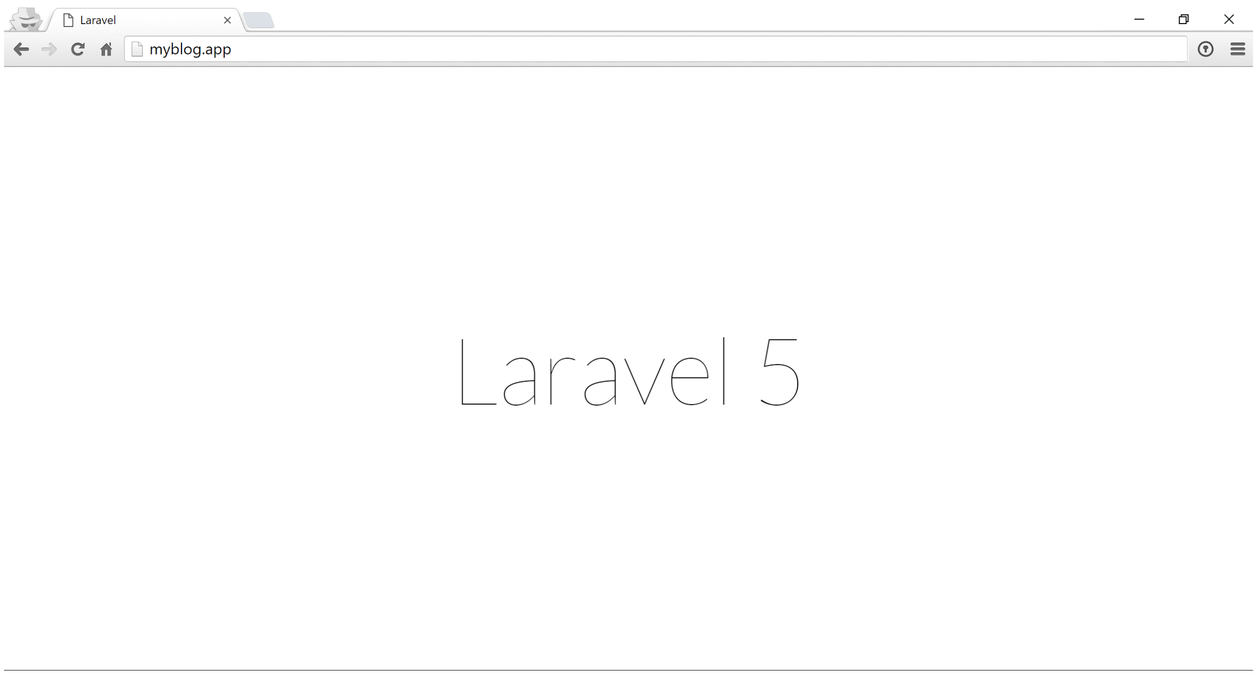
Save the file and you are done.

Sublime Text must be closed before you run this shortcut. Otherwise, if Sublime is opened in normal mode you cannot save changes to the `hosts` file.

Once you have added the domain to your `hosts` file, you can access the site via your web browser:

Accessing the website in the browser

```
http://myblog.app
```



Laravel

Now you have your first Laravel application running on Homestead. This tutorial concludes the course Laravel on Windows.

Laravel on Windows with Laragon

Another approach to getting started with Laravel on Windows is by using Laragon. Laragon offers you a fast, powerful and Isolated Development Environment. It is portable, very flexible and doesn't affect your operating system.

Hello Laragon

In this tutorial, I will tell you about Laragon which is an alternative to Homestead and we will also cover the entire process of installation.

Published at: **04. April, 2016.**

So, what is Laragon and you should use it over Homestead on Windows. In my previous course called Laravel on Windows I've got a few comments saying that I should mention Laragon as an alternative to Homestead for users that are unable to perform the steps in the tutorial (by landjea[49]) and because it is less hassle to set up things natively and having to worry about crap associated with VM's (by Matthew Rath[50]).

I've taken those comments into consideration, took some time to explore Laragon and have come up with this cource where I will explain what Laragon is when you should use it and what are its features.

As stated on the official website[51], Laragon is a fast, powerful and Isolated Development Environment. It is portable and very flexible.

> Installing Laragon is effortless & doesn't affect your OS (Windows). You can move Laragon folder around (to another disk, to another laptop, sync to Cloud,…) and it still works.

To even more simplify this, Laragon is a WAMP (Windows, Apache, MySQL, PHP); Widows web development environment. It does not affect your operating system. You install it as a software, start it up, do your programming and when finished you just exit.

When to use Laragon over Homestead?

[49] https://disqus.com/by/landjea/
[50] https://disqus.com/by/matthew_rath/
[51] https://laragon.org

This is difficult to answer because it depends on many factors. Since Homestead is the officially supported way of running Laravel I would recommend using it, but if for some reason you can't (don't have administrator rights or unable to run a VM) the next best thing is Laragon.

You can always use `php artisan serve` and SQLite database to avoid using Homestead or Laragon if you wish. *This assumes that you have PHP installed on your OS.*

This introduction is long enough, let's move on to the fun stuff.

Features

One thing that I find lacking on the Laragon website is the summary of current features. On the official website, you have the download link and the link to the forum. No install instructions or anything similar can be found on the front page.

It took me some time, to summarize all the features from the Announcements category[52] on the Forum and even more time to find out how to use some features.

> Don't worry, as it turns out it is all very simple.

Software and services that you get with Laragon `1.0.7` are:

- Cmder[53]
- Git
- Node.js
- NPM
- SSH
- Putty
- PHP 7 & 5.6 (Easily switchable with one click)
- Activate/deactivate PHP extensions on the fly
- xDebug
- Composer
- Apache
- MariaDB/MySQL
- phpMyAdmin
- Full Lumen and Laravel support
- Auto create virtual hosts

[52] https://forum.laragon.org/category/1/announcements
[53] http://cmder.net/

- Mail Catcher - Laragon will show a small window on the bottom right of your screen and help you quickly view content of the generated email
- Mail Sender - You can use `mail()` function to send mail to the Internet easily and effortlessly
- Mail Analyzer: Analyze what happens when an email is sent and show helpful information to make sure that your email configurations are correct.
- ngrok - allows connections from the Internet to the local server

Useful shortcuts

Global hotkey to open shell (cmder): `CTRL+ALT+T`

Shell shortcuts:

Useful shortcuts

```
e -> open notepad++
e. -> open explorer
ll -> list current dir with full information
vi -> if you love vim
```

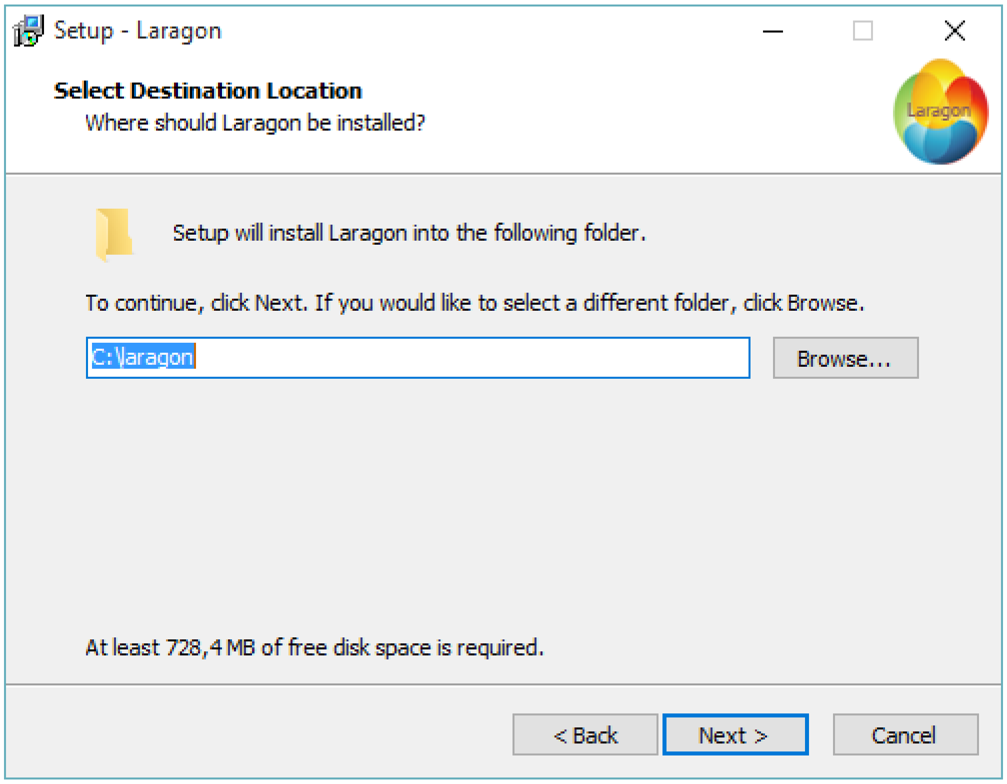
Now that we know what Laragon is and what are its features we can move on to installation.

Installation

The installation is very simple, just click the download button on the official website[54] and follow the installer.

You can choose where to put the Laragon folder (Later you can move this folder where ever you want, but I suggest placing it in the root of any drive):

[54] https://laragon.org



Select Destination Location

Where should Laragon be installed?

Setup will install Laragon into the following folder.

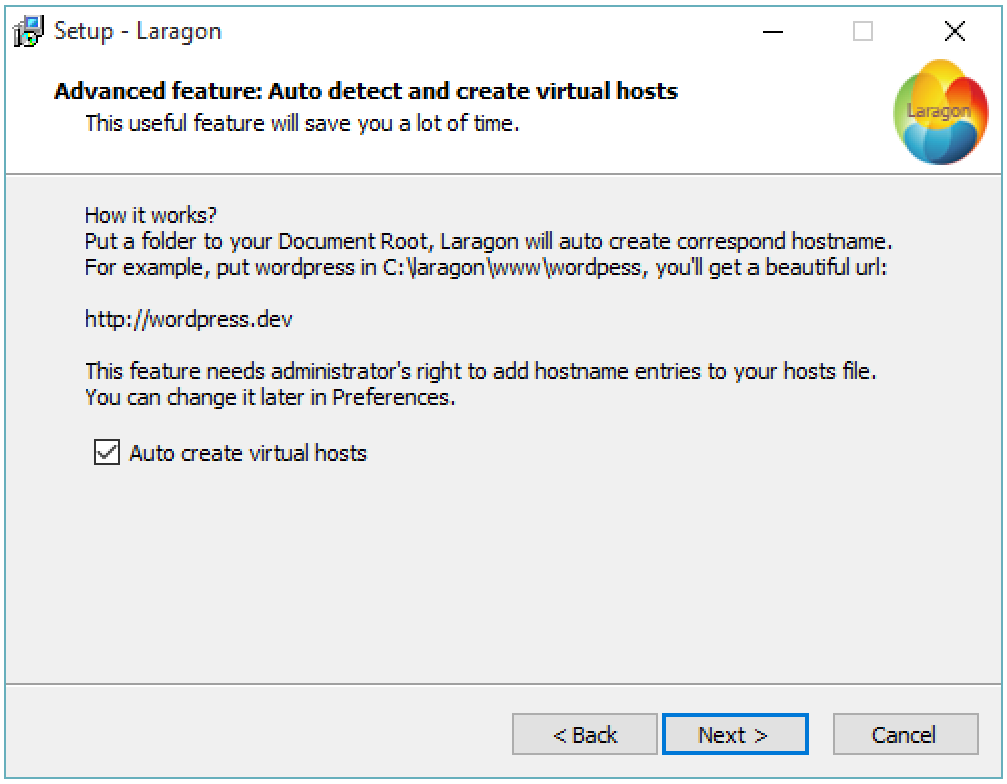
To continue, click Next. If you would like to select a different folder, click Browse.

C:\laragon

At least 728,4 MB of free disk space is required.

Installation Directory

Be sure to enable *Auto create virtual hosts* feature:



Advanced feature: Auto detect and create virtual hosts
This useful feature will save you a lot of time.

How it works?
Put a folder to your Document Root, Laragon will auto create correspond hostname.
For example, put wordpress in C:\laragon\www\wordpress, you'll get a beautiful url:

http://wordpress.dev

This feature needs administrator's right to add hostname entries to your hosts file.
You can change it later in Preferences.

[✓] Auto create virtual hosts

Auto enable virtual hosts

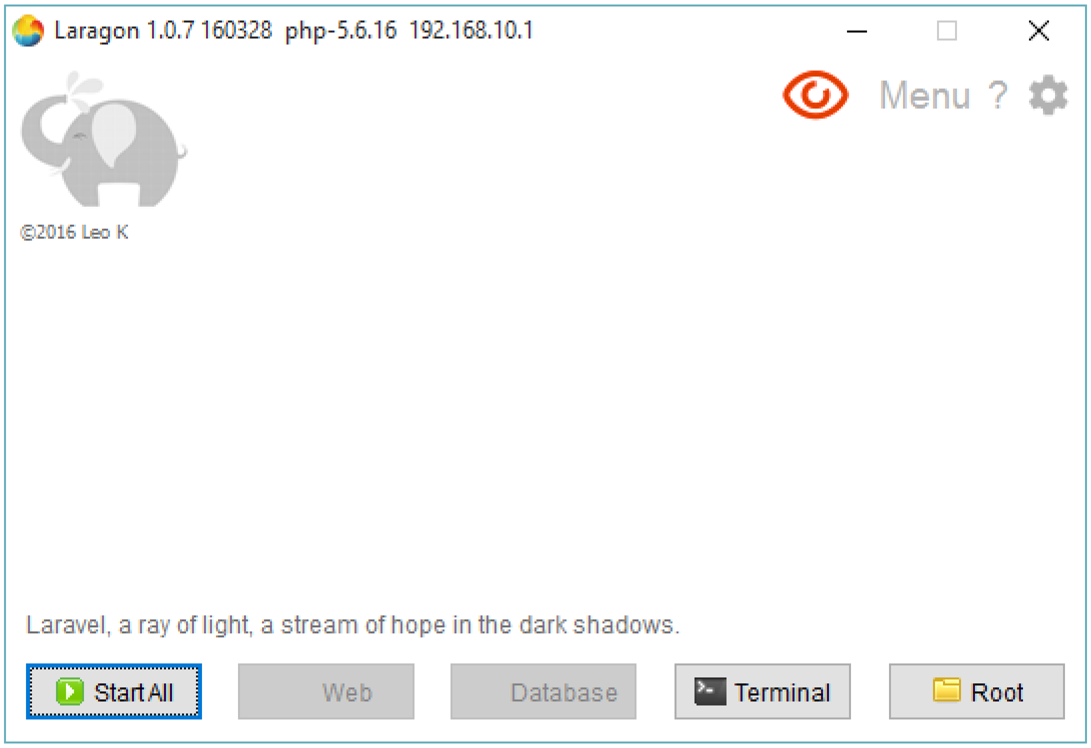
Great, now the installation is complete, but don't start Laragon yet.

Auto create virtual hosts

Once the installation has completed, you have to decide if you want to use the *Auto create virtual hosts* feature or not. If you want to use it, you must **run Laragon as an administrator**. If not, you can run it as a normal user but then that feature will not work.

This feature converts project folder name in `C:\laragon\www\` to a friendly domain name. If your projects folder is called *superawesomewebsite* then Laragon will create a local domain which you can access at `http://superawesomewebsite.dev`

Now let's run Laragon as an administrator. You should see a screen like this:



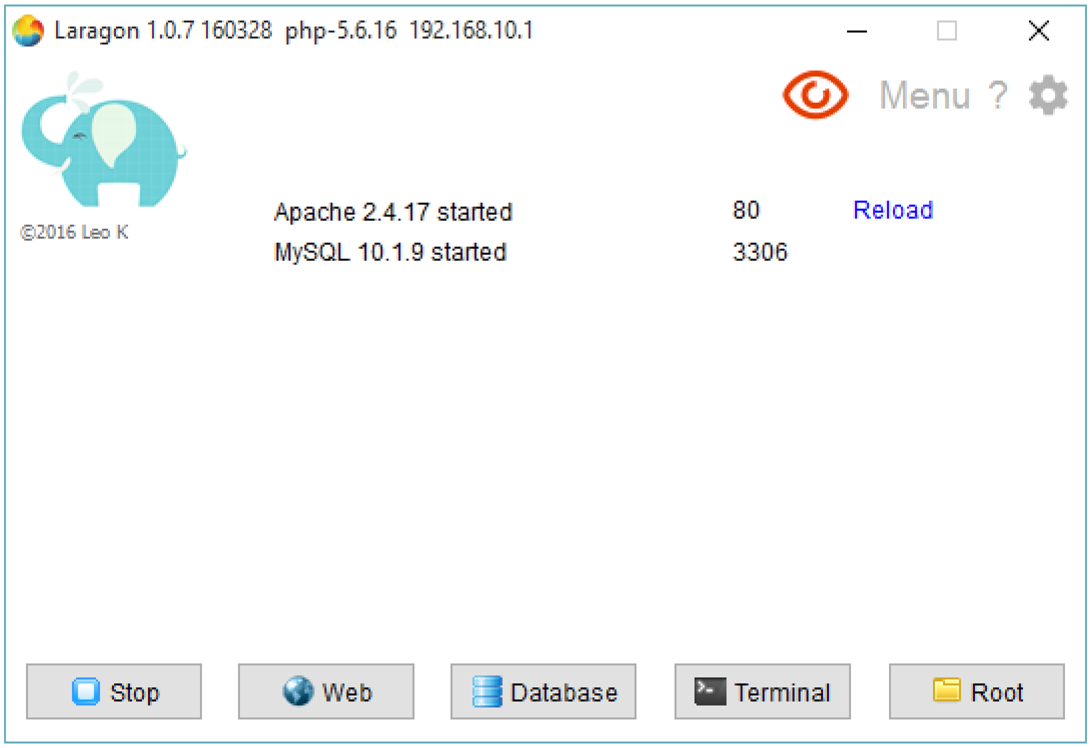
Laravel, a ray of light, a stream of hope in the dark shadows.

Start screen

Switch PHP versions

Before we click on *Start All* button, you can decide which version of PHP you want to be used. Go to `Menu -> PHP -> Version` and choose the one you want. I prefer to use PHP 7 :)

Now click on *Start All* and you should get Apache and MySQL running.


Laragon 1.0.7 160328 php-5.6.16 192.168.10.1

```
Apache 2.4.17 started     80     Reload
MySQL 10.1.9 started      3306
```

All services started

You now have Laragon installed and running.

Your first Laravel application

Learn how to create a blank Laravel application and serve it locally with Laragon on your PC for development purposes.

Published at: **04. April, 2016.**

In the previous tutorial, we have installed Laragon and started its services. *I am running Laragon as an administrator so that the *auto create virtual hosts* feature is enabled*. In this tutorial, I will show you two ways on how to start a brand new Laravel 5 application.

First, using the GUI of Laragon and then using the shell (cmder).

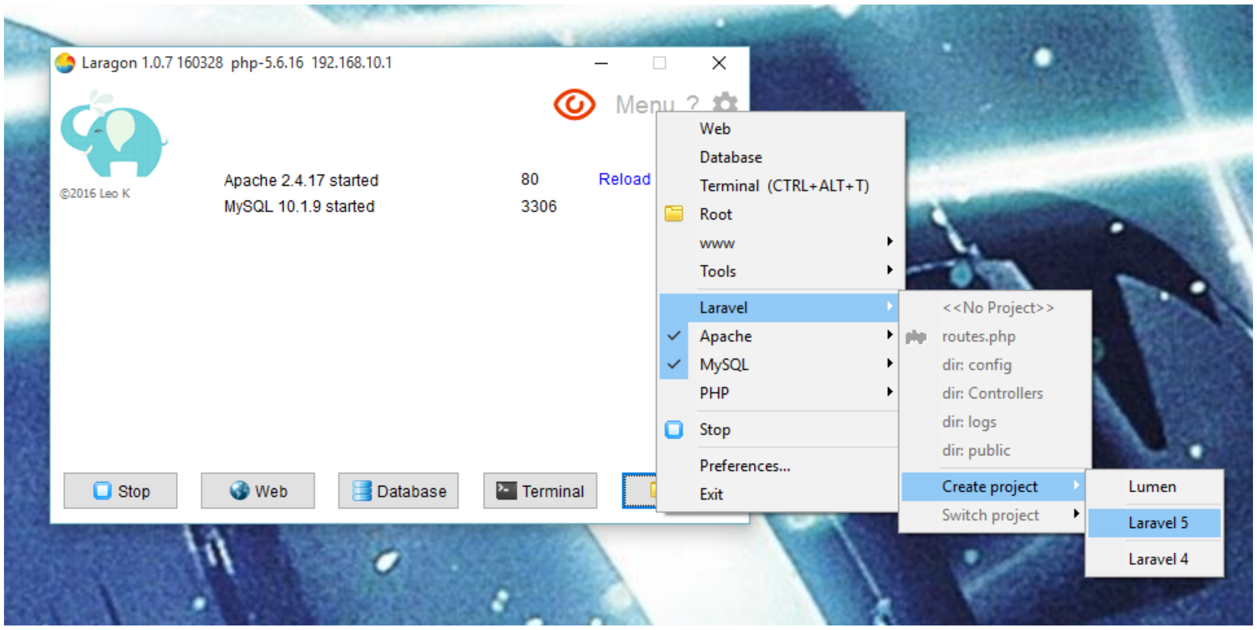
Repositories root directory

To find out where Laragon stores its projects click on *Root* button on the GUI (Graphical User Interface). It will open File Explorer on repositories root directory. You can see the full path in the path bar above: `C:\laragon\www`; if you left the default installation folder of Laragon during the installation.

This is the location where you should place your existing projects if you have any yet.

Install Laravel via GUI

Now, this is the easiest way of installing Laravel 4, 5 or Lumen I have ever seen. Click on `Menu -> Laravel -> Create project -> Laravel 5`.



Install Laravel 5

You will be asked for the project name. Enter the name and confirm. A command line window will open and the installation of Laravel will begin.

I have entered `laravel5` as project name, but you can choose how you want to name your projects. In future reference, you can replace `laravel5` with the project name you entered.

Once completed you will see a message:

Laragon message

```
Run Laragon as Administrator to get beautiful URL:
http://laravel5.dev"
```

Before you visit that URL, be sure to press *reload* on Laragon GUI for changes to take effect. If you visit the URL `http://laravel5.dev` in your browser you will see a welcome screen of Laravel 5.

Laravel 5 Welcome

That's it! You project is now located in `C:\laragon\www\laravel5` folder. You can use any text editor or IDE to open it and start working on your brand new Laravel 5 application.

Install Laravel via shell

If for some reason you don't want to use the GUI to create a Laravel 5 project, you can always use the shell aka Terminal.

On the Laragon GUI click on *Terminal* to open the Cmder shell which points to your repositories root directory. **Or** there is a global keyboard shortcut mentioned in the previous tutorial `CTRL+ALT+T` which opens the same Cmder shell.

In shell type:

Installing Laravel project using Composer

```
composer create-project laravel/laravel your-project-name --prefer-dist
```

to create a new Laravel project. *This is mentioned in the official Laravel documentation under Installation*[55]*.*

Now you have your first Laravel application running on Laragon.

Remote access using ngrok

Enable your friends and clients to access your application over the Internet, while working on it locally.

Published at: **18. April, 2016**.

Let's dive right into this tutorial. *What do we want to do?*

We want to expose our local application to the Internet so that our clients or friends can view what we are working on or the current progress of the project while it is still not on production server.

> Secure tunnels to localhost - ngrok[a]
>
> [a]https://ngrok.com/

Some of the benefits of using ngrok from their website:

- Demo without deploying
- Simplify mobile device testing
- Build webhook integrations with ease
- Run personal cloud services from your own private network

  This is a really nice feature that Laragon has out-of-the-box.

Remote Access

Let's continue where we left off in the previous tutorial. We have created a blank Laravel application on `http://laravel5.dev` domain and all Laragon services are running.

On the Laragon GUI click on *Terminal* to open the Cmder shell **Or** use the shortcut `CTRL+ALT+T` which opens the same Cmder shell.

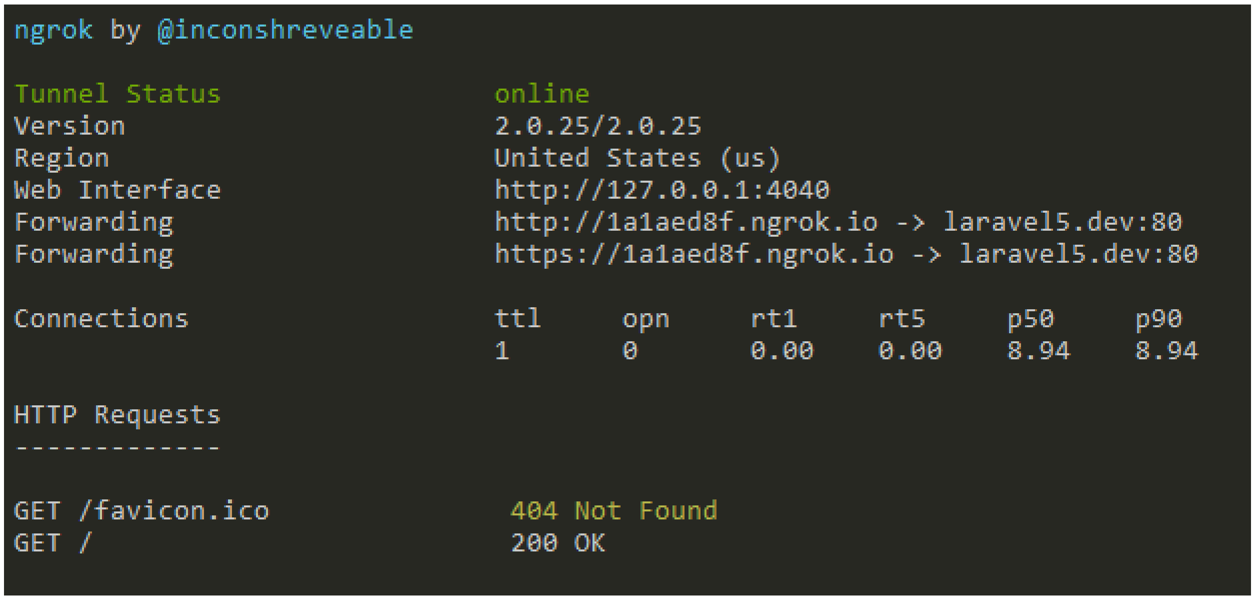
In shell type:

[55]https://laravel.com/docs/5.2/installation

Enabling remote access using ngrok

```
ngrok http laravel5.dev:80
```

You will get an output similar to this one:



```
ngrok by @inconshreveable

Tunnel Status                 online
Version                       2.0.25/2.0.25
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://1a1aed8f.ngrok.io -> laravel5.dev:80
Forwarding                    https://1a1aed8f.ngrok.io -> laravel5.dev:80

Connections                   ttl     opn     rt1     rt5     p50     p90
                              1       0       0.00    0.00    8.94    8.94

HTTP Requests
-------------

GET /favicon.ico               404 Not Found
GET /                          200 OK
```

ngrok output

Take a note of this line:

Obtaining remote access URL

```
Forwarding http://1a1aed8f.ngrok.io -> laravel5.dev:80
```

In my case this is `http://1a1aed8f.ngrok.io`, but your output will be different, so keep that in mind and write it down somewhere. You will need it for the next step.

Now go to the Laragon GUI and click on `Menu -> Apache -> http-vhosts.conf` and add the domain from ngrok to `ServerAlias` of `laravel5.dev` virtualhost entry.

Keep in mind that if you have named your project differently than *laravel5*, use your project name instead.

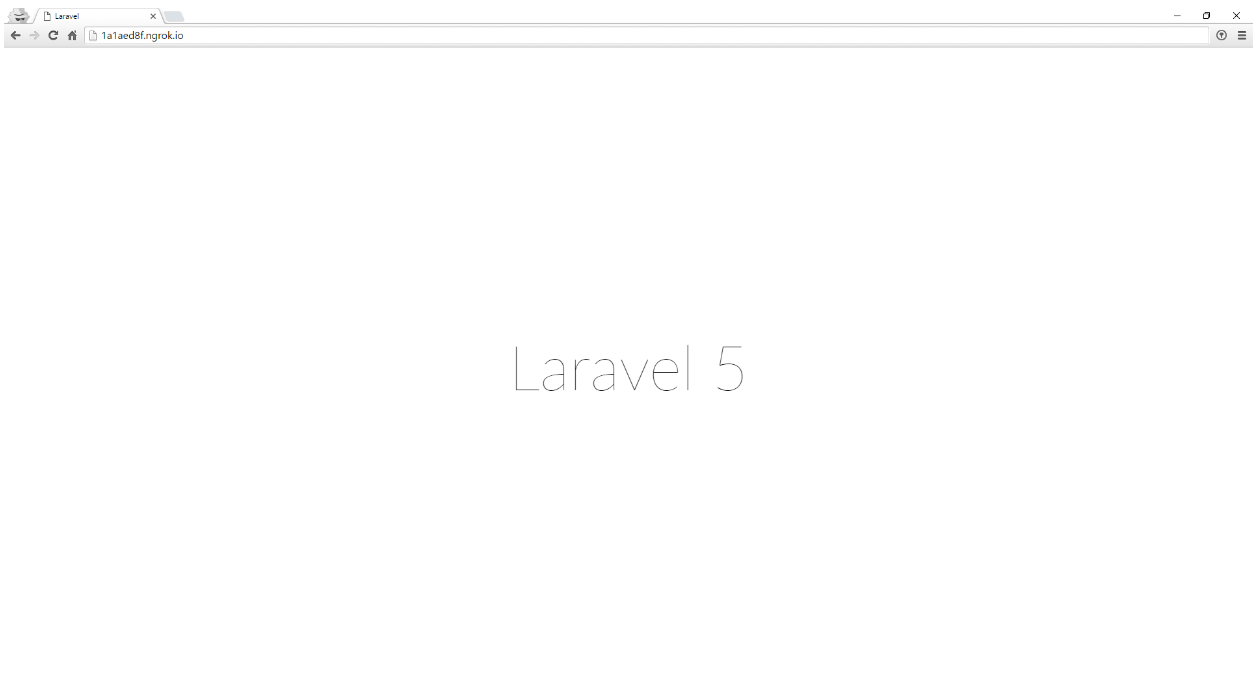
Now my virtual host entry looks like this:

Virtual host entry for Apache

```
<VirtualHost *:80> #laragon magic!
    DocumentRoot "C:/laragon/www/laravel5/public/"
    ServerName laravel5.dev
    ServerAlias *.laravel5.dev 1a1aed8f.ngrok.io
</VirtualHost>
```

One last thing before this starts working is to restart Apache. You do that by going to the Laragon GUI and clicking on `Menu -> Apache -> Reload` or just click on `Reload` on the GUI screen next to the Apache service.

You can now access your local application over the Internet using the given URL from ngrok. In my case, this is `http://1a1aed8f.ngrok.io`.



ngrok remote

The URL will be changed each time you run ngrok.

Quickstart

1. Run ngrok for project `ngrok http project.dev:80`.

2. Add URL given from ngrok to ServerAlias of that project in Apache `http-vhosts.conf` file (`Menu -> Apache -> http-vhosts.conf`).
3. Reload Apache (`Menu -> Apache -> Reload`).
4. Send given ngrok URL to friend, client etc…

This was a nice and simple tutorial on a thing that is usually very complicated to do, but thanks to Laragon it was a breeze.

Configure Laravel 5 for Shared Hosting

There are a few things that you have to change in Laravel to make it work on shared hosting. The most important thing is to change the public path and correctly bootstrap the application.

Published at: **29. July, 2016.**

View Source Code

Source code for this tutorial is available here[56].

I recently had to build a website that would be used on shared hosting. At first, I went with a standard HTML `index.html` file and then create a file called `contact.php` to handle the contact form submit action. The website can be found at studio-renata.hr[57] as soon as they move the website to the server that has PHP 5.6 version.

As I started coding the website the number of lines started to grow and I had a lot of duplicating syntax for stuff like images, containers etc.. Somehow I got over it and was feeling happy with it, but then I had to deal with validating the contact form using AJAX. As you can guess I did a quick research and found out that Laravel can run on shared hosting, but with a few limitations:

- There is no console (artisan)
- You cannot use Composer to install/update
- You cannot use Git to version your application

That being said, there are a few requirements that your server needs to have in order for Laravel to work:

- PHP >= 5.5.9
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension

[56] https://github.com/laravelista/configure-laravel-5-for-shared-hosting
[57] http://studio-renata.hr

> To be honest, I haven't even checked that my shared hosting server has mentioned PHP extensions, but since it all works I guess that it does.

In this tutorial, I will show you how to configure your Laravel application to be able to run on shared hosting. You can view the code for this tutorial on GitHub[58].

Moving Files

I have a fresh installation of Laravel on my PC. To see how to install Laravel see the official documentation[59] or check out these great courses on my website for Homestead on Windows or Laragon.

View changes in this commit

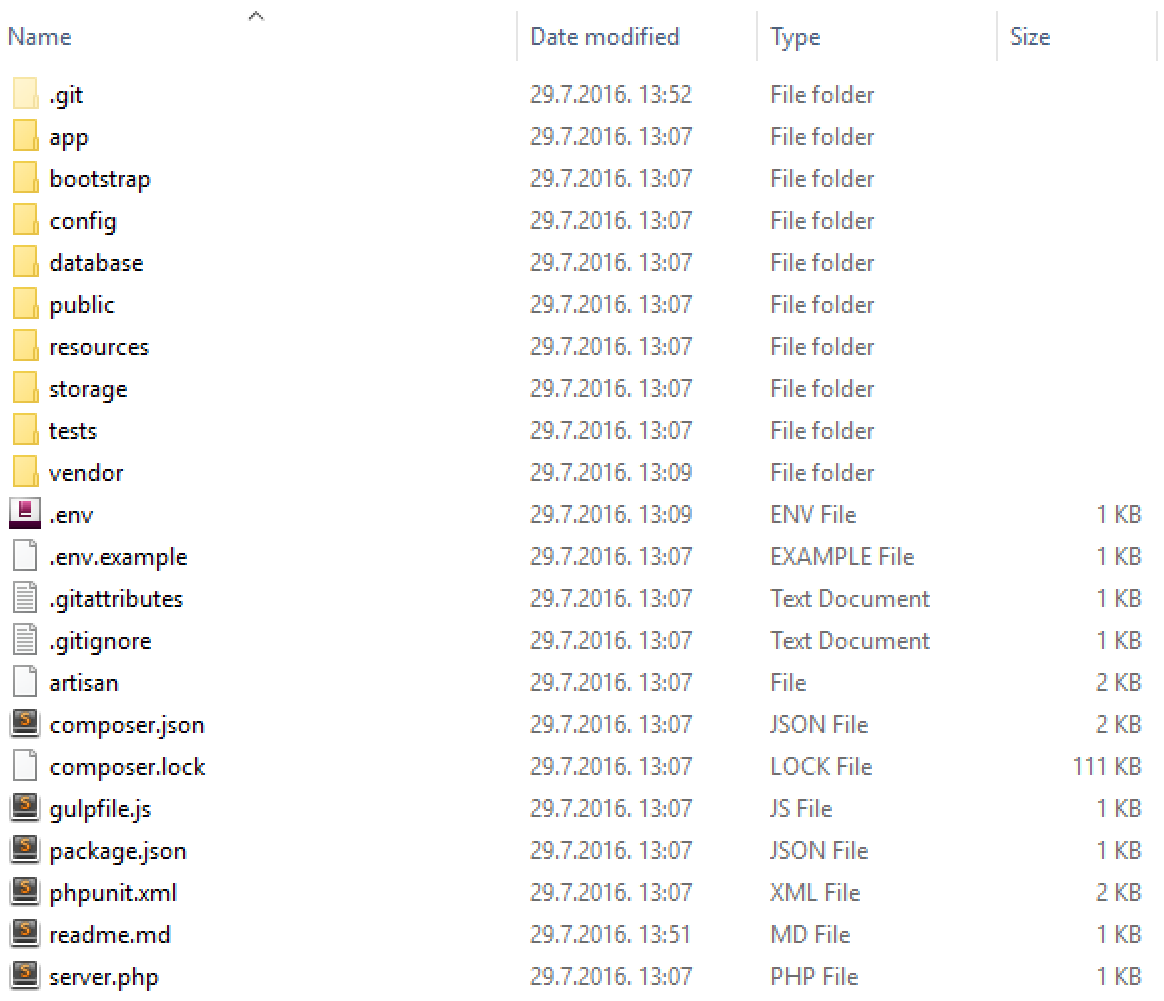
0546e265b7187312bb360b93edbfc91a8e969942[60].

First, we will organize our application folder structure to better match our needs. The default folder structure looks like this:

[58] https://github.com/laravelista/configure-laravel-5-for-shared-hosting

[59] https://laravel.com/docs/5.2/installation

[60] https://github.com/laravelista/configure-laravel-5-for-shared-hosting/commit/0546e265b7187312bb360b93edbfc91a8e969942



Name	Date modified	Type	Size
.git	29.7.2016. 13:52	File folder	
app	29.7.2016. 13:07	File folder	
bootstrap	29.7.2016. 13:07	File folder	
config	29.7.2016. 13:07	File folder	
database	29.7.2016. 13:07	File folder	
public	29.7.2016. 13:07	File folder	
resources	29.7.2016. 13:07	File folder	
storage	29.7.2016. 13:07	File folder	
tests	29.7.2016. 13:07	File folder	
vendor	29.7.2016. 13:09	File folder	
.env	29.7.2016. 13:09	ENV File	1 KB
.env.example	29.7.2016. 13:07	EXAMPLE File	1 KB
.gitattributes	29.7.2016. 13:07	Text Document	1 KB
.gitignore	29.7.2016. 13:07	Text Document	1 KB
artisan	29.7.2016. 13:07	File	2 KB
composer.json	29.7.2016. 13:07	JSON File	2 KB
composer.lock	29.7.2016. 13:07	LOCK File	111 KB
gulpfile.js	29.7.2016. 13:07	JS File	1 KB
package.json	29.7.2016. 13:07	JSON File	1 KB
phpunit.xml	29.7.2016. 13:07	XML File	2 KB
readme.md	29.7.2016. 13:51	MD File	1 KB
server.php	29.7.2016. 13:07	PHP File	1 KB

Default folder structure

We will rename folder `/public` to `public_html`

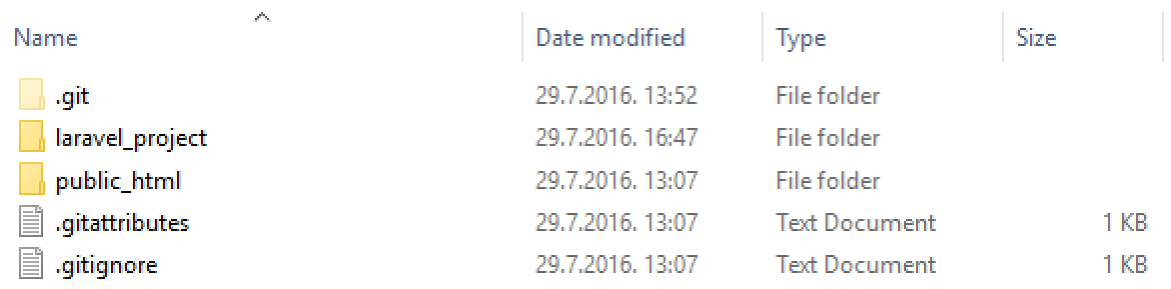
> You can rename it to whatever you need, but remember to use that name in future code changes in this tutorial.

and create a new directory called `laravel_project`.

> You can name this folder whatever you want, but remember to use it in future code references.

Move all files and folders to `laravel_project` directory except `public_html/` and `.git`(if you have created it) folder.

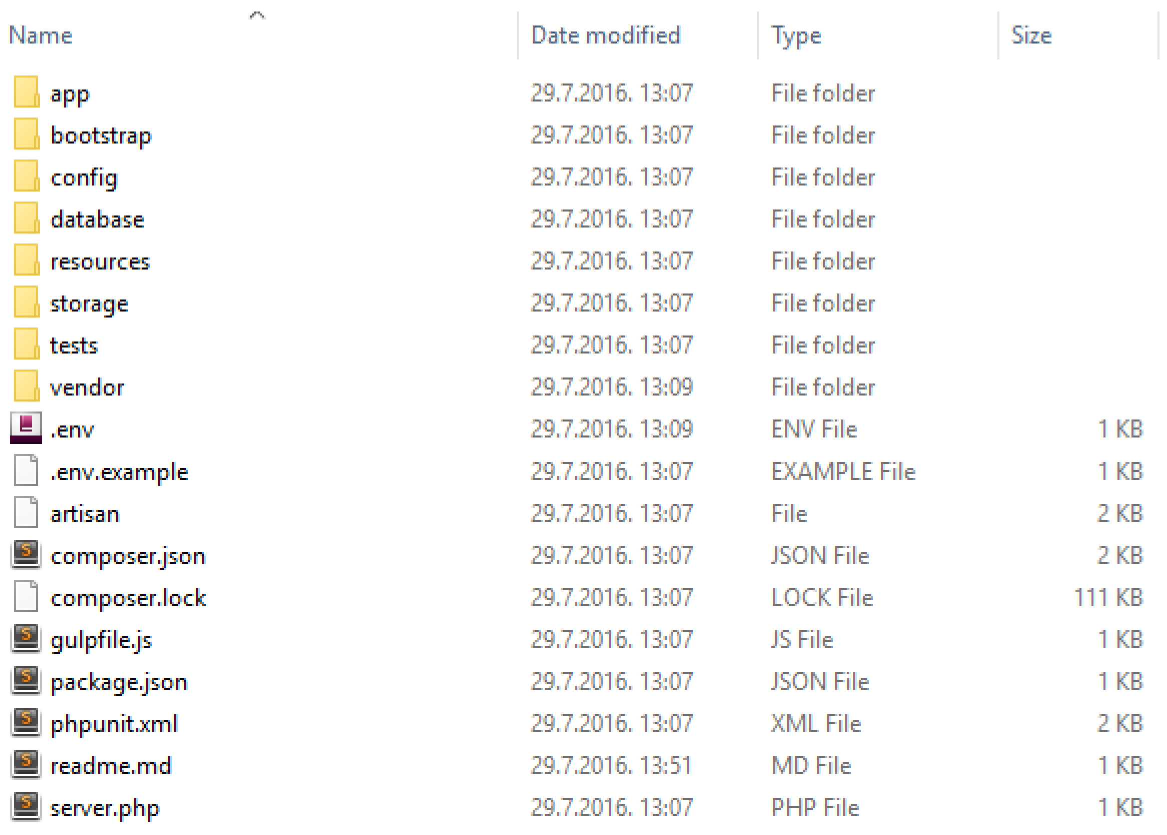
Now, your root folder structure should look like this:



Name	Date modified	Type	Size
.git	29.7.2016. 13:52	File folder	
laravel_project	29.7.2016. 16:47	File folder	
public_html	29.7.2016. 13:07	File folder	
.gitattributes	29.7.2016. 13:07	Text Document	1 KB
.gitignore	29.7.2016. 13:07	Text Document	1 KB

New folder structure 1

and inside `laravel_project` should look like this:



Name	Date modified	Type	Size
app	29.7.2016. 13:07	File folder	
bootstrap	29.7.2016. 13:07	File folder	
config	29.7.2016. 13:07	File folder	
database	29.7.2016. 13:07	File folder	
resources	29.7.2016. 13:07	File folder	
storage	29.7.2016. 13:07	File folder	
tests	29.7.2016. 13:07	File folder	
vendor	29.7.2016. 13:09	File folder	
.env	29.7.2016. 13:09	ENV File	1 KB
.env.example	29.7.2016. 13:07	EXAMPLE File	1 KB
artisan	29.7.2016. 13:07	File	2 KB
composer.json	29.7.2016. 13:07	JSON File	2 KB
composer.lock	29.7.2016. 13:07	LOCK File	111 KB
gulpfile.js	29.7.2016. 13:07	JS File	1 KB
package.json	29.7.2016. 13:07	JSON File	1 KB
phpunit.xml	29.7.2016. 13:07	XML File	2 KB
readme.md	29.7.2016. 13:51	MD File	1 KB
server.php	29.7.2016. 13:07	PHP File	1 KB

New folder structure 2

View changes in this commit

6dec090ae1ca11673ee15d7898280becebd46038[61].

[61]https://github.com/laravelista/configure-laravel-5-for-shared-hosting/commit/6dec090ae1ca11673ee15d7898280becebd46038

Update Bootstrap Files Location

Since `index.php` file inside `public_html/` directory is the first file that will be loaded by the HTTP server, we have to adjust the paths to `autoload.php` and `app.php` files located in `/laravel_project/bootstrap/`, to match our new folder structure.

Open file `public_html/index.php` and change:

- **line 22** `require __DIR__.'/../bootstrap/autoload.php';` to `require __DIR__.'/../laravel_project/bootstrap/autoload.php';`
- **line 36** `$app = require_once __DIR__.'/../bootstrap/app.php';` to `$app = require_once __DIR__.'/../laravel_project/bootstrap/app.php';`

That is all that you have to change in `public_html` folder.

View changes in this commit

158c72d065db262ff7139bac9f51178cc5a082a1[62].

Change public_path in Application

So, this party at first seems very scary, but it really isn't. We have created a new class which extends the main Laravel application class and overwrite the `publicPath` method to return the path to our new `public_html` folder. Then we use that class to create a new Laravel application. *This way our `public_path` works everywhere in our application including console (which you can still use in development).*

Create a new file in `laravel_project/app/` called `Application.php` and paste the following code inside:

Extending the Application class

```php
<?php namespace App;

/**
 * Extend Laravel main application class to change public_path.
 *
 * See here for more info:
 * http://stackoverflow.com/questions/31758901/laravel-5-change-public-path
 */
```

[62] https://github.com/laravelista/configure-laravel-5-for-shared-hosting/commit/158c72d065db262ff7139bac9f51178cc5a082a1

```
class Application extends \Illuminate\Foundation\Application
{

    /**
     * Get the path to the public / web directory.
     *
     * @return string
     */
    public function publicPath()
    {
        return $this->basePath . DIRECTORY_SEPARATOR . '..' . DIRECTORY_SEPARATO\
R . 'public_html';
    }

}
```

Now in `laravel_project/bootstrap/app.php` change:

```
$app = new Illuminate\Foundation\Application(
    realpath(__DIR__.'/../')
);
```

to

Using the new Application class

```
$app = new App\Application(
    realpath(__DIR__.'/../')
);
```

This instantiates a new Laravel application using our `Application` class. Before we can tests if our new folder structure works, there is one more step to complete if you plan on using `php artisan serve` to test the site.

View changes in this commit

ee254ae1497c739e930c94cd6b85a2c5e25ecf10[63].

Make `php artisan serve` work

This is very simple. Open file `laravel_project/server.php` and change:

[63] https://github.com/laravelista/configure-laravel-5-for-shared-hosting/commit/ee254ae1497c739e930c94cd6b85a2c5e25ecf10

```
if ($uri !== '/' && file_exists(__DIR__.'/public'.$uri)) {
    return false;
}

require_once __DIR__.'/public/index.php';
```

to

Updating the path to index.php

```
if ($uri !== '/' && file_exists(__DIR__.'/../public_html'.$uri)) {
    return false;
}

require_once __DIR__.'/../public_html/index.php';
```

Save changes and run

Serving the application

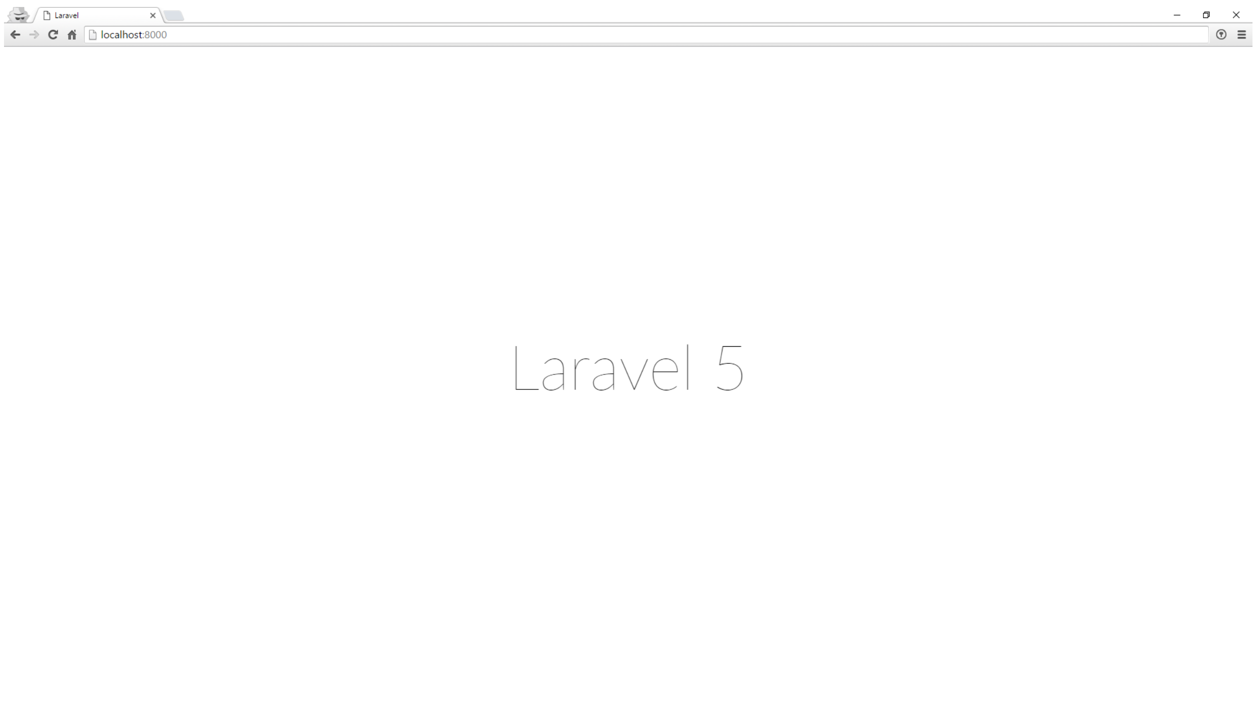
```
cd laravel_project
php artisan serve
```

You should get output similar to this one:

Development server started

```
$ php artisan serve
Laravel development server started on http://localhost:8000/
```

If you open your browser on `http://localhost:8000` you should see a standard Laravel Hello page.



Hello Laravel

You can now compress your application, upload it to your shared hosting and extract. It should work now.

View changes in this commit

81fd608afe4b551b480640848c28804a5dcc2005[64].

This concludes this tutorial.

[64]https://github.com/laravelista/configure-laravel-5-for-shared-hosting/commit/81fd608afe4b551b480640848c28804a5dcc2005