SOUTHERN LEYTE
STATE UNIVERISTY
BONTOC

San Ramon, Bontoc, Southern Leyte, 6604 Philippines
Telefax No.: (053) 382-3121; E-mail: slsubc@yahoo.com
Website: www.slsuonline.edu.ph

Advanced Database Systems Laboratory

LABORATORY GUIDE 2020

REXAL S. TOLEDO

SOUTHERN LEYTE
STATE UNIVERSITY-BONTOC

Southern Leyte State University

Vision

A high quality corporate University of Science, Technology and Innovation.

Mission

SLSU will:

a) *Develop Science, Technology, and Innovation leaders and professionals;*
b) *Produce high-impact technologies from research and innovations;*
c) *Contribute to sustainable development through responsive community engagement programs;*
d) *Generate revenues to be self-sufficient and financially-viable.*

Quality Policy

We at Southern Leyte State University commit enthusiastically to satisfy our stakeholders' needs and expectations by adhering to good governance, relevance and innovations of our instruction, research and development, extension and other support services and to continually improve the effectiveness of our Quality Management System in compliance to ethical standards and applicable statutory, regulatory, industry and stakeholders' requirements.

The management commits to establish, maintain and monitor our quality management system and ensure that adequate resources are available.

COURSE OVERVIEW

Course No.	IT301/IT301L
Course Code	
Descriptive Title	Advanced Database Systems
Credit Units	2 units (Lecture) / 1 unit (Laboratory)
School Year/Term	2020-2021 / 1st semester
Mode of Delivery	
Name of Instructor	Rexal S. Toledo
Course Description	This course covers modern database and information system as well as research issues in the field. It will cover selected topics on NoSQL, object-oriented, active, deductive, spatial, temporal and multimedia databases. The course includes advanced issues of object-oriented database, XML, advanced client server architecture, Information Retrieval and Web Search and distributed database techniques.
Course Outcomes	1. Analyze a complex computing problem and to apply principles of computing and other relevant disciplines to identify solutions. 2. Design, implement, and evaluate a computing-based solution to meet a given set of computing requirements in the context of the program's discipline. 3. Apply software development fundamentals to produce computing-based solutions. 4. Communicate effectively in a variety of professional contexts.

TABLE OF CONTENTS

MODULE 1

Installation

GUIDE

1 Laravel Documentation

2 Download and Install Laragon Tutorial

3 Laravel Introduction and first installation

4 Composer Installation

POST-TEST: Activities

LESSON 1

Laravel Documentation

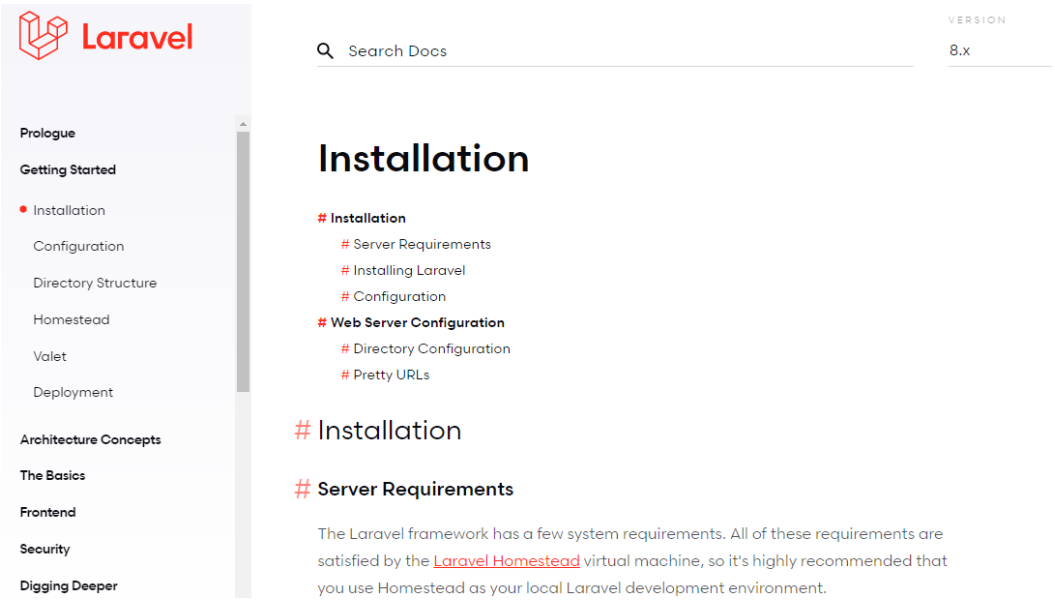
LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Navigate Laravel documentation site.

For more complete documentation on building a local Laravel development environment, check out the full Homestead and installation documentation.

https://laravel.com/docs/8.x/installation



LESSON 2

Download and Install Laragon

LEARNING OUTCOMES

After studying this lesson, you should be able to:

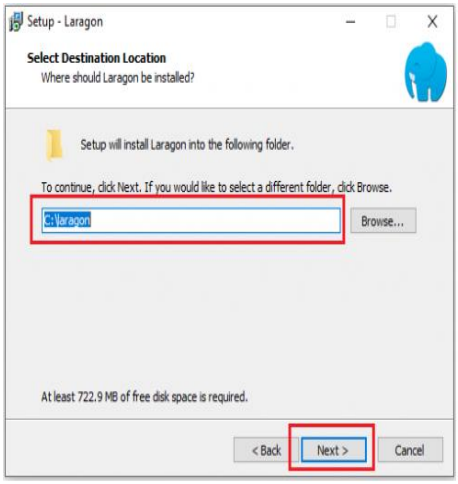
1. Setup Laragon

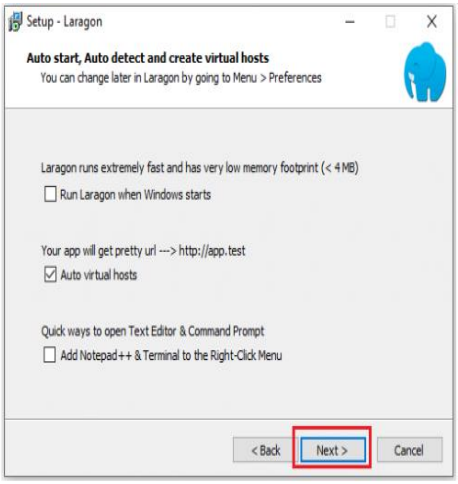
Install Laragon

Go to Laragon's official download site (**http://www.laragon.org/download/**) and download the latest version of the appropriate installer for your version of Windows. For the 64-bit Windows users, download either the Full or Lite edition. For the purpose of this tutorial, we shall be using the full edition.

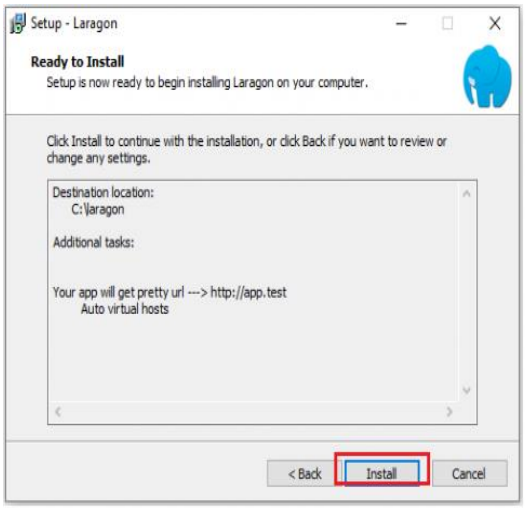
After the download is complete, double-click on the Laragon installer to initiate the installation process. You may see a notice from **User Account Control** asking if you want to allow the app to make changes to your device; click the 'Yes' button to continue the installation process.

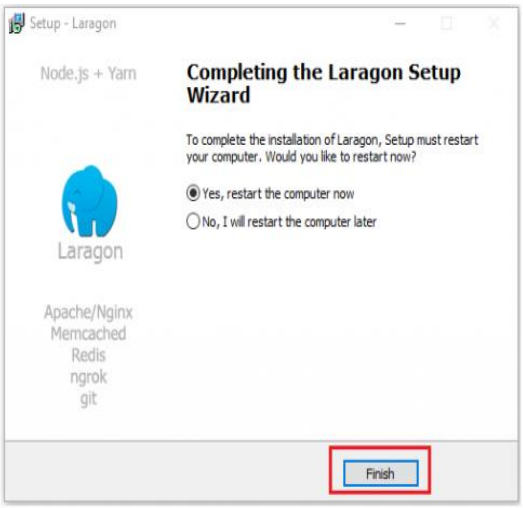
1. On the **"Select Setup Language**" dialog, select your language of choice, in this case, English then click on the OK button.
2. On the **"Select Destination Location"** page, go with the default destination option which is C:\laragon, and then click on the Next button. On the **"Setup Options"** page, it is my personal preference to uncheck these two checkboxes: 'Run Laragon when Windows starts' and 'Add sublime & Terminal to the Right-click Menu'. Now, click Next.





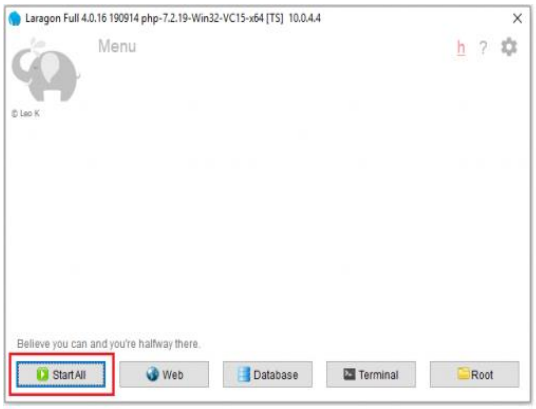
3. On the "**Ready to Install**" page, click Install. When the installation is done, click on Finish so as to restart your machine.

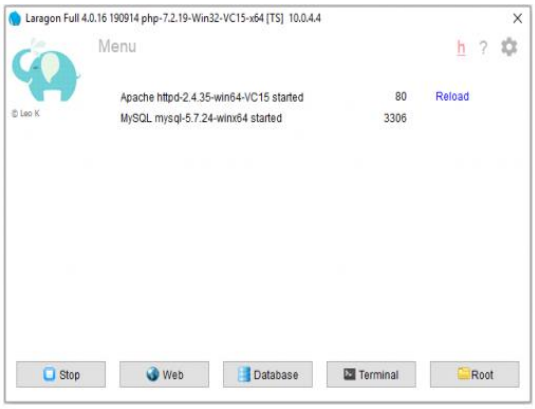




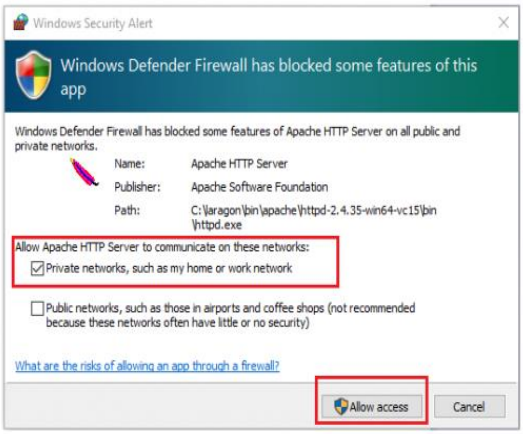
Start Laragon

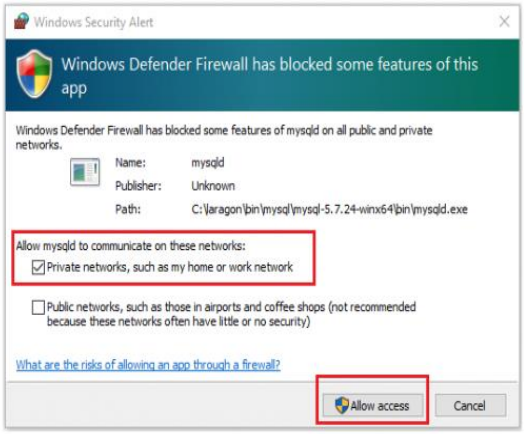
After your PC is done rebooting, launch the Laragon app from the desktop or the Start menu. When Laragon opens, you should see an interface that looks like the image on the left below. Click on the **Start All** button to start Apache and MySQL.





In the event you get a firewall notice like the ones depicted in the image set below, you can choose to deny or allow access. But since I trust my private network, I normally click on the checkbox for private networks and then the 'Allow access' button for both **Apache HTTP Server** and **mysqld** so they can communicate on my private network.





Let us proceed with installing selected applications into the Laragon directory and configuring as part of our Laragon development environment.

LESSON 3

Laravel Introduction and First Installation

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Learn Laravel
2. Know the prerequisites for installing Laravel
3. Explain all the necessary concepts in easy language for you to learn Laravel easily and understand it better.

Introduction

Laravel is an elegant, expressive, and flexible PHP framework with an extreme focus on clean code and speed which describes itself as "The PHP framework for web artisans". It is a free and open source PHP framework created by Taylor Otwell, based on Model View Controller (MVC) architecture.

Creating a web application from scratch can be daunting especially if you are a beginner. A simple web application also contains various small and big pieces and creating those pieces every time you are creating a web app can be boring and repetitive and there is no point in reinventing wheels. That's when Laravel comes to your rescue.

Laravel framework provides various PHP libraries and helper functions and can help you to focus on more important pieces while providing common functions and logic to speed up the development time and ease up the development.

Initially, there is a bit of a learning curve especially if you are a beginner and have no experience with any kind of web framework. But believe me, once you flow with it, you will not only love, and you will become addicted to it. Laravel aims at creativity in development. It uses the word 'Web Artisan' to point out the creative hidden inside the developer's heart. Result -> Efficient Application with fewer lines and well-designed code.

To make it easier for you to learn, I wrote this Laravel tutorial with a beginner

audience in mind. Thus you will find it easy to follow this tutorial for Laravel to learn.

Installation and Configuration

Laravel offers various ways to install in windows or mac. The best and easiest way to install Laravel is through Composer. Composer is a dependency manager for PHP which you can install on your web server.

Prerequisites for Installing Laravel

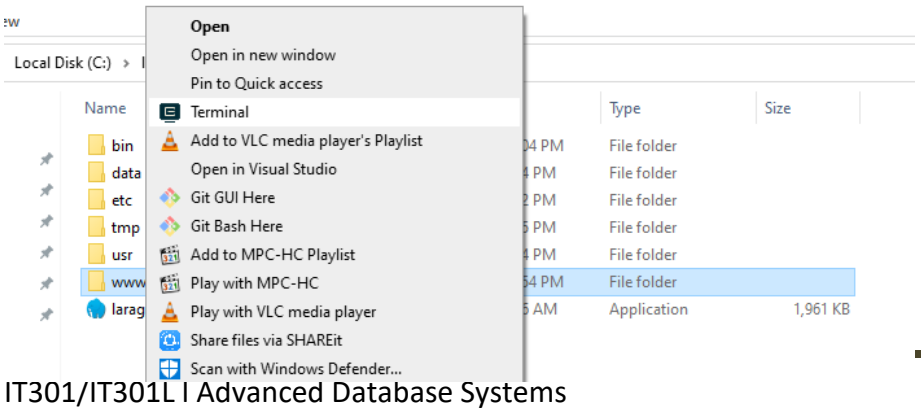
Before installing Laravel on your local platform (Localhost) you need to install the following programs:

- Web Server – Apache or Nginx
- PHP >= 7.2.5 (*This php version is a prerequisite for Laravel 7*)
- MySQL
- Composer
- An IDE will be really helpful for Laravel development. I recommend Sublime 3 or Atom. Both are free to use but Sublime also has a PRO version.
- Some PHP extensions which might be pre-installed:
  - BCMath PHP Extension
  - Ctype PHP Extension
  - Fileinfo PHP extension
  - JSON PHP Extension
  - Mbstring PHP Extension
  - OpenSSL PHP Extension
  - PDO PHP Extension
  - Tokenizer PHP Extension
  - XML PHP Extension

SIMPLE STEPS TO START YOUR FIRST LARAVEL PROJECT

Step 1: Install composer (see at Module 1: Lesson 4), Sublime (HTML/text editor), and Laragon (preferred).

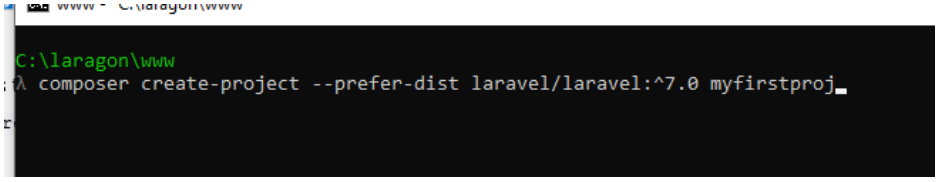
Step 2: Go to c:/laragon/www. -> right-click **www** folder -> then click terminal

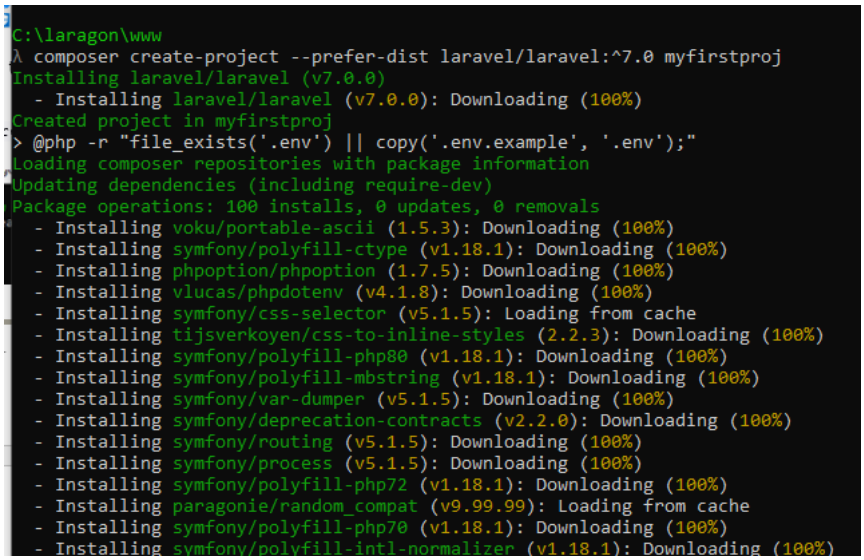


Step 3: To install Laravel version 7.0 -> Type this command,

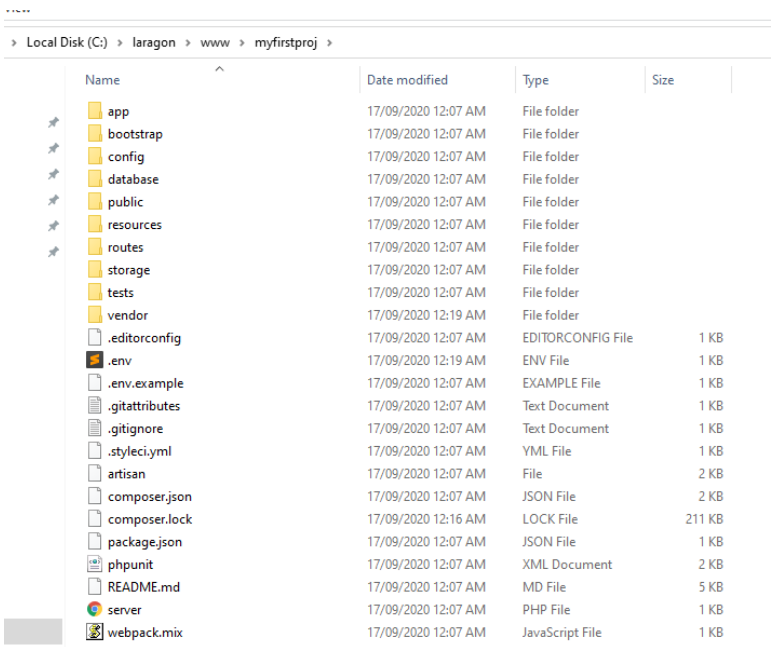
```
composer create-project --prefer-dist laravel/laravel:^7.0 myfirstprog
```

, then press enter.

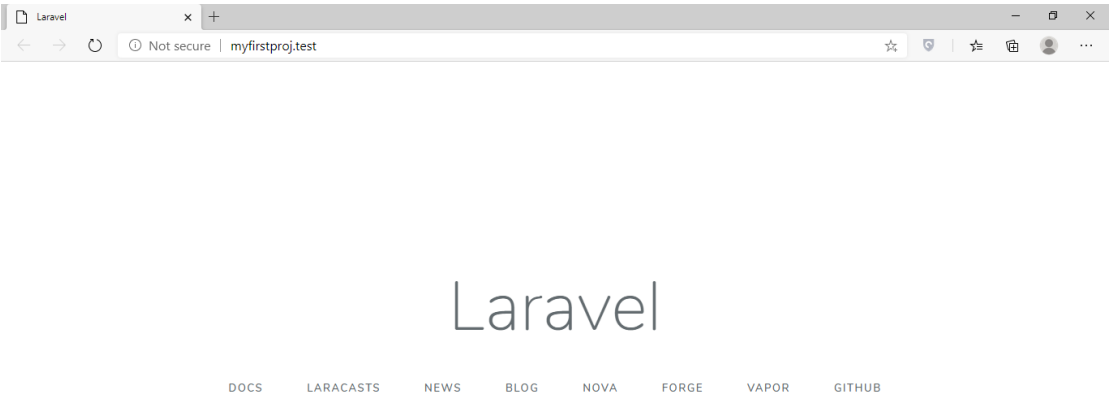




Step 4: After installing the Laravel, you'll see these folders and files.



Step 5: Browse: http://myfirstproj.test/.



Exploring Directory Structure

Laravel applications follow the **Model-View-Controller** architecture design pattern.

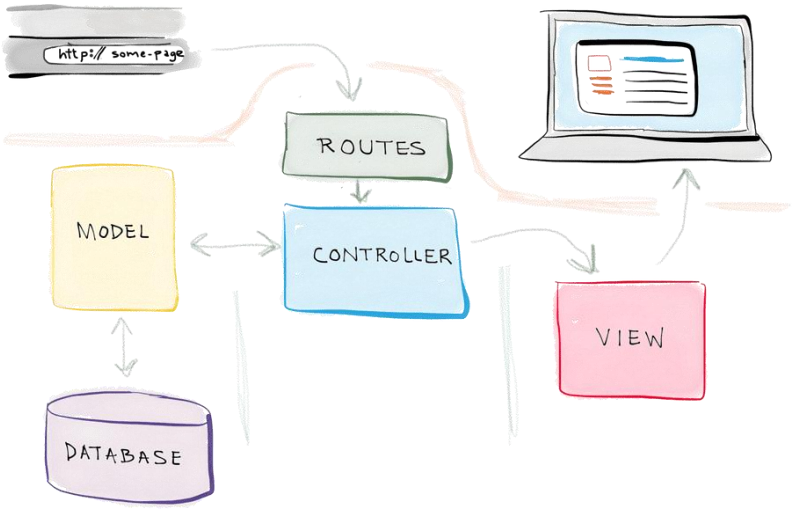


Image Source: SelfTaughtCoders.com

- **Models** represent the entities in the database and help you to query the database and return the data.
- **Views** are the pages that will be displayed when accessed the app. View Component is used for the User Interface of the application.
- **Controllers** handle user requests, get required data from the models, and pass them to the Views. Controllers act as an intermediary between Model and View Components to process the business logic and incoming request.

When you installed the composer, and created your first Laravel web app, you might have noticed the app folder with different files and folders. I know if you are a beginner, you may have a lot of questions about what are these folders for, etc.

Let's understand some

```
app
    Console
    Exceptions
    Http
    Providers
bootstrap
config
database
    migrations
    seeds
public
resources
    js
    lang
    saas
    views
routes
storage
    app
    framework
    logs
tests
vendor
```

- **App:** This directory is the meat of the application and contains the core code.
  - **Console:** This directory contains all the custom Artisan commands created using **make:command**
  - **Exceptions:** This directory contains the application's exception handler and is a good place to add custom exception classes to handle different exceptions thrown by your application
  - **Http:** This directory contains all your controllers, middleware, and requests
  - **Providers:** This directory contains all your service providers for the application. You can know more about service providers here
- **Bootstrap:** This directory contains framework bootstrap as well as configuration files. It also contains the **Cache** directory which contains framework generated cache files
- **Config:** This directory contains all your application's configuration files.
- **Database:** This directory contains all database migrations and seeds. You can also store the SQLite database file here
- **Public:** This directory contains assets like images, js files and CSS.
- **Resources:** This directory contains all view files and LESS or SASS files. It also contains a **lang** directory to store language files.
- **Routes:** This directory contains all routes definitions for the application. **php** is the file that receives all the requests to your application and here you can redirect the requests to their respective controller methods.
- **Storage:** This directory contains blade templates, session files, cache files, and others.
- **Tests:** This directory contains all the test files
- **Vendor:** This directory contains all composer dependencies

LESSON 4

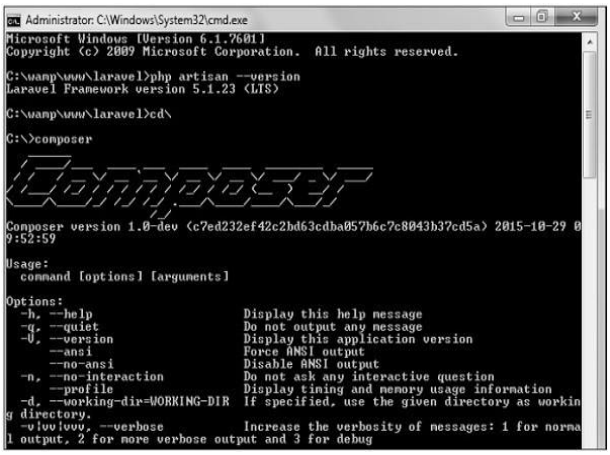
Composer Installation

LEARNING OUTCOMES

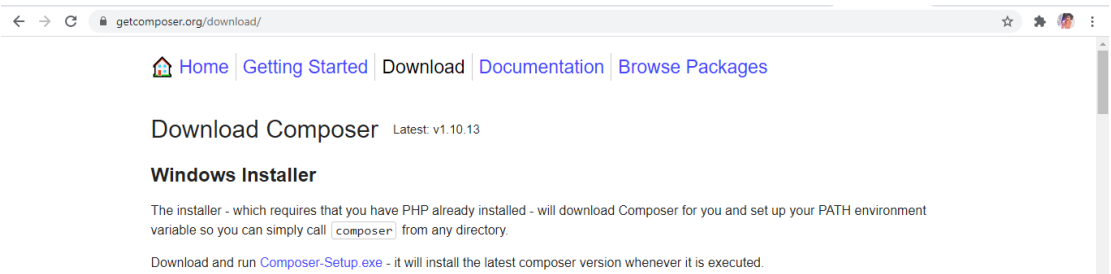
After studying this lesson, you should be able to:

1. Set-up installation of Composer.

Steps for windows users:



1. Download Composer from https://getcomposer.org/download/ and install it. After installation, you should check whether it's installed globally or not. Open Command Prompt and enter command "composer" just like shown below.



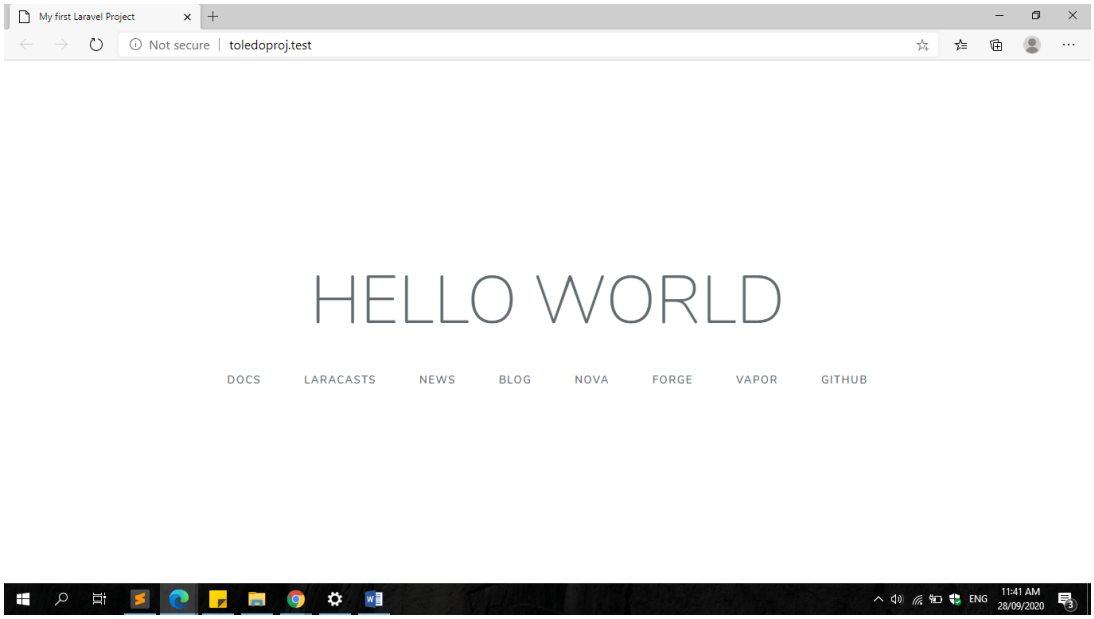
POST-TEST

LABORATORY MODULE 1: ACTIVITY

Install and set-up your first Laravel project. Follow the naming convention: **"<surname>proj"**.

Example: toledoproj.

Then change the word Laravel into HELLO WOLRD and change the title to "My first Laravel Project".



Tip: To modify, locate the file in Resources -> Views Folder.

MODULE 2

Using MySQL, PHP artisan and .ENV

LESSON

5 Create a new Database with MySQL

6 Update Database Connection on .env file

7 PHP artisan commands

Activities

LESSON 5

Create a new Database with MySQL

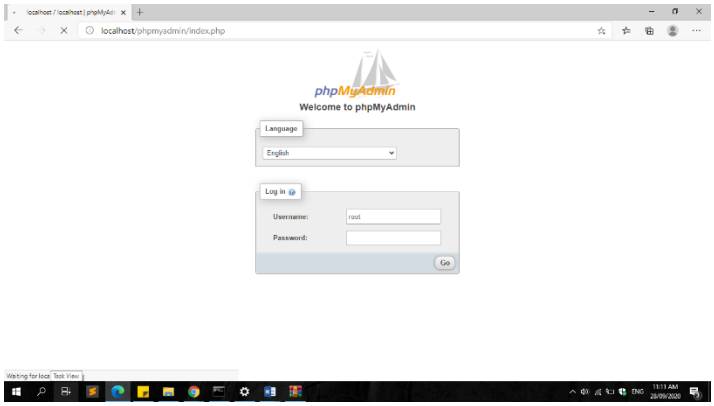
LEARNING OUTCOMES

After studying this lesson, you should be able to:

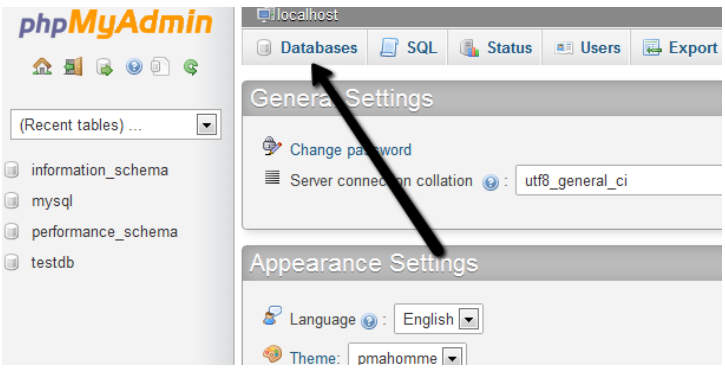
1. Learn how Create MySQL database using PHPMyAdmin
2. Explore phpMyAdmin

Creating Database

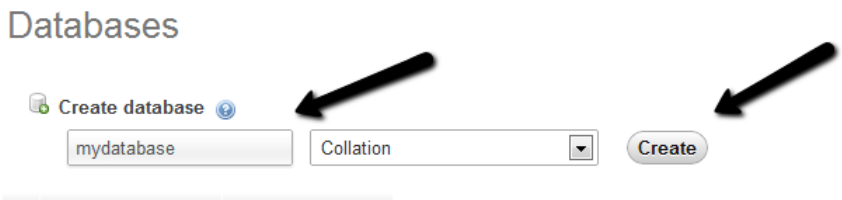
1. Browse to your phpMyAdmin URL using your Internet Web Browser, and login.



2. From the main menu choose Databases.



3. In the create database field type in a name for your database. Leave the collation drop down box if you wish to use the default MySQL schema collation. Click **Create**.



LESSON 6

Database Connection

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Configure database connection

Laravel provides **config/database.php** to config database for production server but Laravel also works with **.env** file where you can configure your database for your development server. As we are developing this app in localhost we will work with the **.env** file, so

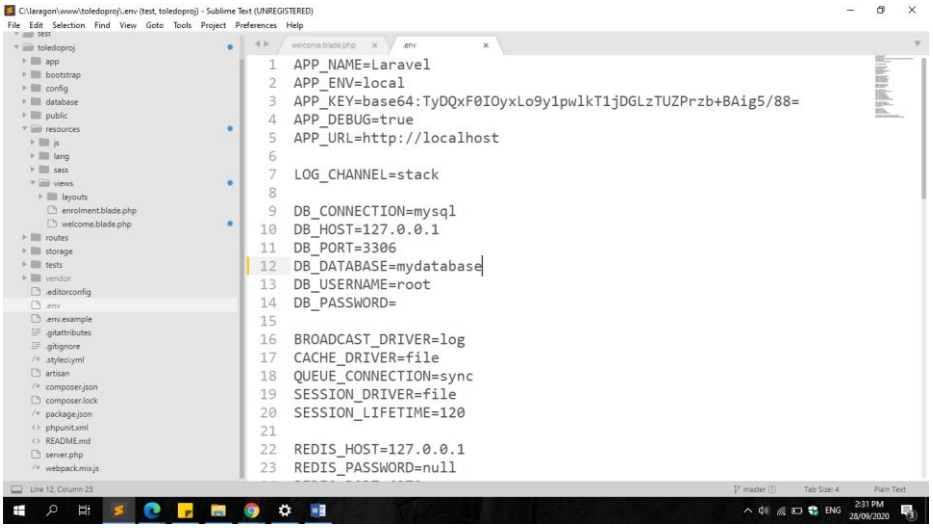
1. Clone **.env.example** file as **.env**

```
cp .env.example .env
```

2. In the file you will find code like below:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret
```

3. Then update the database name. My database name is: **mydatabase**



LESSON 7

PHP artisan commands

LEARNING OUTCOMES

After studying this lesson, you should be able to:

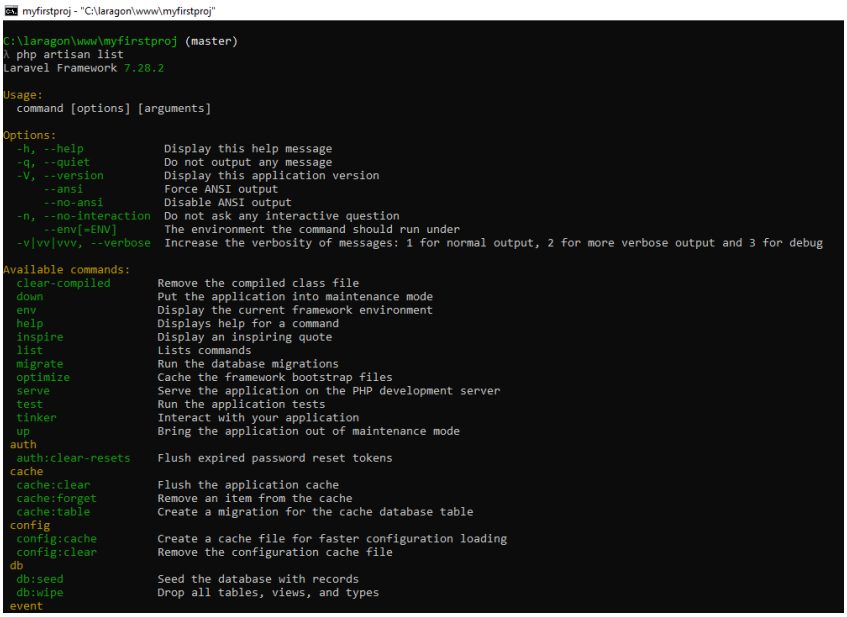
2. Use php artisan commands

PHP ARSTISAN COMMANDS

Artisan is the command-line interface included with Laravel. It provides a number of helpful commands that can assist you while you build your application. To view a list of all available Artisan commands, you may use the list command:

```
php artisan list
```

Every command also includes a "help" screen which displays and describes the command's available arguments and options. To view a help screen, precede the name of the command with `help`:



```
  db:wipe              Drop all tables, views, and types
 event
  event:cache          Discover and cache the application's events and listeners
  event:clear          Clear all cached events and listeners
  event:generate       Generate the missing events and listeners based on registration
  event:list           List the application's events and listeners
 key
  key:generate         Set the application key
 make
  make:cast            Create a new custom Eloquent cast class
  make:channel         Create a new channel class
  make:command         Create a new Artisan command
  make:component       Create a new view component class
  make:controller      Create a new controller class
  make:event           Create a new event class
  make:exception       Create a new custom exception class
  make:factory         Create a new model factory
  make:job             Create a new job class
  make:listener        Create a new event listener class
  make:mail            Create a new email class
  make:middleware      Create a new middleware class
  make:migration       Create a new migration file
  make:model           Create a new Eloquent model class
  make:notification    Create a new notification class
  make:observer        Create a new observer class
  make:policy          Create a new policy class
  make:provider        Create a new service provider class
  make:request         Create a new form request class
  make:resource        Create a new resource
  make:rule            Create a new validation rule
  make:seeder          Create a new seeder class
  make:test            Create a new test class
 migrate
  migrate:fresh        Drop all tables and re-run all migrations
  migrate:install      Create the migration repository
  migrate:refresh      Reset and re-run all migrations
  migrate:reset        Rollback all database migrations
  migrate:rollback     Rollback the last database migration
  migrate:status       Show the status of each migration
 notifications
  notifications:table  Create a migration for the notifications table
 optimize
  optimize:clear       Remove the cached bootstrap files
 package
  package:discover     Rebuild the cached package manifest
 queue
  queue:failed         List all of the failed queue jobs
  queue:failed-table   Create a migration for the failed queue jobs database table
  queue:flush          Flush all of the failed queue jobs
  queue:forget         Delete a failed queue job
  queue:listen         Listen to a given queue
  queue:restart        Restart queue worker daemons after their current job
  queue:retry          Retry a failed queue job
  queue:table          Create a migration for the queue jobs database table
  queue:work           Start processing jobs on the queue as a daemon
 route
  route:cache          Create a route cache file for faster route registration
  route:clear          Remove the route cache file
  route:list           List all registered routes
 schedule
  schedule:run         Run the scheduled commands
 session
  session:table        Create a migration for the session database table
 storage
  storage:link         Create the symbolic links configured for the application
 stub
  stub:publish         Publish all stubs that are available for customization
 vendor
  vendor:publish       Publish any publishable assets from vendor packages
 view
  view:cache           Compile all of the application's Blade templates
  view:clear           Clear all compiled view files

C:\laragon\www\myfirstproj (master)
λ
```

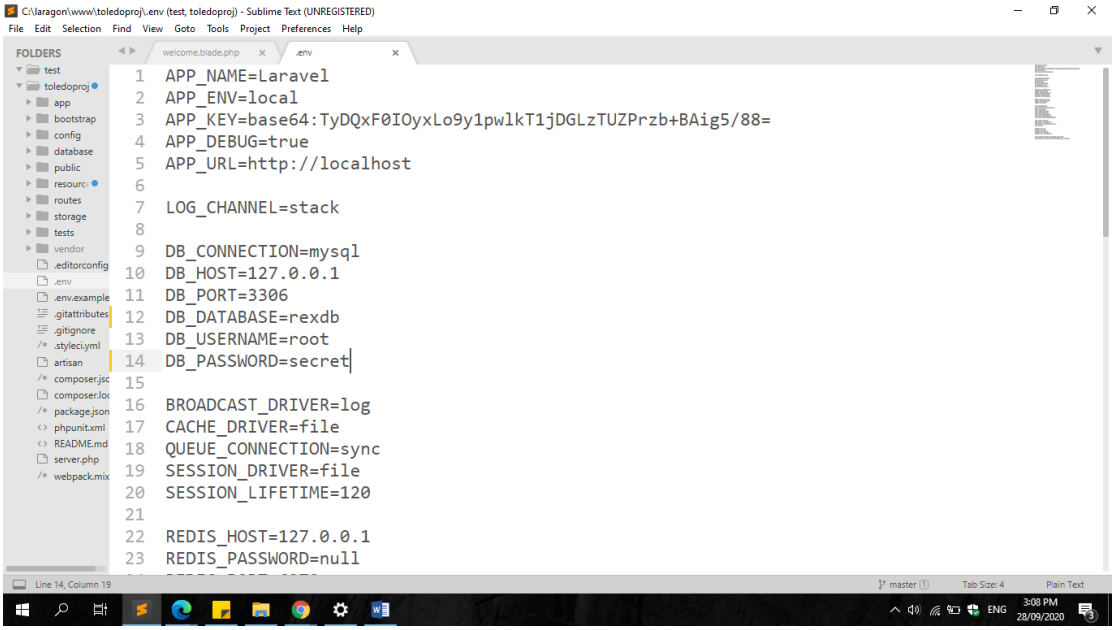
POST-TEST

LABORATORY MODULE 2: ACTIVITIES

1. Create a database name "**activitydb**".

2. After creating database, update your database connection to **.ENV** file. Use your own database.

Example:




```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=rexdb
DB_USERNAME=root
DB_PASSWORD=secret
```

MODULE 3

Using route, controller and Views

LESSON

LESSON 8

HTTP Routing

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Use routes in Laravel
2. Understand the purpose of routing in Laravel.

Routing

The Default Route Files

For most applications, you will begin by defining routes in your routes/web.php file. The routes defined in routes/web.php may be accessed by entering the defined route's URL in your browser. For example, you may access the following route by navigating to http://your-app.test/user in your browser:

```
Route::get('/user', 'UserController@index');
```

You may also specify route names for controller actions:

```
Route::get('user/profile', 'UserProfileController@show')-
>name('profile');
```

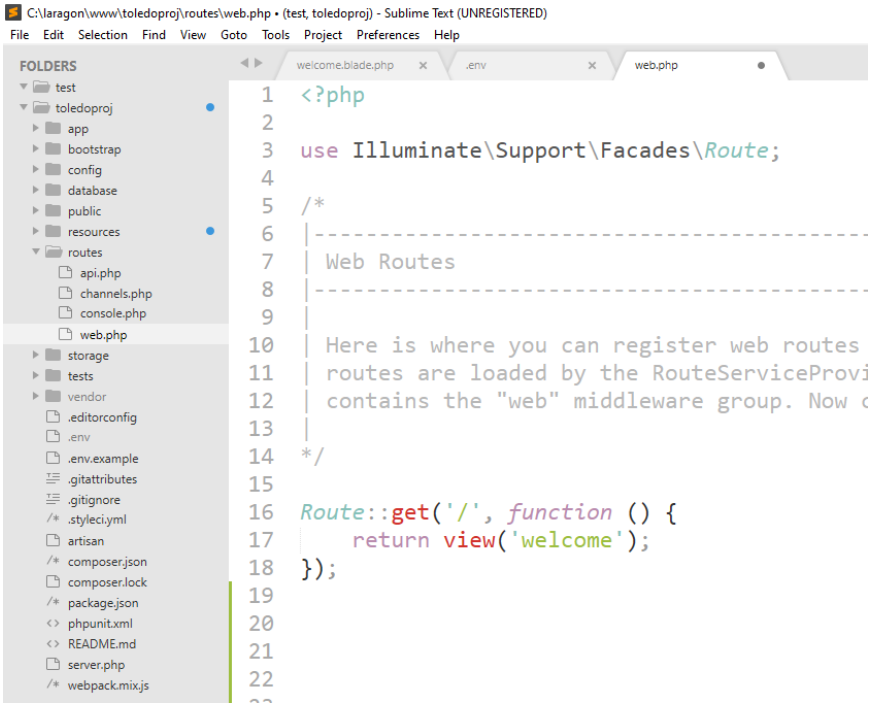
View Routes

If your route only needs to return a view, you may use the Route::view method. Like the redirect method, this method provides a simple shortcut so that you do not have to define a full route or controller. The view method accepts a URI as its first argument and a view name as its second argument. In addition, you may provide an array of data to pass to the view as an optional third argument:

```
Route::view('/welcome', 'welcome');

Route::view('/welcome', 'welcome', ['name' => 'Taylor']);
```

For more details: https://laravel.com/docs/7.x/routing

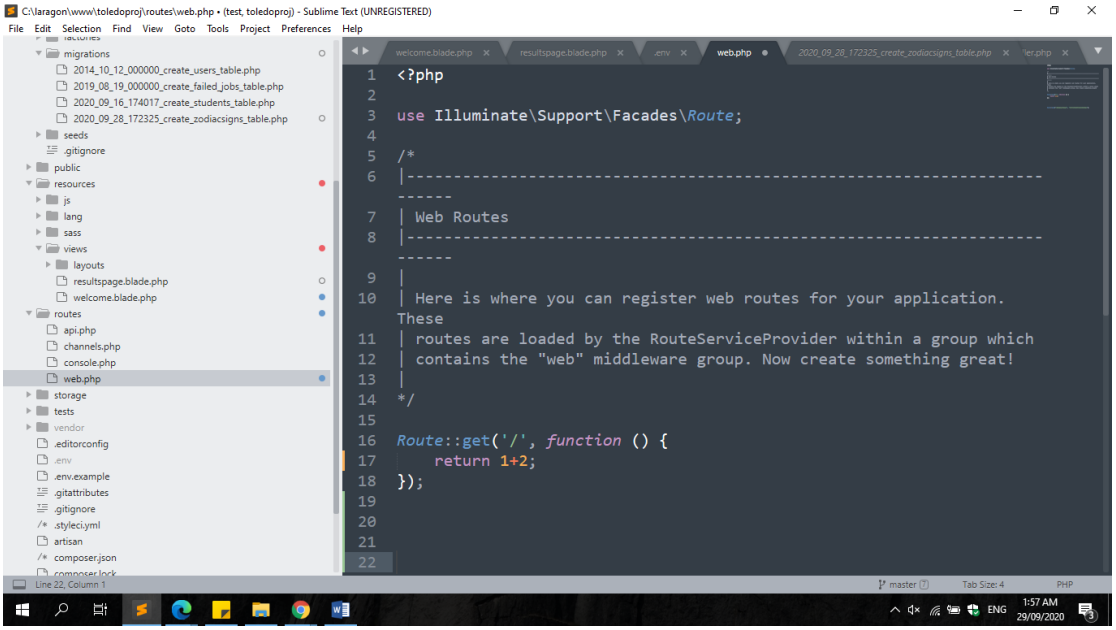
Routing means accepting the request and redirect it to the appropriate function. Login and register are added by default by Laravel.



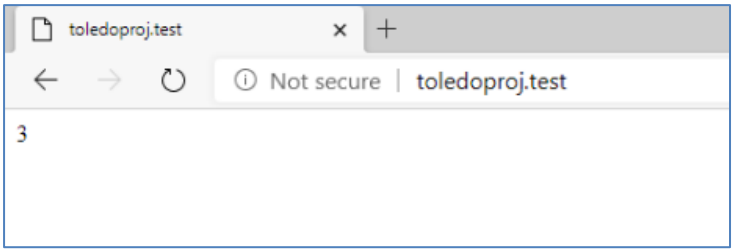
Laravel provides various route files inside `**/routes**` folder for various use cases. For example, routing configuration for API will go in `**/routes/api.php**` file while the routing configuration for our regular web application will go in `**/routes/web.php**`.

EXERCISE:

Our goal is to print the result of 1 + 2. Now, let's edit **web.php**. Below is the edited version of the file.



Output:



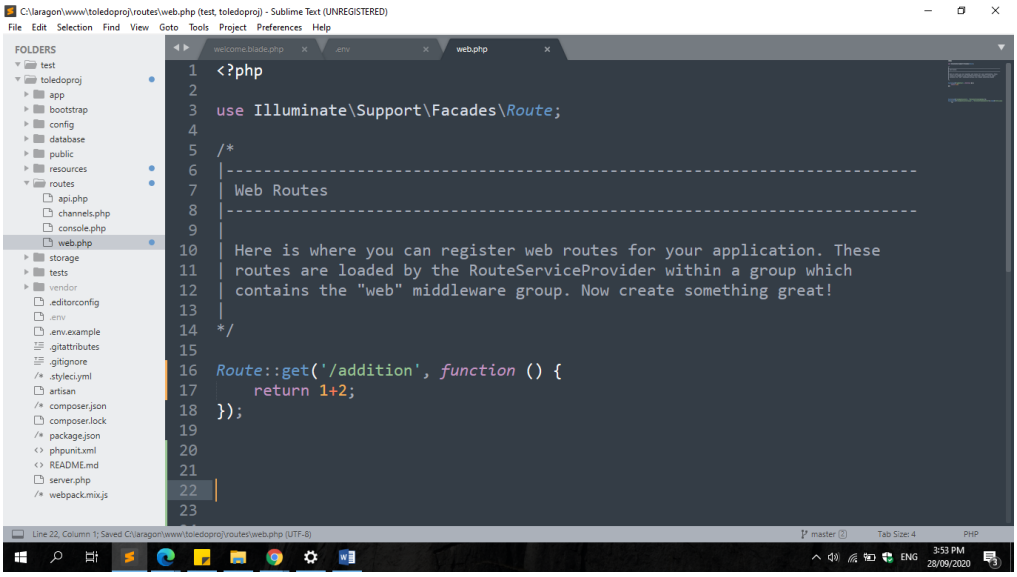
EXERCISE:

Our next goal is to print the result of 1 + 2. But, this time we will add **/addition.**

Example:

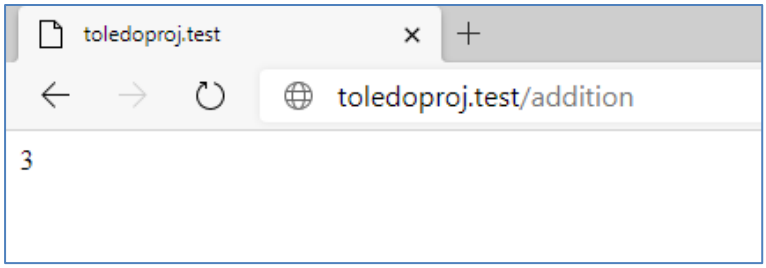
http://toledoproj.test/addition

Now, let's edit web.php. Below is the edited version of the file.



I just added "addition" after the slash. So, now the URL to access the result is http://toledoproj.test/addition.

Output:



LESSON 9

Controllers

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Use Controllers

Controllers

Instead of defining all of your request handling logic as Closures in route files, you may wish to organize this behavior using Controller classes. Controllers can group related request handling logic into a single class. Controllers are stored in the app/Http/Controllers directory.

Defining Controllers

Below is an example of a basic controller class. Note that the controller extends the base controller class included with Laravel. The base class provides a few convenience methods such as the middleware method, which may be used to attach middleware to controller actions:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\User;

class UserController extends Controller
{
    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return View
     */
    public function show($id)
    {
        return view('user.profile', ['user' =>
        User::findOrFail($id)]);
    }
}
```

You can define a route to this controller action like so:

```
Route::get ('user/{id}', 'UserController@show');
```

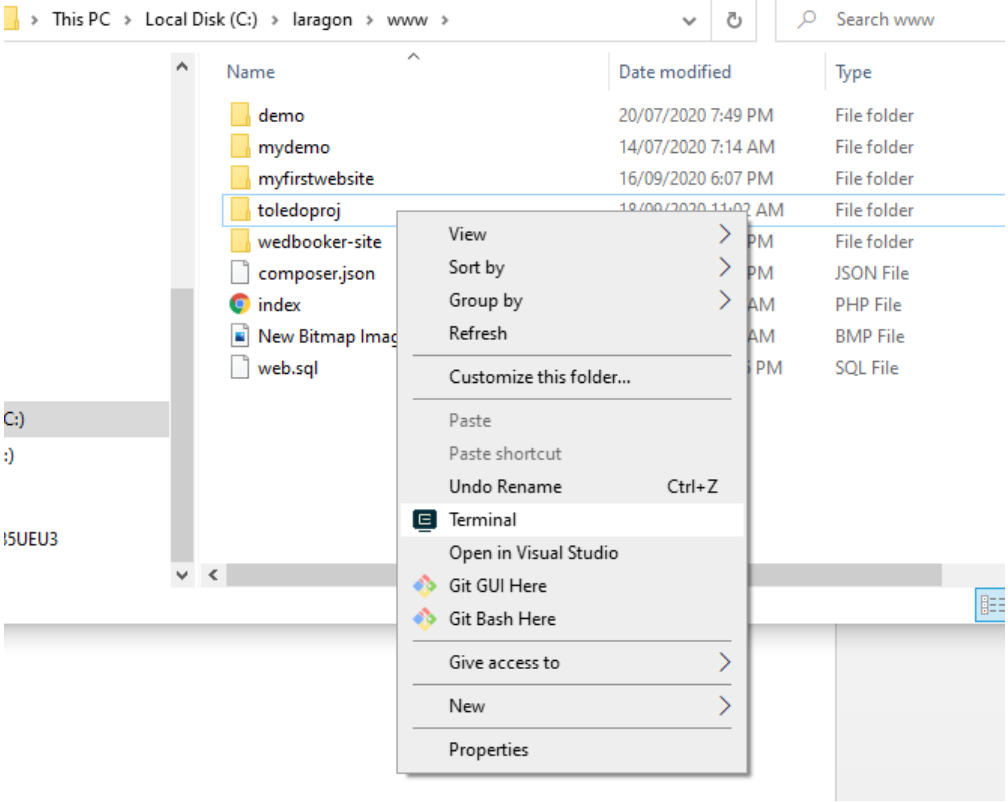
Now, when a request matches the specified route URI, the `show` method on the `UserController` class will be executed. The route parameters will also be passed to the method.

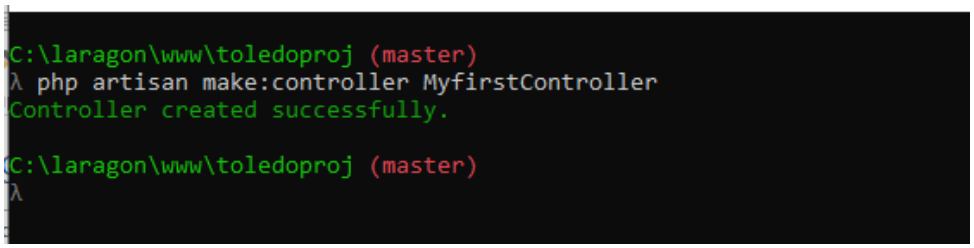
For more details: https://laravel.com/docs/7.x/controllers

EXERCISE:

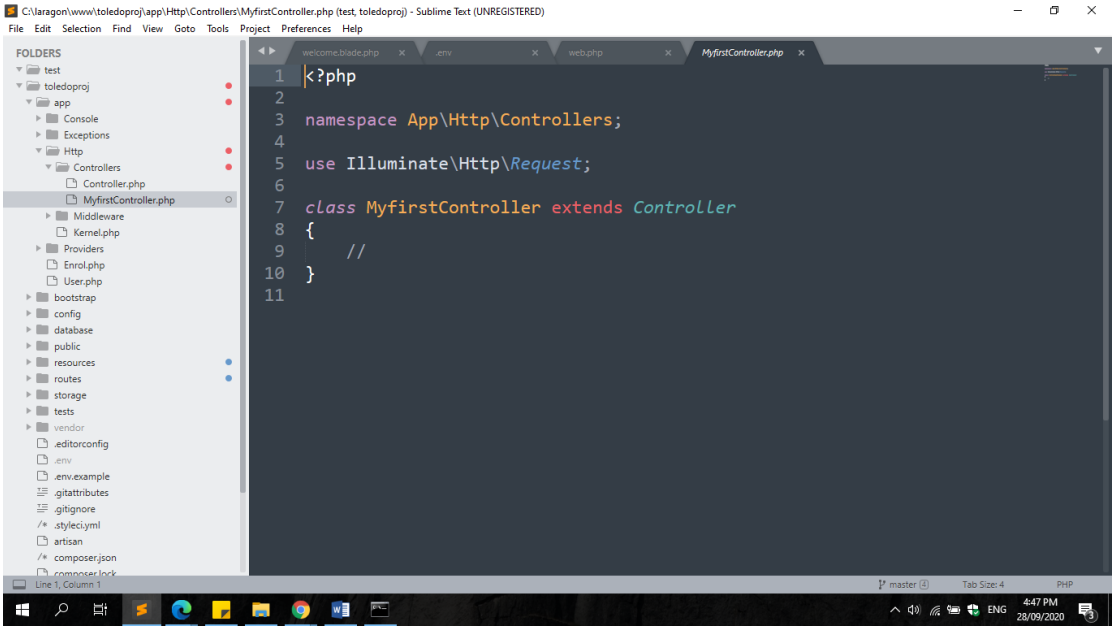
Step 1: **Make a Controller**. In the terminal, type:

```
php artisan make:controller MyfirstController
```





To find your created controller file. Go to **app/Http/Controller/.**



Step 2: Open **routes/web.php** file, then define the route of the controller action. Just add the following codes:

http://toledoproj.test/displaythesum

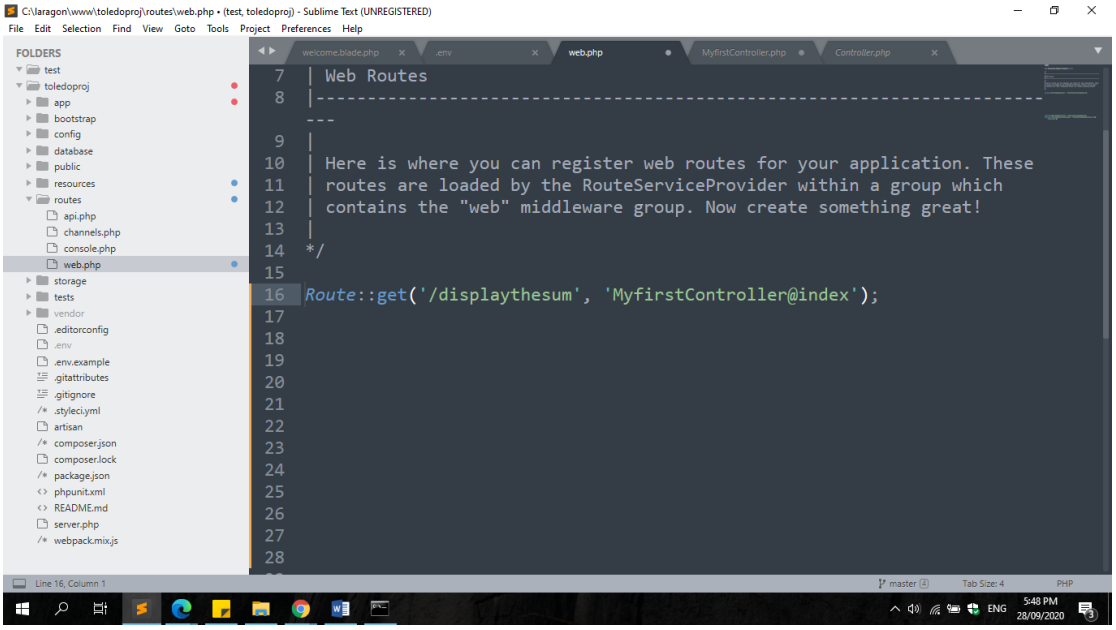
Controller Class

Function

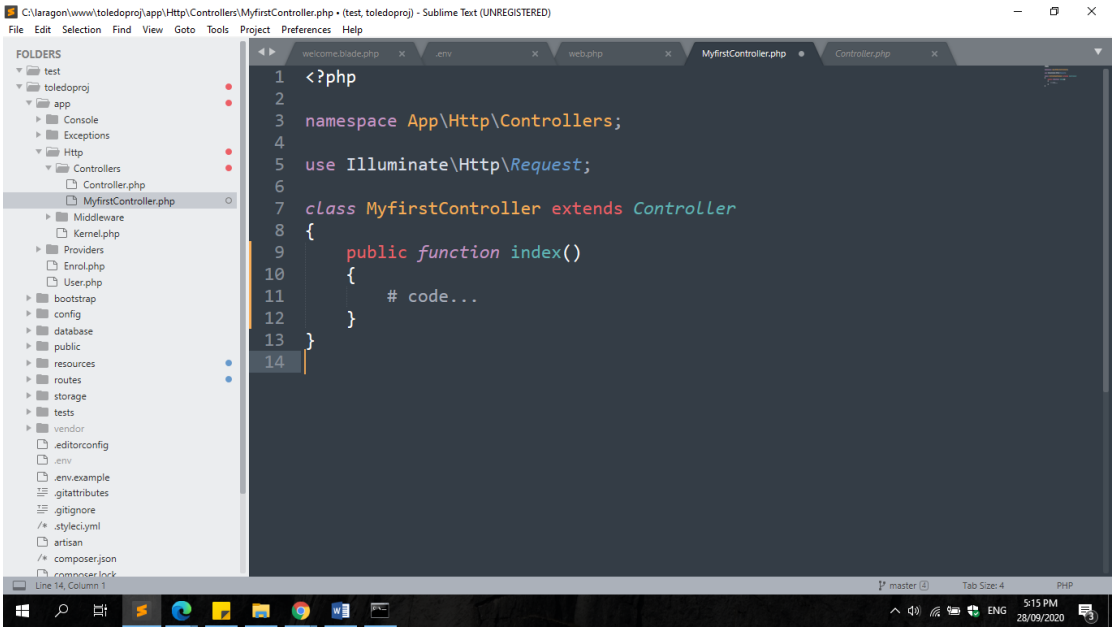
```
Route::get('/displaythesum', 'MyfirstController@index');
```

Note: "index" can be change, but it depends upon your needs. If you want to save something use "store". Example:

```
Route::get('/displaythesum', 'MyfirstController@store);
```



Step 3: Go back to **MyfirstController** then add function "index".



Step 4: You've now successfully created your first controller. But we need to display the sum, to continue please proceed to **Lesson 10**: About Views.

Conclusion: Using the route I created, I have now my desire URI to display the sum of two numbers.

LESSON 10

Using Views

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Use views and understand its purpose.
2. Redirect from route to controller then views.

Creating Views

Views contain the HTML served by your application and separate your controller / application logic from your presentation logic. Views are stored in the `resources/views` directory. A simple view might look something like this:

```
<!-- View stored in resources/views/greeting.blade.php -->
<html>
    <body>
        <h1>Hello, {{ $name }} </h1>
    </body>
</html>
```

Since this view is stored at `resources/views/greeting.blade.php`, we may return it using the global `view` helper like so:

```
Route::get('/', function () {
    return view('greeting', ['name' => 'James']);
});
```

Views may also be nested within subdirectories of the `resources/views` directory. "Dot" notation may be used to reference nested views. For example, if your view is stored at `resources/views/admin/profile.blade.php`, you may reference it like so:

```
return view('admin.profile', $data);
```

Passing Data to Views

As you saw in the previous examples, you may pass an array of data to views:

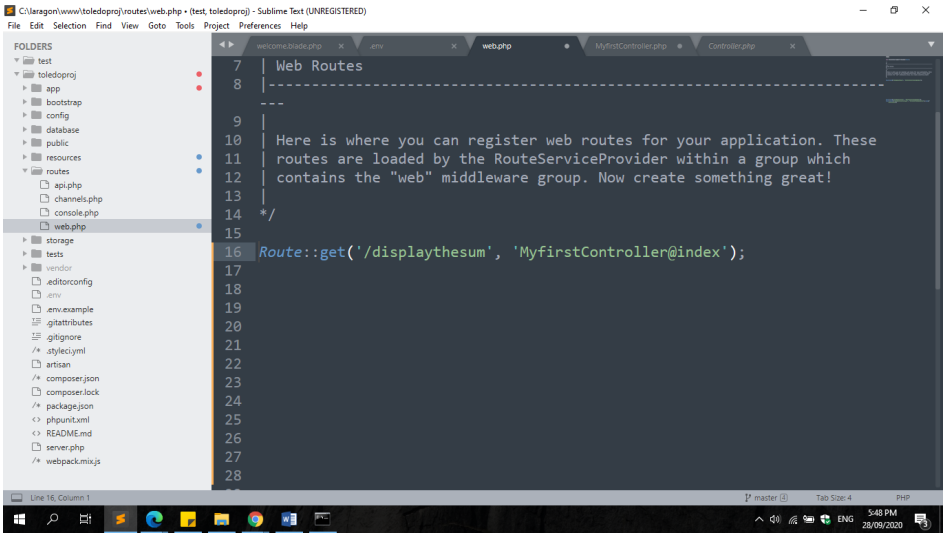
```
return view('greetings', ['name' => 'Victoria']);
```

When passing information in this manner, the data should be an array with key / value pairs. Inside your view, you can then access each value using its corresponding key, such as `<?php echo $key; ?>`.

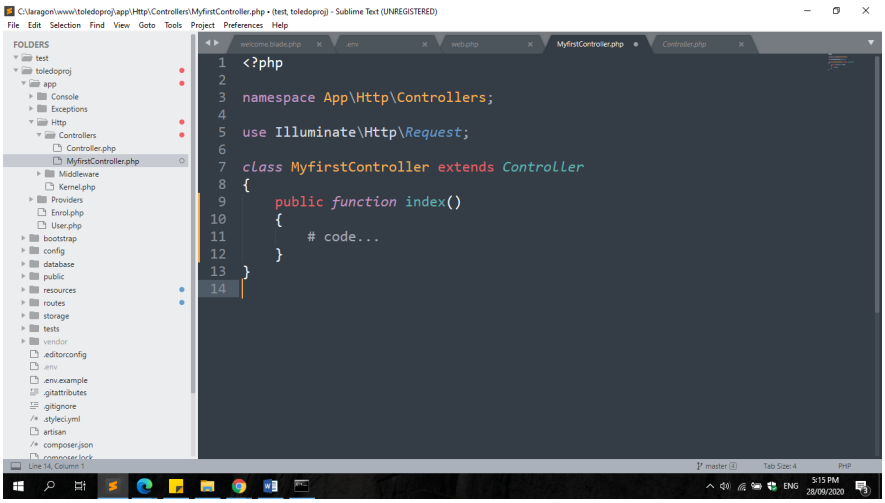
For more details: https://laravel.com/docs/7.x/views

EXERCISE: (Lesson 9 **Continuation**)

Route:

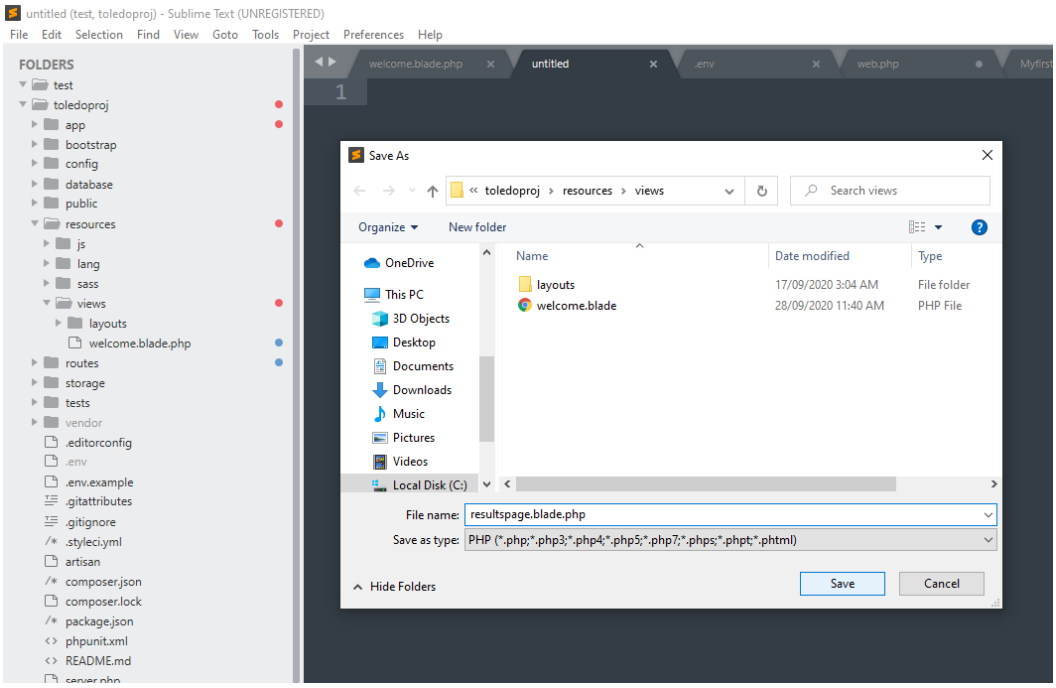


Controller:

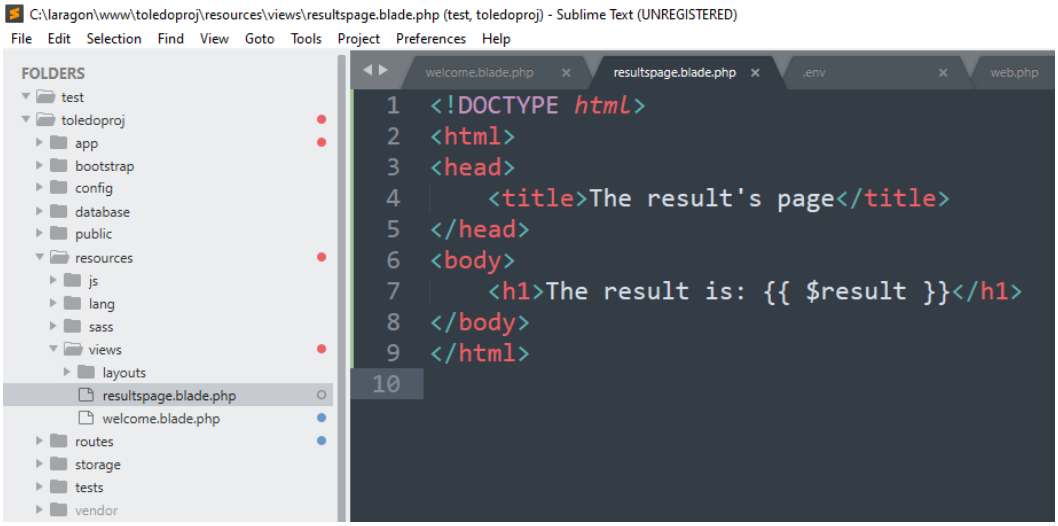


Step 1: We need to create a Blade file first to redirect from controller to views (BLADE).

Go to **resources/views.** Then create a blade filename "resultpage.blade.php".



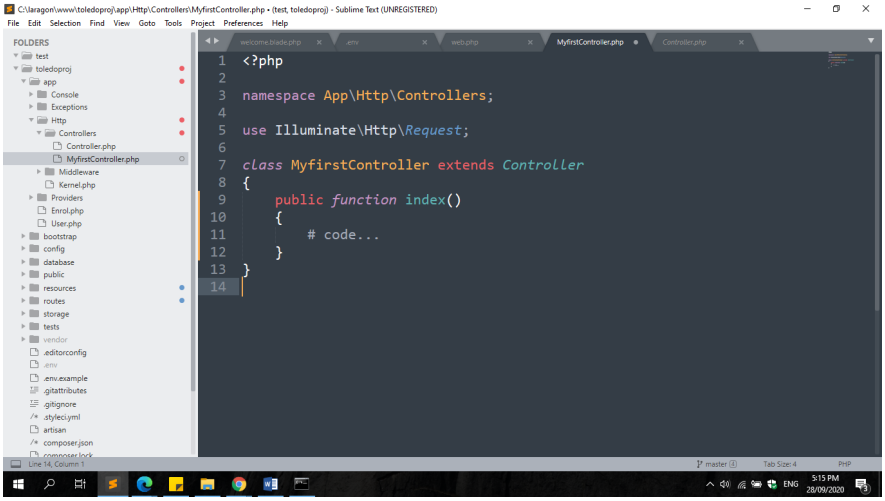
Step 2: Now just add HTML tags.



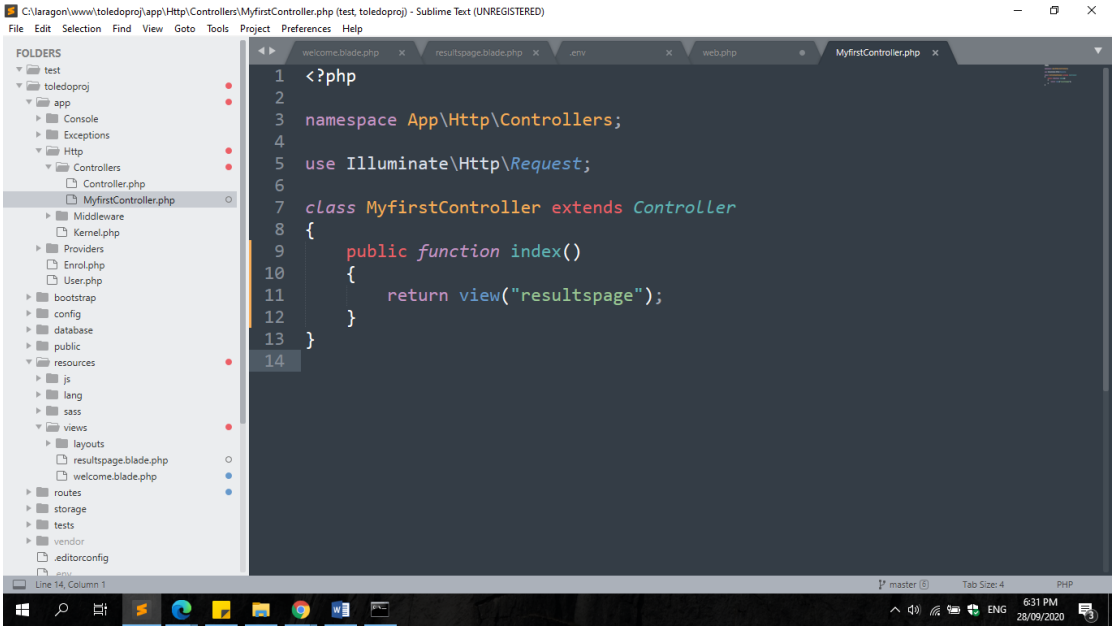
Note: Using `{{$result}}` is a method to display the value of the `result` variable. By default, Blade `{{ }}` statements are automatically sent through PHP's `htmlspecialchars` function to prevent XSS attacks. If you do not want your data to be escaped, you may use the following syntax:

```
{!!$result!!}.
```

Step 3: Go back to **MyfirstController.php**.



Then, use **return** view() method to tell the index function to redirect from **controller/MyfirsController** to **views/resultspage.blade.php.**

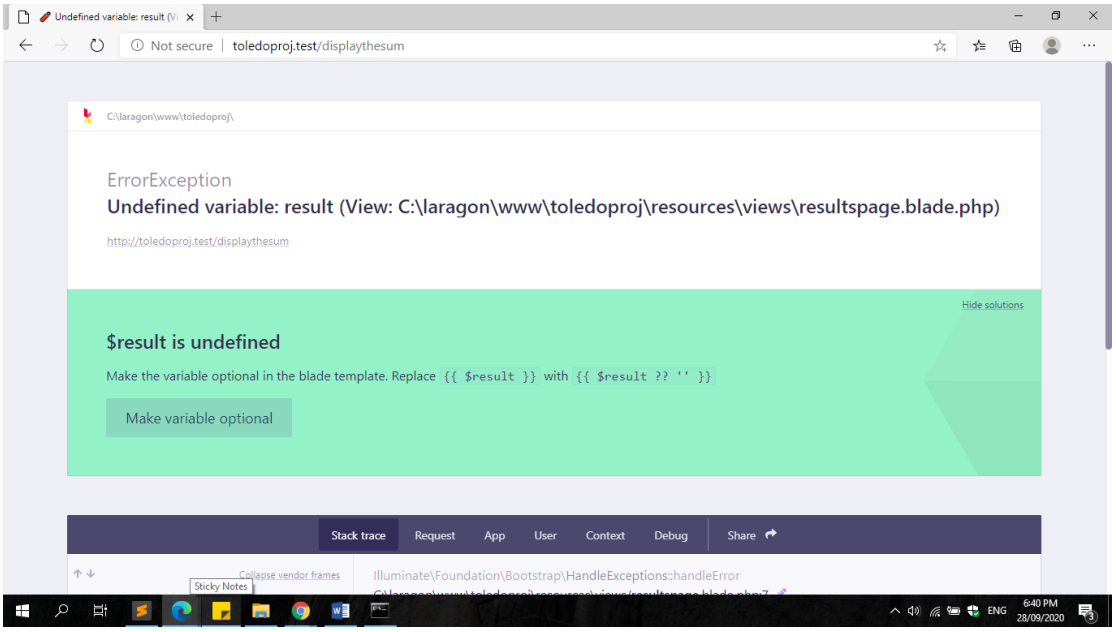


Note: No need to add blade.php. For example, **return** view ("results.blade.php");

Step 4: Using any browser, navigate to URI: For example http://toledoproj.test/displaythesum.

But if you notice, there's an error during execution. But why?

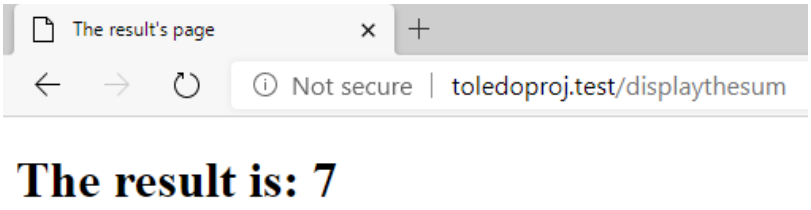
ErrorException **Undefined variable: result**



Remember the `$result` variable that we created earlier? To prevent this kind of error. We need to define the variable first before we display it by passing data to Views. Example:

When passing information in this manner, the data should be an array with key / value pairs. Inside your view, you can then access each value using its corresponding key, such as `<?php echo $result; ?>` or in blade `{{ $result }}`.

Step 5: Go back to any browse then browse the URI:



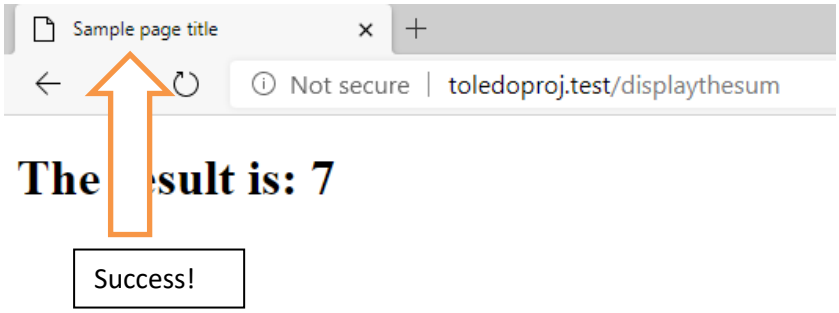
Note: You can also pass the value of the title. For example:

C:\laragon\www\toledoproj\app\Http\Controllers\MyfirstController.php (test, toledoproj) - Sublime Text (UNREGISTERED)

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class MyfirstController extends Controller
{
    public function index()
    {
        $sum = 5+2;
        return view('resultspage', [
            'result' => $sum,
            'title' => "Sample page title"
        ]);
    }
}
```

C:\laragon\www\toledoproj\resources\views\resultspage.blade.php (test, toledoproj) - Sublime Text (UNREGISTERED)

```php
<!DOCTYPE html>
<html>
<head>
    <title>{{ $title }}</title>
</head>
<body>
    <h1>The result is: {{ $result }}</h1>
</body>
</html>
```



POST-TEST

LABORATORY MODULE 3:

1. Create a route **without** controller and blade to the following URI:

URI/URL	RETURN (Display)
http://localhost/addition	The sum of 5 and 8 is 13.
http://localhost/multiply	The difference of 10 and 1 is 9.
http://localhost/product	The product of 3 and 2 is 6.
http://localhost/quotient	The quotient of 10 and 2 is 5.
http://localhost/numbers	[1, 2, 3, 4, 5, 6, 7, 8, 9]

2. Create a route **without** controller and blade to the following URI and group using **Route Prefixes**:

URI/URL
http://localhost/math/addition
http://localhost/math/multiply
http://localhost/math/product
http://localhost/math/quotient

3. Create 3 pages using Route, Controller, and Views.

URI/URL	Description
http://localhost/	Display the homepage of the website.
http://localhost/product/list	List of products added from the database. But this time, display any products as a sample.
http://localhost/product/create	Display the form to store products

MODULE 4

Database: Migrations, Schema and Model

LESSON

LESSON 11

Migrations

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Make migrations using php artisan.

Migrations are like version control for your database, allowing your team to modify and share the application's database schema. Migrations are typically paired with Laravel's schema builder to build your application's database schema. If you have ever had to tell a teammate to manually add a column to their local database schema, you've faced the problem that database migrations solve.

The Laravel `Schema` facade provides database agnostic support for creating and manipulating tables across all of Laravel's supported database systems.

Generating Migrations

To create a migration, use the make:migration Artisan command:

```
php artisan make:migration create_users_table
```

The --table and --create options may also be used to indicate the name of the table and whether or not the migration will be creating a new table. These options pre-fill the generated migration stub file with the specified table:

```
php artisan make:migration create_users_table --create=users

php artisan make:migration add_votes_to_users_table --table=users
```

Migration Structure

A migration class contains two methods: `up` and `down`. The `up` method is used to add new tables, columns, or indexes to your database, while the `down` method should reverse the operations performed by the `up` method.

For example, the following migration creates a `flights` table:

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateFlightsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('flights', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('flights');
    }
}
```

Running Migrations

To run all of your outstanding migrations, execute the `migrate` Artisan command:

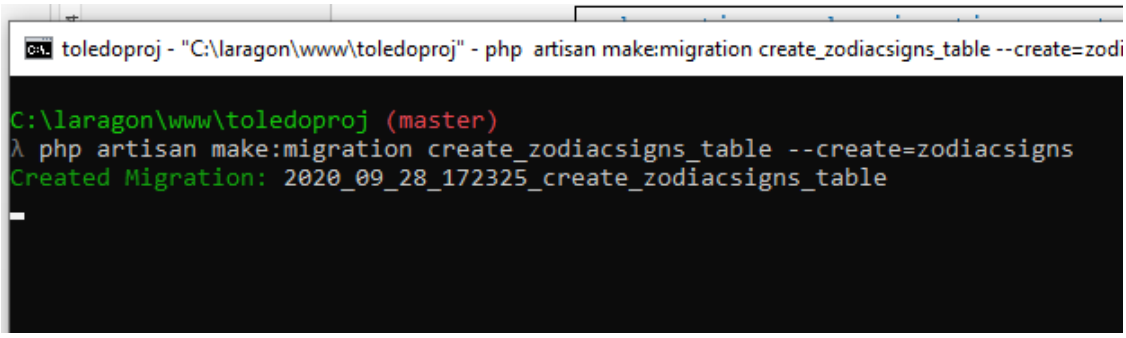
```
php artisan migrate
```

EXERCISE:

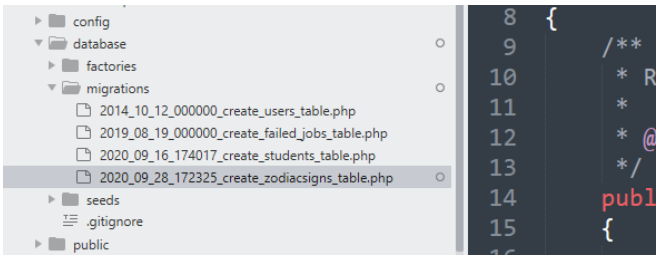
1. How to use migration using `php artisan make:migration --create=` to create the database table.

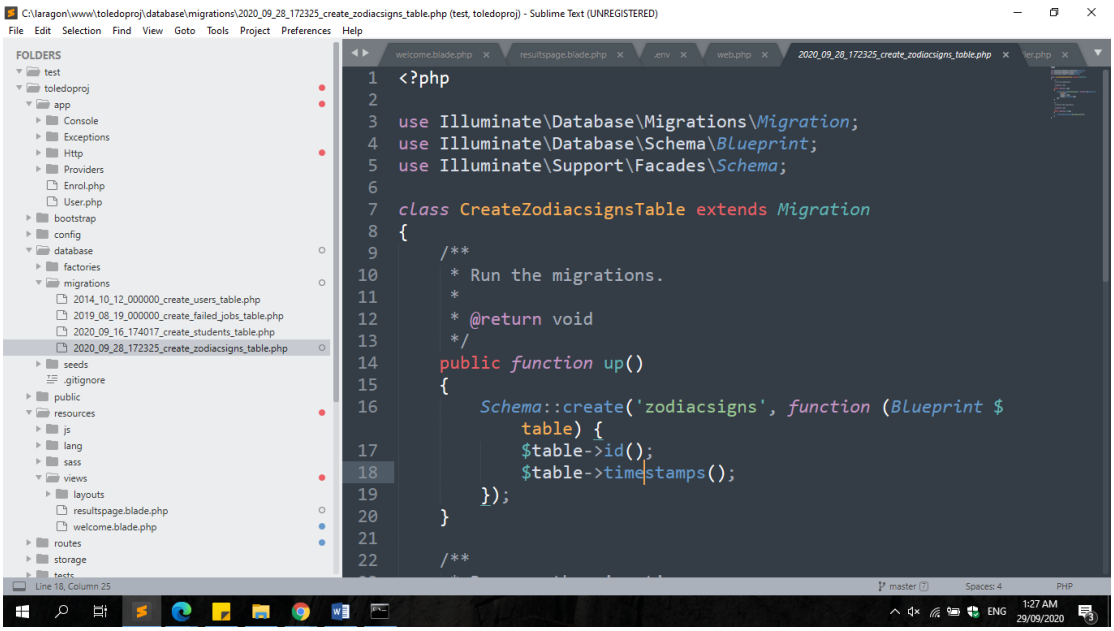
In the terminal, type the command:

```
 php artisan make:migration create_zodiacsigns_table --
create=zodiacsigns
```



Then, Go to **database/migrations** and you'll the created migration.





LESSON 11

Schema: Table, Row, and Indexes

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Create tables, row, and indexes (e.g. Primary key or Unique).

Creating Tables

To create a new database table, use the create method on the Schema facade.

```
Schema::create('users', function (Blueprint $table) {
    $table->id();
});
```

To drop an existing table, you may use the drop or dropIfExists methods:

```
Schema::drop('users');
Schema::dropIfExists('users');
```

Column Modifiers

In addition to the column types listed above, there are several column "modifiers" you may use while adding a column to a database table. For example, to make the column "nullable", you may use the nullable method:

```
Schema::table('users', function (Blueprint $table) {
    $table->string('email')->nullable();
});
```

For more info: https://laravel.com/docs/7.x/migrations#columns

Creating Indexes

The Laravel schema builder supports several types of indexes. The following example

creates a new email column and specifies that its values should be unique. To create the index, we can chain the unique method onto the column definition:

```
$table->string('email')->unique();
```

Alternatively, you may create the index after defining the column. For example:

```
$table->unique('email');
```

You may even pass an array of columns to an index method to create a compound (or composite) index:

```
$table->index(['account_id', 'created_at']);
```

LESSON 12

Model

LEARNING OUTCOMES

After studying this lesson, you should be able to:

1. Create a model using php artisan command.

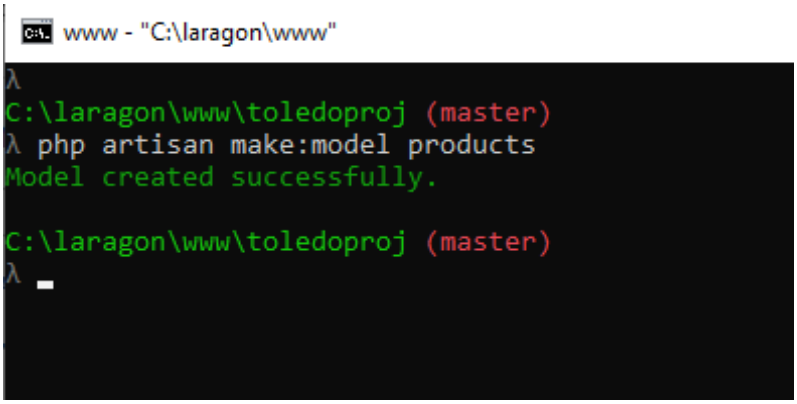
Defining Models

To get started, let's create an Eloquent model. Models typically live in the app directory, but you are free to place them anywhere that can be auto-loaded according to your composer.json file. All Eloquent models extend Illuminate\Database\Eloquent\Model class.

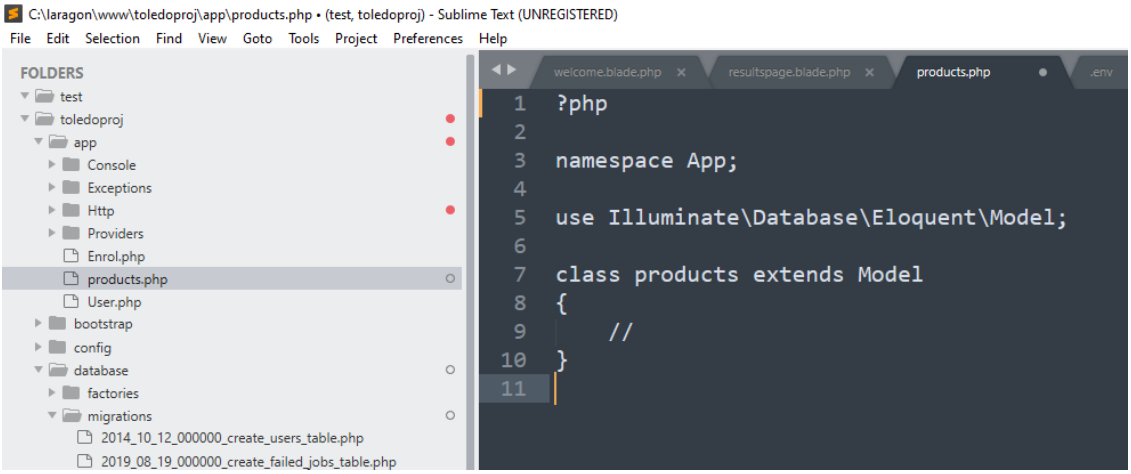
The easiest way to create a model instance is using the `make:model` Artisan command:

```
php artisan make:model Flight
```

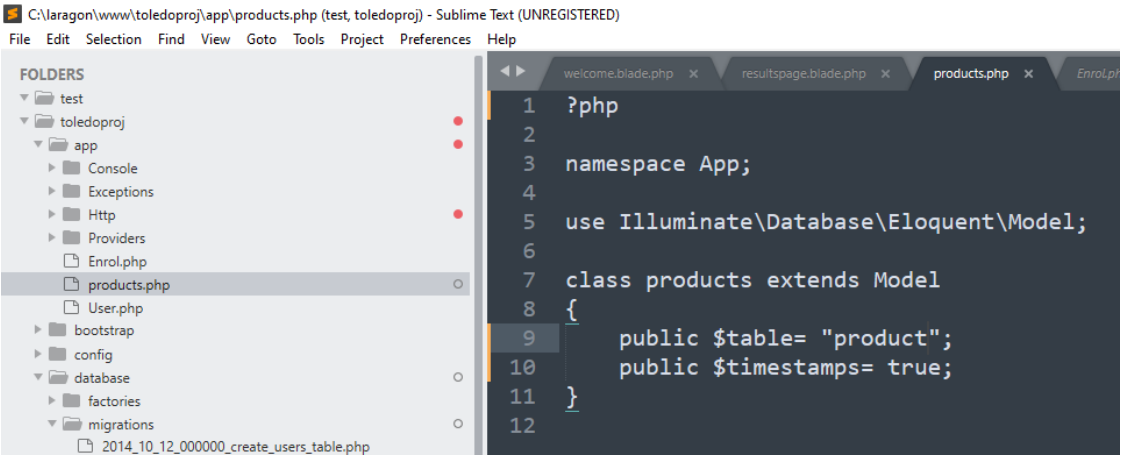
EXERCISE:



Next, find your created Model. Go to **app/**.



Add the following codes:



POST-TEST

LABORATORY MODULE 4:

1. Create a table "products" using migrations php artisan and add columns or fields.

Table name: **products**	
ID	INT
Product_name	VarChar
Types	VarChar
Descriptions	VarChar
price	Double or Decimal
updated_at	Timestamps
Created_at	Timestamps

2. Add model "product" using php artisan command, then add:

```
class products extends Model
  {
        public $table= "product";
        public $timestamps= true;
  }
```

REFERENCES

BOOK

Laravel: Up and Running, O'Reilly Media, Inc. (2016).

INTERNET

https://laravel.com/docs/
https://www.parthpatel.net/laravel-tutorial-for-beginner/#Whats_next_for_this_project