Laravel server side validation:-

$("#student\_form").on("submit",function(e){

e.preventDefault();

// var \_token=$("input[name=\_token]").val();

// var name=$("input[name=name]").val();

// var email=$("input[name=email]").val();

// var password=$("input[name=password]").val();

$.ajax({

url : $(this).attr('action'),

type: $(this).attr('method'),

data: new FormData(this),

processData:false,

// dataType:'json',

contentType:false,

beforeSend:function()

{

$(document).find('.error').text('');

},

success:function(data)

{

console.log(data.status);

if(data.status=='0'){

$.each(data.error,function(key,value){

$("."+key+"\_err").text(value[0]);

});

}else{

alert(data.message);

}

}

});

});

<form method="post" action="/student-save" id="student\_form">

@csrf

<div class="form-group">

<label>Name : </label>

<input type="text" name="name" class="form-control">

<span class="error name\_err"></span>

</div>

<div class="form-group">

<label>Email : </label>

<input type="text" name="email" class="form-control">

<span class="error email\_err"></span>

</div>

<div class="form-group">

<label>Mobile : </label>

<input type="text" name="mobile" class="form-control">

<span class="error mobile\_err"></span>

</div>

<input type="submit" value="Save" class="btn btn-info">

</form>

public function saveStudent(Request $request){

$validator=Validator::make($request->all(),[

'name' =>'required',

'email' =>'required',

'mobile' =>'required'

]);

if(!$validator->passes()){

return response()->json(["status"=>0,"error"=>$validator->errors()->toArray()]);

}

$data=[

"name"=>$request->input('name'),

"email"=>$request->input('email'),

"mobile"=>$request->input('mobile')

];

$sql=DB::table("students")->insert($data);

return response()->json(["status"=>1,"message"=>"Data saved successfully"]);

}

Queue Mail

Step 1:-

First you migrate your migration.

Step 2:-

run cmd for queue table.

php artisan queue:table

php artisan migrate

then create a new table, which name is jobs

Step 3:-

Create Queue Job

php artisan make:job SendEmailJob

now you can see in app folder create a nwe folder, which name is 'jobs', and create a file inside jobs table.

open SendEmailJob.php file

use Mail;

use App\Models\User;

protected $details; ye kisi function me nahi rahega. only class ke under rahega.

public function \_\_construct($details)

{

$this->details = $details;

}

public function handle()

{

$email=$this->details['email'];

$name=$this->details['name'];

$input['subject'] = $this->details['subject'];

$dat=["email"=>$email];

$input['email'] = $email;

$input['name'] = $name;

Mail::send('welcome', $dat, function($message) use($input){

$message->to($input['email']);

$message->subject($input['subject']);

});

}

//for send bulk sms

public function handle()

{

$data=DB::table('users')->get();

$input['subject'] = $this->details['subject'];

$dat=["email"=>"hello"]; //send data to view

foreach ($data as $key => $value) {

$input['email'] = $value->email;

$input['name'] = $value->name;

\Mail::send('email', $dat, function($message) use($input){

$message->to($input['email'], $input['name'])

->subject($input['subject']);

});

}

}

Step 4:-

create a controller

public function sendmail(Request $request)

{

$details = [

'subject' => 'Welcome Mail',

'email'=>'suman.krgr8@gmail.com',

'name'=>'suman'

];

// send all mail in the queue.

$job = (new \App\Jobs\SendEmailJob($details))

->delay(

now()

->addSeconds(2)

);

dispatch($job);

return back();

echo "mail send successfully in the background...";

}

php artisan queue:work

Breze

install breeze:-

composer require laravel/breeze --dev

php artisan breeze:install

use illuminate\Support\Facades\Auth;

get login user ful detal

Auth::user()

if you want get only id of logined user

Auth::id();

if you want to get only name of logined user

Auth::user()->name;

if you want to get logined user detail from request

public function index(Request $request)

{

return $request->user();

}

if you want to check user is logined or not

if(Auth::check())

{

//

}

if you want to restrist route, logined user access and without logined not access

Route::get('/get',[PostController::class,'index'])->middleware('auth');

ye middleware default kernal.php me add rahta hai

'auth' => \App\Http\Middleware\Authenticate::class

Seeder

How to use faker:-

first create a seeder

php artisan make:seeder StudenSeeder

use Faker\Factory as Faker;

public function run()

{

$faker=Faker::create();

foreach(range(1,10) as $value){

DB::table('students')->insert([

"name" => $faker->name(),

"city" => $faker->city(),

"fee" => $faker->randomFloat(2),

'gender' => $faker->randomElement($array = array ('male', 'female')) ,

]);

}

}

Now go to databaseSeeder.php

public function run(){

$this->call([

StudentSeeder::class,

]);

}

Now call seeder

php artisan db:seed

How to run specific seeder

php artisan db:seed --class=StudentSeeder

GATES

how to define gates:-

define(),allow(),denies(),forUser(),any(),none(),authorize(),check(),can(),cannot(),insepect(),before()

after()

on blade:-

@can

@cannot

@canany

Where write the gates method:-

Gates are define within the boot method of the App\Provider\AuthServiceProvider

use illuminate\Support\Facades\Gate;

public function boot()

{

$this->registerPolicies();//allready exist in boot method

Gate::define('isAdmin',function($user){

if($user->email==='suman@gmail.com')

{

return true;

}

else

{

return false;

}

});

}

Another way to write Gate code:

create a folder in App, Folder name Gate

App\Gate\AdminGate.php

<?php

namespace App\Gate;

class AdminGate

{

public function check\_admin($user)

{

if($user->email==='suman@gmail.com')

{

return true;

}

else

{

return false;

}

}

}

Now you go to boot method of AuthServiceProvider

use illuminate\Support\Facades\Gate;

use App\Gate\AdminGate;

public function boot()

{

$this->registerPolicies();

Gate::define('isAdmin',[AdminGate::class,'check\_admin']);

}

How to check by middleware

Route::get('/post',[PostController::class,'index'])->middleware(['auth','can:isAdmin']);

How to check on blade

@can('isAdmin')

//

@endcan

@cannot()...@endcannot

$canany()...@endcanany

How to check on controller:-

public function edit($id)

{

$post=Post::find($id);

if(Gate::deines('isAdmin',$post))

{

abort(403);

}

return view('edit',compact('post'));

}

CUSTOM FAÇADE

Step 1 :-

Create A folder In App Like 'Suman' and create a file like Invoice.php

Invoice.php

<?php

namespace App\Suman;

use Carbon\Carbon;

class Invoice {

public function companyName(){

return 'Theequicom pvt ltd';

}

}

Step 2 :-

Create another file in app->suman foler like InvoiceFacade.php

<?php

namespace App\Suman;

use Illuminate\Support\Facades\Facade;

class InvoiceFacade extends Facade{

protected static function getFacadeAccessor()

{

return 'invoice';

}

}

Step 3:-

Create a service provider class

php artisan make:provider SumanServiceProvider

it's create a file in app->provider

use App\Suman\Invoice;

public function boot()

{

$this->app->bind('invoice',function(){

return new Invoice();

});

}

Now go to app->confic

provider section

App\Providers\SumanServiceProvider::class,

aliases section

'Invoice'=> App\Suman\InvoiceFacade::class

echo Invoice::companyName();

Multiple Database connection

On model

class McibModel extends Model {

protected $connection= 'second\_db\_connection';

protected $table = 'agencies';

}

return array(

'connections' => array(

'mysql' => array(

'driver' => 'mysql',

'host' => 'localhost',

'database' => 'database1',

'username' => 'user1',

'password' => 'pass1'

'charset' => 'utf8',

'collation' => 'utf8\_unicode\_ci',

'prefix' => '',

),

'second\_db\_connection' => array(

'driver' => 'mysql',

'host' => 'localhost',

'database' => 'database2',

'username' => 'user2',

'password' => 'pass2'

'charset' => 'utf8',

'collation' => 'utf8\_unicode\_ci',

'prefix' => '',

),

),

How to specific database connect by query builder

$programs=DB::connection('mysql2')

->table('node')

->where('type', 'Programs')

->get();

EXCEL IMPORT

composer require maatwebsite/excel

php artisan migrate

php artisan make:import UsersImport --model=User

use App\Imports\ExcelImport;

use Excel;

public function excelSave(Request $request){

$file=$request->file('excelfile');

$data=Excel::Import(new ExcelImport,$file);

dd($data);

}

App->Import

use Illuminate\Support\Collection;

use Maatwebsite\Excel\Concerns\ToCollection;

use Maatwebsite\Excel\Concerns\WithMultipleSheets;

use App\Imports\StudentImport;

use App\Imports\AccountImport;

class ExcelImport implements WithMultipleSheets

{

public function sheets() : array

{

return [

0=>new StudentImport,

1=>new AccountImport

];

}

}

App->import->StudentImport

use Illuminate\Support\Collection;

use Maatwebsite\Excel\Concerns\ToCollection;

use DB;

use Illuminate\Support\Facades\Validator;

use Maatwebsite\Excel\Concerns\WithHeadingRow;

class StudentImport implements ToCollection,WithHeadingRow

{

public function collection(Collection $collection)

{

Validator::make($collection->toArray(), [

'\*.name' => 'required',

'\*.email' => 'required',

'\*.mobile' => 'required',

])->validate();

foreach($collection as $row){

DB::table('students')->insert([

"name"=>$row['name'],

"email"=>$row['email'],

"mobile"=>$row['mobile']

]);

}

}

}

App->import->Accountimport

use Illuminate\Support\Collection;

use Maatwebsite\Excel\Concerns\ToCollection;

use DB;

class AccountImport implements ToCollection

{

/\*\*

\* @param Collection $collection

\*/

public function collection(Collection $collection)

{

foreach($collection as $row){

DB::table('users')->insert([

"name"=>$row[0],

"email"=>$row[1],

"password"=>$row[2]

]);

}

}

}

On View File :

<div class="col-xl-6">

@if (count($errors) > 0)

<div class="row">

<div class="col-md-8 col-md-offset-1">

<div class="alert alert-danger alert-dismissible">

<button type="button" class="close" data-dismiss="alert" aria-hidden="true">×</button>

<h4><i class="icon fa fa-ban"></i> Error!</h4>

@foreach($errors->all() as $error)

{{ $error }} <br>

@endforeach

</div>

</div>

</div>

@endif

<form method="post" action="excel-import" enctype="multipart/form-data">

@csrf

<div class="form-group">

<label>Excel</label>

<input type="file" name="excelfile" class="form-control">

</div>

<input type="submit" value="Upload" class="btn btn-info">

</form>

</div>

EVENT

php artisan make:event SendMail

app/Events/SendMail.php

public $userId;

public function \_\_construct($userId)

{

$this->userId = $userId;

}

php artisan make:listener SendMailFired --event="SendMail"

app/Listeners/SendMailFired.php

public function handle(SendMail $event)

{

$user = User::find($event->userId)->toArray();

Mail::send('emails.mailEvent', $user, function($message) use ($user) {

$message->to($user['email']);

$message->subject('Event Testing');

});

}

app/Providers/EventServiceProvider.php

protected $listen = [

'App\Events\SendMail' => [

'App\Listeners\SendMailFired',

],

];

On Controller

use Event;

use App\Events\SendMail;

public function index()

{

Event::fire(new SendMail(2));

return view('home');

}

Event :-

To simplify, we can say an event is an action taken in the application, and the listener is the operation that responds to the event.

Middleware

Session check by Middleware:-

php artisan make:middleware CustomAuth;

now go to CustomAuth.php middleware

import

use Illuminate\Http\Request;

use Session;

public function handle(Request $request, Closure $next)

{

if(!session('user'))

{

return redirect()->route('user.index');

}

return $next($request);

}

On Controller:-

function login(Request $request)

{

$email=$request->email;

$password=$request->password;

$data=DB::table('users')->where(['email'=>$email,'password'=>$password])->select('name','email')->first();

if($data)

{

$request->session()->put('user',$email);

return redirect('admin/dashboard');

}

else

{

return "Not";

}

}

Now go to kernel.php

'myauth'=>[

\App\Http\Middleware\CustomAuth::class,

],

Now on route:-

Route::get('admin/login', function () {

if(Session::has('user'))

{

return view('admin.dashboard');

}

return view('admin.login');

})->name('user.index');

Route::group(['middleware'=>'myauth'],function(){

Route::view('/admin/register','admin.register')->name('user.register');

Route::Post('user/signup',[UserController::class,'signup'])->name('user.signup');

Route::post('user/login',[UserController::class,'login'])->name('user.login');

Route::view('/admin/dashboard','admin.dashboard')->name('user.dashboard');

});

Middleware

Laravel Middleware acts as a bridge between a request and reaction.

Middleware provide a convenient mechanism for filtering HTTP request. For example, Laravel includes a middleware that verifies the user is authenticated or not. If the user is not authenticated, the middleware will redirect the user to the login screen. However, if the user is authenticated, the middleware will allow the request to proceed further into the application.

Pagination

public function index(Request $request)

{

$users = User::paginate(5);

return view('users', compact('users'));

}

{!! $users->links() !!}

Throting:-

In Laravel we use throttle middleware to restrict the amount of traffic for a given route or group of routes. The throttle middleware accepts two parameters that determine the maximum number of requests that can be made in a given number of minutes.

Route::middleware('throttle:60,1')->get('/user', function () {

//

});

Service provider

Service providers in laravel application is the central place where application is bootstrapped. That is, laravel's core services and our application's services, classes and their dependencies are injected in service container through providers.

If you open the config/app.php, file you will see a providers array, here all default service provider are listed.

All service providers extend the Illuminate\Support\ServiceProvider class. Most service providers contain a register and a boot method. Within the register method, you should only bind things into the service container.

public function register()

{

$this->app->bind('SiteRepo', function($app) {

return new SiteRepo();

});

}

if we need to register a view composer within our service provider? This should be done within the boot method.

View::composer(['demo1', 'demo2'], function($view) {

$data=DB::table('students')->get();

$view->with('key', $data);

});

Service container

The Laravel service container is a powerful tool for managing class dependencies and performing dependency injection.

So whenever you want to inject a service into other services, you can add it into constructor or method, and it’s injected automatically from service container by the service provider.

What is facade

Facades provide a "static" interface to classes that are available in the application's service container.

What is faker

Faker is a PHP package that generates dummy data for testing. With Faker you can generate mass amount of testing data as you needed.

What is seeder

With the help of seeder you can insert dulk dummy data into your database

first create a seeder

php artisan make:seeder StudenSeeder

use Faker\Factory as Faker;

public function run()

{

$faker=Faker::create();

foreach(range(1,10) as $value){

DB::table('students')->insert([

"name" => $faker->name(),

"city" => $faker->city(),

"fee" => $faker->randomFloat(2),

]);

}

}

Now go to databaseSeeder.php

public function run(){

$this->call([

StudentSeeder::class,

]);

}

Now call seeder

php artisan db:seed

How to run specific seeder

php artisan db:seed --class=StudentSeeder

What is dependency injection

The Laravel service container is a powerful tool for managing class dependencies and performing dependency injection. Dependency injection is a fancy phrase that essentially means this: class dependencies are "injected" into the class via the constructor or, in some cases, "setter" methods.