# LARAVEL Guide

## Installation:

Install composer (www.getcomposer.org/)

- https://www.geeksforgeeks.org/how-to-install-php-composer-on-windows/

Install Laravel installer using command

- Composer global require Laravel/installer

## Project:

Make a project:

- ➢ via composer

```
composer create-project laravel/laravel example
```

OR

- ➢ via Laravel installer

```
laravel new {example-app}
```

> Start the server/application

```
php artisan serv
```

## Routing:

Loading Simple View

```
Route::get('/', function(){
    return view("index");
});
```

Redirection on particular view

```
Route::get('/', function(){

    return redirect("welcome");
});
```

User can grab the query parameter using

`request(“query-string”);`

Dynamic Routing:

```
Route::get('/{product}/{id}', function ($p,$id) {
    return "Product :".$p." with id: ".$id;
});
```

Question mark at the end of parameter like id? Will make it optional parameter. E.g

```
Route::get('/{product}/{id?}', function ($p,$id=null) {
    return "Product :".$p." with id: ".$id;
});
```

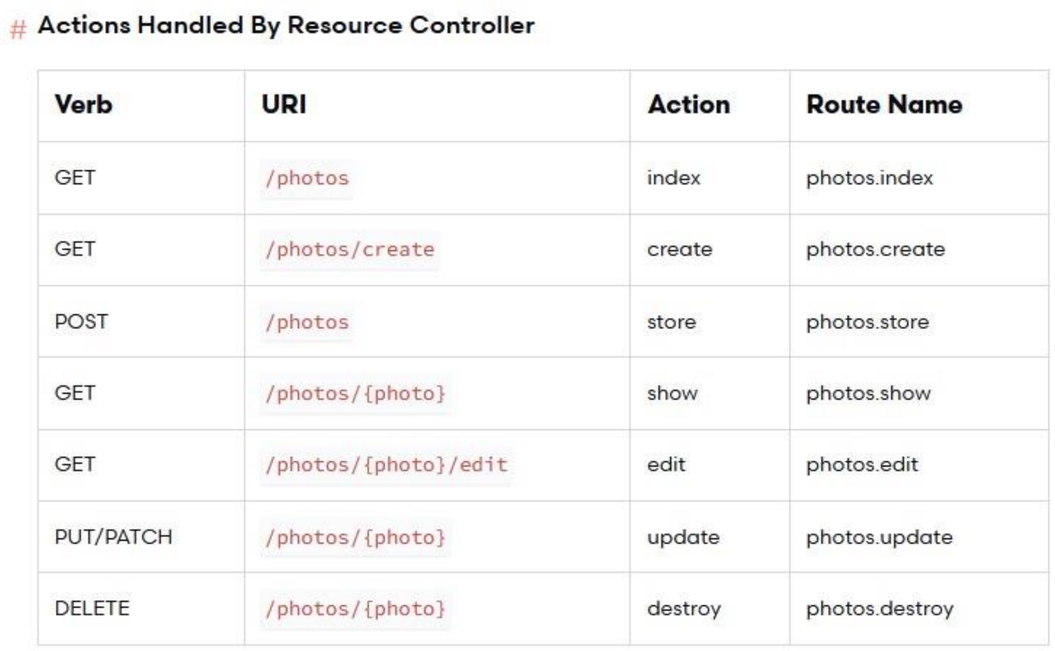
Resource Route (you register CRUD function using resource) like

`Route::resources(  'photos'  =>  PhotosController::class);`

For multiple resources

```
Route::resources([
    'photos' => PhotoController::class,
    'posts' => PostController::class,
]);
```

by default, resource provides following actions



## # Actions Handled By Resource Controller

| Verb | URI | Action | Route Name |
|---|---|---|---|
| GET | `/photos` | index | photos.index |
| GET | `/photos/create` | create | photos.create |
| POST | `/photos` | store | photos.store |
| GET | `/photos/{photo}` | show | photos.show |
| GET | `/photos/{photo}/edit` | edit | photos.edit |
| PUT/PATCH | `/photos/{photo}` | update | photos.update |
| DELETE | `/photos/{photo}` | destroy | photos.destroy |

Passing data to view from url:

```
Route::get('/{name}', function ($n) {
    return view("welcome",['name'=>$n]);
});
```

```
> Calling Controller from Larvel < 8
```

```
 Route::get('/', "PageController@index");
```

```
> Calling Controller from Larvel > 8
```

```
 use App\Http\Controllers\PageController;

 Route::get('/', [PageController::class,'index']);
```

Naming a route

`Route::get('/', [HomeController::class, 'show'])->name('home');`

Use route() instead of url()

Like

`<a href=”{{route(‘home’)}}”>Home</a>`

# Controller

Controller using simple controller command

```
php artisan make:controller {name}
```

```
Route::get('/products/{id}', [productController::class,'show'])
```

> Calling Controller with data, also Validation before sending to Controller

```
Route::get('/products/{id}', [productController::class,'show'])-
>where('id','[0-9]+')
```

you can create both model and resource controller using single command like

```
php artisan make:controller {name} --resource --model={model_name}
```

---

## Blading/View

Call another view from a view

```
 @include("header")
```

Passing data to view

If

```
    private static function getdata()
    {
        return [
            ['id'=>1, 'name'=>"How to code", "author"=>"Tim Lee"],
            ['id'=>2, 'name'=>"information Security", "author"=>"Ghome"],
            ['id'=>3, 'name'=>"Lean me", "author"=>"Saw N"]
        ];
    }
```

Pass this book’s data to view like

```
view('about',['books'=>self::getdata()]);
```

## Managing Layouts:

### Static Resource

Include static resources in your pages like external css files

```
<link rel="stylesheet" type="text/css" href="{{url('css/style.css')}}">
```

## Processing HTML Form with Validation

Use @csrf inside every form, even in every Javascript code

$Request object is passed from route if you used POST method in Route file, and you can display the form data access $req->input();

you can also validate form using

```
$req->validate([
        username => 'required | max:6',
       'userpass'=>  'required',
       'myimage' => 'required | mimes:jpg,jpeg,png|max:5048'
]);
```

Errors of the form can be displayed like {{$errors}}, or using @foreach

```
          @if ($errors->any())
               <div class="alert alert-danger">
                   <strong>Whoops!</strong> There were some problems with your input.<br><br>
                        <ul>
                        @foreach ($errors->all() as $error)
                            <li>{{ $error }}</li>
                         @endforeach
                        </ul>
                </div>
           @endif
```

you can print individual errors like

```
@error('username')

{{ $message }}

@enderror
```

## Database

Set database setting in in env file
import DB Support

```
use Illuminate\Support\Facades\DB;
```

Perform a query

```
DB::select("select * from users");
```

Query Builder

```
DB::table('users')->get();
DB::table('users')->where('id',4)->get();
DB::table('users')->count();
(array)DB::table('users').find(6);
```

Insert

```
DB::table('users')->insert([
     'name'=>'Azhaan',
     'email'=>'mzsabir@gmail.com'
]);
```

Update

```
DB::table('users')
->where('id',22)
->insert([
     'name'=>'Azhaan',
     'email'=>'mzsabir@gmail.com'
]);
```

Delete

```
DB::table('users')->where('id',22)->delete();
```

Join

```
DB::table('users')
 -> join('company','users.id=','=','comapny.user_id')
 -> select('users.*')
 -> where('id',4);
 -> get();
```

## Http Client

import Http Support

```
use Illuminate\Support\Facades\Http;
```

You can get any API (json) data like

Http::get("www.example.com");

HTTP Method types

GET
POST
PUT
DELETE
PATCH
OPTIONS

you can use @method('PUT') after @csrf

## Session Management

Add to session

```
session()->put('key','value');
```

check the session

```
if(session()->has('key'))
```

OR

```
session()->exists('users')
```

Remove from the session

```
session()->pull('key');
```

Retrieving All Session Data

```
session()->all();
```

## Flash Session

you can call a view with data using flash session like

```
return view('blog.show')->with('post',Post::where('id',4));
```

## Upload a file

Upload a file

```
file('mypdf')->store('folder');
```

move the file abc.jpg to images

```
image->move(public_path('images'),"abc.jpg");
```

Get name of Uploaded file

```
file->getClientOriginalName();
```

Get extension of Uploaded file

```
file->image->extension();
```

complete example

```
  $req->validate([
        'file' => 'required|mimes:csv,txt,xlx,xls,pdf|max:2048'
        ]);

 $fileModel = new File;

if($req->file()) {
     $fileName = time().'_'.$req->file->getClientOriginalName();
     $filePath = $req->file('file')->storeAs('uploads', $fileName, 'public');

            $fileModel->name = time().'_'.$req->file->getClientOriginalName();
            $fileModel->file_path = '/storage/' . $filePath;
            $fileModel->save();
```

# Pagination

instead of

`model::all()` you can use model::paginate(5)

Page links

`{{$object->links()}}` // here $object represent array containing data

# Migration

Create a table in migration

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

```
->increments('id')->unique();
->timestamps();
->string('field')->nullable();
->longText('description');
->integer(age);
->unsignedBigInteger('user_id');
->forign('user_id')->references('id')->on('users');
```

start migration (move the table in migration to DB)

```
php artisan migrate
```

`Error Solution:` if you get error like: Specified key was too long

then go to ***boot*** function of (app/providers/appServiceProvider.php) and add following line

```
\Illuminate\Support\Facades\Schema::defaultStringLength(191);
```

(note: you might need to run your app again using php artisan serv)

Reset your migration

```
php artisan migrate:reset
```

Rollback last migration only

```
php artisan migrate:rollback
```

Rollback last 3 steps

```
php artisan migrate:rollback --step 3
```

Refresh the DB if any changed occurred in migration (table structure etc)

```
php artisan migrate:refresh
```

you can migrate single table as well

## Slug

Install sluggable package

```
composer require cviebrock/eloquent-sluggable
```

Inside the model

```
use Cviebrock\EloquentSluggable\Services\SlugService;

public function sluggable():array{
        return [
            'slug'=>['source'=>'title']
        ];
    }
```

Now you can create slug using

```
$slug=SlugService::createSlug(User::class,'slug',$req->input('uname'));
```

// parameter-1: Model

// parameter-2: 'slug'

// paramter-3: string value which will be converted to slug

## Other functions

`uniqueid() return unique number`

## Cache

### Clear Application Cache

```
php artisan cache:clear
```

## Clear Route Cache

```
php artisan route:clear
```

## Clear Configuration Cache

```
php artisan config:clear
```

## Clear Compiled Views Cache

```
php artisan view:clear
```