* Clone your project
* Go to the folder application using cd command on your cmd or terminal
* Run composer install on your cmd or terminal
* Copy .env.example file to .env on the root folder. You can type copy .env.example .env if using command prompt Windows or cp .env.example .env if using terminal, Ubuntu
* Open your .env file and change the database name (DB\_DATABASE) to whatever you have, username (DB\_USERNAME) and password (DB\_PASSWORD) field correspond to your configuration.
* Run php artisan key:generate
* Run php artisan migrate
* Run php artisan serve
* Go to http://localhost:8000/

**Подключение**

-Устанавливаем Composer

-Через командную строку устанавливаем Laravel(как указано на сайте Laravel)

-Устанавливаем наш новый проект Laravel в C:\OSPanel\domains(если установлен Open Server)

-В настройках Open Server и в разделе "Домены" настроим новый домен для нашего нового проекта. Домашний каталог у Laravel (там где лежит файл index.php) находится в каталоге "public", и именно его мы должны указать в качестве папки домена. Имя домена "mysite.loc" может быть любое



***Структура и имена файлов при CRUD***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Verb**(метода запроса в Route) | **URI**(имя страницы в Route) | **Action**(имя функции в Controller) | **Route Name**(имя страницы внутри проекта) | **Пояснение** |
| GET | /photos | index | photos.index | отображение на странице ВСЕХ данных из БД(список всех отзывов о товаре) |
| GET | /photos/create | create | photos.create | страница на которой заполняются данные для внесения в БД(форма в которой пишешь свой отзыв, свои данные)  в Controller:  public function store() { $data=request()->validate([ 'title'=>'string', ]); Post::create($data);  return redirect()->route('post.index'); } |
| POST | /photos | store | photos.store | отправка данных в БД  в View:  <form action=  "{{route('post.store')}}" method="POST">  @csrf |
| GET | /photos/{photo} | show | photos.show | отображение на странице ОДНОГО элемента из БД(один подробный отзывов о товаре)  в Controller:  public function show(Post $post) { return view('post/show', compact('post')); }  Post-Модель  $post-то что и имени запроса в { } но с $ |
| GET | /photos/{photo}/edit | edit | photos.edit | страница с отображением ОДНОГО элемента из БД, который можно редактировать(редактирование своего отзыва о товаре) |
| PUT/PATCH | /photos/{photo} | update | photos.update | отправка изменений в ОДНОМ элементе в БД  в Controller:  public function update(Post $post) { $data = request()->validate([ 'title' => 'string', ..., ]); $post->update($data); return redirect()->route('post.show', $post->id); }  в View:  <form action="{{route('post.update')}}" method="POST"> @csrf @method('patch') |
| DELETE | /photos/{photo} | destroy | photos.destroy | удаление ОДНОГО элемента из БД  в Controller:  public function destroy(Post $post) { $post->delete(); return redirect()->route('post.index'); }  в View:  <form action="{{route('post.delete', $post->id)}}" method="POST"> @csrf @method('delete') <button type="submit" >Delete</button> </form> |

***Функции(хэлперы)***

**protected $table='имя таблицы';** – в Model указывать имя то таблицы с которой связана Model

**dd(переменная)** – показывает значение и останавливает выполнение кода

**dump(переменная)** – показывает значение без остановки выполнения кода

***Eloquent ORM***

Laravel содержит [ORM-библиотеку](https://ru.wikipedia.org/wiki/ORM) Eloquent, которая работает с базой данных. Eloquent использует связь Model со своей БД.

**find(id)** – возвращает элемент из БД соответствующий id

$user = User::find(1);

**create([])** –добавляет элементы в БД(в Model необходимо вставить protected $guarded=[] или protected $guarded=false;)

$user = User::create([

‘title’=>’some’,

…]);

**update([])** –изменяет существующий элементов в БД

$user = User::find(…)->update([

‘title’=>’update some’,

…]);

**delete()** –удаляет существующий элементов в БД(В Миграции(создание таблицы) указать $table->SoftDeletes()-данные останутся в таблице и их можно восстановить и в Model указать use SoftDeletes;)

$user = User::find(…)->delete();

**restore()** –восстанавливает удаленный элементов в БД(если он был удален при помощи SoftDeletes)

$user = User:: withTrashed() ->where('airline\_id', 1) ->restore()

**firstOrCreate([проверка], [новый элемент])** –возвращает элемент БД соответствующий условиям(например значение определенного атрибута), а если такого нет то создаст *Новый* и возвращает его. Можно использовать, чтобы избежать дублирования

**updateOrCreate([проверка], [обновление элемента или новый элемент])** – тоже что и firstOrCreate(), только если находится элемент соответствующий условию в него вносятся изменения и возвращает измененный элемент

***Методы Laravel***

**all()** –возвращает массив из Коллекция(collect). Позволяет получить все элементы БД

$user = User::all();

**where(‘ключ’, ‘значение’)** –фильтрует Коллекция(collect) по “ключ” / “значение”

$posts=User::where('is\_published', 1)->get();

-or-

$posts=User:: all()->where('is\_published', 1);

**first()** –возвращает первый элемент Коллекция(collect). Позволяет получить первый элемент БД

$user = User::first();

***Controller***

**Создание Controller**

**-**в командной строке php artisan make:controller [Name(CamelCase)Conroller]

Структура

***Migration***

**Создание Migration(создание таблицы) c Model**

**-**в командной строке php artisan make:model [Name] -m

-подключение к базе: файл .env заменить DB\_DATABASE на имя вашей базы

**Накатить(выполнить функцию up) Migration**

**-**в командной строке php artisan migrate

**Oткатить(выполнить функцию down) Migration**

**-**в командной строке php artisan migrate:rollback

**Обновить Migration**

**-**в командной строке php artisan migrate:fresh

**Добавить колонку в Migration**

**-**в командной строке php artisan make:migration [add\_colomun\_’имя атрибута’\_to\_’имя миграции где создавалась таблица(2023\_01\_24\_084927\_create\_posts\_table.php)’]

public function up() {

Schema::table('posts', function (Blueprint $table) {

$table->text('description')->nullable();

}); }

public function down() {

Schema::table('posts', function (Blueprint $table) {

$table->dropColumn('description'); }); }

**Удалить колонку в Migration**

**-**в командной строке php artisan make:migration [delete\_colomun\_’имя атрибута’\_to\_’имя миграции где создавалась таблица(2023\_01\_24\_084927\_create\_posts\_table.php)’]

public function up() {

Schema::table('posts', function (Blueprint $table) {

$table->dropColumn('description');

}); }

public function down() {

Schema::table('posts', function (Blueprint $table) {

$table->text('description')->nullable();

}); }

**Редактировать колонку(имя,тип данных..) в Migration**

**Если будет ошибка при php artisan migrate, то выполнить composer require doctrine/dbal**

**-**в командной строке php artisan make:migration [edit\_colomun\_’имя атрибута’\_to\_’имя миграции где создавалась таблица(2023\_01\_24\_084927\_create\_posts\_table.php)’]

public function up() {

Schema::table('posts', function (Blueprint $table) {

!name!

$table->renameColumn('content', ‘post\_content’);

!type!

$table->text('column\_name')->change();

}); }

public function down() {

Schema::table('posts', function (Blueprint $table) {

!name!

$table->renameColumn(' post\_content ', ‘content’);

!type!

$table->string('column\_name')->change();

}); }

**Удалить таблицу в Migration**

**-**в командной строке php artisan make:migration [delete\_’имя миграции где создавалась таблица(2023\_01\_24\_084927\_create\_posts\_table.php)’]

public function up() {

Schema::dropIfExists('posts')

public function down() {

Schema::create('posts', function (Blueprint $table) {

$table->id();

$table->text('description')->nullable();

…

}); }

***View***

**В Contoller в функции указываем – return view(‘file in folder view’, compact(таблица)), данные из compact(таблица) передаются в файл в папке view**

**Bootstrap – composer require Laravel/ui** after **php artisan ui bootstrap** after **npm run dev**

**<link rel="stylesheet" href="{{asset('css/app.css')}}"> - подключаем на странице стили bootstrap, с помощью asset путь всегда будет прописан правильно, даже если имя файла blade.php поменяется или его расположение**

**@foreach(array as item){{item}}endforeach** –перебор массива в файлах blade.php, при этом **item** выводится на html странице

**@yield(‘name’)** –создается шаблон на html странице с именем ‘name’

**@extends (‘folder/file’)** –на другой странице blade.php вставляется этот шаблон, с указанием пути к файлу

**@section (‘name’) ‘text’ endsection** – в этой же странице где @extends вставляется @section с тем же именем ‘name’ который подставляем свое значение ‘text’ вместо @yield

**->name(‘name’)** –присваиваешь имя страницы и далее в проекте можно использовать это name когда надо указать ссылку на страницу…

<a href="{{route('main.index')}}">Main</a>

**@error(‘title’/** атрибут name=’title’ например в input**){{$message/** переменная из Laravel выводит какая ошибка **}} enderror** –проверяет введено ли значение, если нет выводит <Тег>

в Controller

$data = request()->validate([ 'title' => **'required|string'**,…])

в View

<input type="text" name="title" value**="{{ old('title') }}/**сохраняет значение если оно введено, а в других полях нет **"**>

***Один ко многим***

*в Migration*

$table->unsignedBigInteger('category\_id)->nullable();

$table->index('category\_id', 'post\_category\_idx');

$table->foreign/**Внешний ключ**('category\_id'/**Имя колонки которая связана с другой таблицей**', 'post\_category\_fk')->on/**С какой таблицей связь**('categories')->references('id')/**Колонка с которой связь**;

**Быстрый доступ к связанным элементам из БД**

*в Controller*

public function index() {

$categories=Category::find(1);

dd($categories->posts/**имя функции из модели**); }

*в Model*

class Category extends Model {

public function posts() {

1 вариант - return $this->hasMany(Post::class/**Другая модель в связке**, 'category\_id'/**имя колонки с внешним ключом**, 'id'/**имя колонки из другой таблицы с которой связь**); } }

(return $this-> belongsTo(...)-одно значение ищем, выше много)

2 вариант-Имя колонки которая ссылается на другую таблицу должно быть как и имя Модели\_id для 2 таблицы и:

return $this->hasMany(Post::class), остальные атрибуты не нужны

***Многие ко многим***

*в Migration(есть 1 и 2 таблицы, чтобы сделать связь между ними многие ко многим создается таблица 3, в 3 уже создаётся две связи с один ко многим с 1 и 2 таблицей)*

$table->unsignedBigInteger('tag\_id');

$table->index('post\_id', 'post\_tag\_post\_idx');

$table->index('tag\_id', 'post\_tag\_tag\_idx');

*(связь с 1 таблицей)*$table->foreign('post\_id'/**Имя колонки которая связана с другой таблицей**', 'post\_tag\_post\_fk')->on('posts'/**С какой таблицей связь**)->references('id') )/**Колонка с которой связь**;

*(связь с 1 таблицей)*$table->foreign('tag\_id', 'post\_tag\_tag\_fk')->on('tags')->references('id');

**Быстрый доступ к связанным элементам из БД**

*в Controller*

public function index() {

$tag=Category::find(1);

dd($tag->posts/**имя функции из модели**); }

*в Model*

class **Tag** extends Model {

public function posts() {

1 вариант - return $this-> belongsToMany (**Post**::class/**Другая модель в связке**, ‘post\_tags’/**Таблица 3 связуюшая 1 и 2,** 'tag\_id'/**имя колонки с внешним ключом в модели Tag**, 'post\_id'/ **имя колонки с внешним ключом в модели Post**); } }

2 вариант -при создании миграции имя 3 таблицы должно быть php artisan

make:migration create\_1таблица\_2таблица\_table –create(Таблица 1 и 2 в ед.числе). Имя колонок в 3 Таблице, должны быть как имя Модели1\_id и Модели2\_id и:

return $this-> belongsToMany (Post::class), остальные атрибуты не нужны

-для того чтобы принимал в тегах ‘name’ большего 1 значения необходимо поставить [] name=’tags[]’

-из одной формы передать данные в 1 и 2 таблицы

В Controller

$tagss = $data['tags']; - данные для 2 таблицу

unset($data['tags']); - удаляем из общего массива данные для 2 таблицы

$post = Post::create($data);- ссылка на Модель для 1 таблицы

$post ->tags()/функция из Модели связующая 1 и 2 таблицу->attach/Метод Laravel($tagss);

когда редктируем элемент БД надо:

$post ->tags()/функция из Модели связующая 1 и 2 таблицу->sync/Метод Laravel($tagss);

***Однометодные контроллеры***

*в Controller*

public function **\_\_invoke**() {

без имени функции }

-если надо однометодные контроллеры сгруппировать в папку то:

Route::group(['namespace'=>'Post/**имя папки**'], function() {

Route::get('/posts', 'PostController /**указывается только имя контроллера без функции** ')->name('post.index');

…})

namespace App\Http\Controllers\**Post**-указать путь к папке к сгруппированным контроллером, стоит по умолчанию namespace App\Http\Controllers;

***Класс Request(данные из формы)***

Для получения данных формы в Laravel используется специальный объект запроса Request. Создание:

php artisan make:request [**Имя**Request] или если однометодные контроллеры php artisan make:request Post(**имя папки**)\[**Имя**Request] в созданном файле

public function authorize() {

return true; }

public function rules() {

return [

'title' => 'required|string'

…]; }

в Controller

public function \_\_invoke(StoreRequest $request) { $

data = $request->validated();

…}

***Класс Service(логика* по CRUD БД*)***

Создается для вынесения основной логики по CRUD(например-внесение данные из формы в БД) в отдельный класс, который взаимодействует со своим Controller. Что позволяет лучше понимать код

-в папке с однометодными контроллерами создаем BaseController.php в нем

class BaseController extends Controller {

public $service;

public function \_\_construct(Service/**имя класса в файле Service** $service/**задаем переменную-экземпляр класса**) { $this->service = $service; } }

-в тех однометодных контроллерах где подключаем Service надо наследование поменять на BaseController

class UpdateController extends BaseController {

public function \_\_invoke(Post $post, UpdateRequest $request) {

$data = $request->validated();

$this->service->update($post, $data/**данные которые надо передать в Service**); } }

-в папке app создаем Service там папку такую же как у однометодных контроллеров в ней файл Service

class Service {

public function update($post, $data) {

логика по CRUD} }