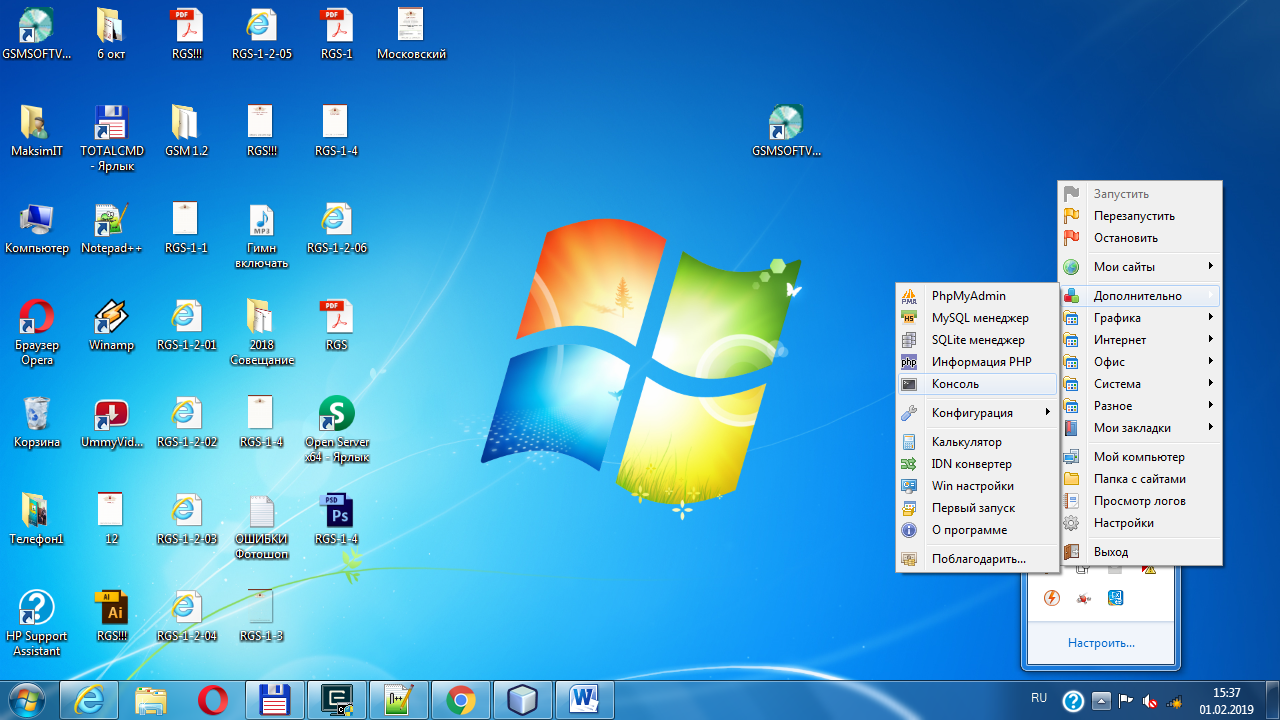
В этой части сделаем условно-динамический контент главной страницы. Мы создадим возможность изменения текста ее разделов.

В предыдущей главе мы создали страницу resources\main.blade.php, добавили Карусель, растянули ее по всей ширине окна (шириной Карусели, ее автоматическим изменением размера под разные типы экранов и прочими ее фишками за нас будет делать используемый нами CSS-фреймворк – Bootstrap 4). Карусель состоит из 3-х слайдов, в которые мы добавили три фотографии. Каждый является как бы презентацией трех информационных разделов нашего сайта, и сами разделы поместили ниже Карусели с полным текстом. В Карусели же у нас будет усеченный текст каждого раздела. И на эту страницу добавили созданные ранее и автоматически подставляемые «слои» панели навигации и футера.

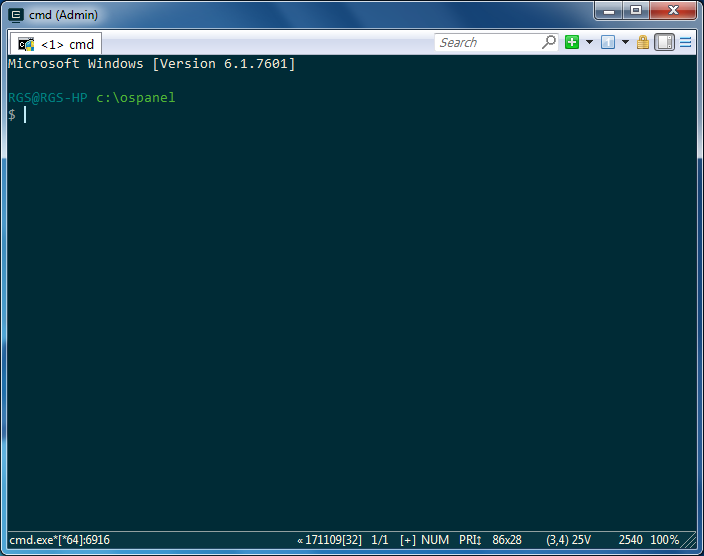
Создадим сразу таблицу базы данных content с помощью миграции. В таблице будет поле id – integer, unsigned, Автоматическое приращение. Что и станет первичным ключом. Второе поле: title – здесь будет название раздела и соответственно слайда – varchar (строковое значение) можем определить ему размер поля в том случае, если у нас есть представление о том, какой длины будет создавать названия администратор сайта. Но мы оставим значение по умолчанию. Третьим полем siteContent станет основное содержание раздела, которое может быть большим по размеру, поэтому выбирая из TEXT и MEDIUMTEXT (см. памятку по размерам строковых полей) я выбрал TEXT, т.к. слишком много текста в разделах будет довольно утомительно для посетителей. Ну и не будем отказываться от предлагаемых нам полей создания записи и ее модификации: created\_at, updated\_at. На размер нашей небольшой таблицы это сильно не повлияет, а нам может пригодиться.

|  |  |
| --- | --- |
| **Памятка по размерам строковых полей** | |
| VARCHAR | Может хранить не более 255 символов. |
| TINYTEXT | Может хранить не более 255 символов. |
| TEXT | Может хранить не более 65 535 символов. |
| MEDIUMTEXT | Может хранить не более 16 777 215 символов. |
| LONGTEXT | Может хранить не более 4 294 967 295 символов. |

Чтобы открыть консоль нам надо развернуть панель Инструменты OpenServer’а в трее и выбрать Дополнительно – Консоль



Открывается следующее окно:



Мы находимся в корне папки OpenServer, в данном случае в OSPanel – название папки зависит от версии пакета OpenServer и от места установки и настроек.

Переходим в наш проект командой:

cd domains\mary-dance.lc

Создаем модель командой

php artisan make:model Content -m

Эта команда с дополнительным параметром –m (migration) создаст одновременно модель и миграцию.

Исправим код в файле database\migrations\<метка\_даты>\_create\_conents\_table.php

<?php

use Illuminate\Support\Facades\Schema;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Database\Migrations\Migration;

class CreateContentsTable extends Migration

{

public function up()

{

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

$table->increments('id');

$table->string('title');

$table->text('pageContent');

$table->timestamps();

});

}

public function down()

{

Schema::dropIfExists('contents');

}

}

В методе (функция) up() задается создание таблицы contents и ее полей, которые были определены ранее, плюс «сервисные» поля timestamps.

Класс Schema предоставляет набор удобных статических методов для работы с таблицами, которые сами ничего не делают, а всего лишь вызывают методы других классов – это классы-«посредники», которые в Laravel называются фасадами – все они находятся в пространстве имен Illuminate\Support\Facades. Класс Blueprint представляет отдельную таблицу.

Для создания таблицы мы применили статический метод create фасада Schema:

Schema::create(<имя таблицы>, <функция, создающая таблицу>)

Функция, создающая таблицу, должна принимать единственный параметр – объект класса Blueprint, представляющий создаваемую таблицу.

|  |  |
| --- | --- |
| **Методы класса Blueprint:** | |
| **Команда** | **Описание** |
| $table->bigIncrements('id'); | Первичный последовательный ключ типа **BIGINT** |
| $table->bigInteger('votes'); | Поле **BIGINT** |
| $table->binary('data'); | Поле **BLOB** |
| $table->boolean('confirmed'); | Поле **BOOLEAN** |
| $table->char('name', 4); | Поле **CHAR** с указанной длиной |
| $table->date('created\_at'); | Поле **DATE** |
| $table->dateTime('created\_at'); | Поле **DATETIME** |
| $table->decimal('amount', 5, 2); | Поле **DECIMAL** с указанной размерностью и точностью |
| $table->double('column', 15, 8); | Поле **DOUBLE** с указанной точностью |
| $table->enum('choices', ['foo', 'bar']); | Поле **ENUM** |
| $table->float('amount'); | Поле **FLOAT** |
| $table->increments('id'); | Первичный последовательный ключ (*autoincrement*) |
| $table->integer('votes'); | Поле **INTEGER** |
| $table->json('options'); | Поле **JSON** |
| $table->jsonb('options'); | Поле **JSONB** |
| $table->longText('description'); | Поле **LONGTEXT** |
| $table->mediumInteger('numbers'); | Поле **MEDIUMINT** |
| $table->mediumText('description'); | Поле **MEDIUMTEXT** |
| $table->morphs('taggable'); | Добавляет **INTEGER** поле taggable\_id и **STRING** поле taggable\_type |
| $table->nullableTimestamps(); | То же что и timestamps(), но разрешены значения **NULL** |
| $table->smallInteger('votes'); | Поле **SMALLINT** |
| $table->tinyInteger('numbers'); | Поле **TINYINT** |
| $table->softDeletes(); | Добавляет поле deleted\_at для [мягкого удаления](https://laravel.ru/docs/v5/eloquent#%D0%BC%D1%8F%D0%B3%D0%BA%D0%BE%D0%B5) |
| $table->string('email'); | Поле **VARCHAR** |
| $table->string('name', 100); | Поле **VARCHAR** с указанной длиной |
| $table->text('description'); | Поле **TEXT** |
| $table->time('sunrise'); | Поле **TIME** |
| $table->timestamp('added\_on'); | Поле **TIMESTAMP** |
| $table->timestamps(); | Добавляет поля created\_at и updated\_at |
| $table->rememberToken(); | Добавляет поле remember\_token с типом **VARCHAR(100) NULL** |
| ->nullable() | Указывает, что поле может быть **NULL** |
| ->default($value) | Указывает значение по умолчанию для поля |
| ->unsigned() | Переводит **INTEGER** в беззнаковое число **UNSIGNED** |

Можно задавать у любого поля сразу несколько атрибутов последовательно, например:

$table->integer(‘number’)->default(1)->nullable()->first();

В методе down() указано действие для отката миграции: если таблица существует,- удалить ее.

Теперь запускаем миграцию, а модель пока не трогаем.

php artisan migrate

composer dump-autoload

Таблица создана, теперь наполним ее первичными данными, но не вручную. Создадим фабрику моделей для автоматического наполнения первичными данными нашей базы данных. Мы будем использовать очень популярную библиотеку faker, которая генерирует разнообразные фейковые (следуя из названия) данные. Данную библиотеку часто применяют для модульного и unit-тестирования при разработке. По умолчанию используется английский язык и форматы данных в выводе, однако существует возможность переопределить данные на другие языки вывода; мы же не будем ничего менять. Если вы используете полную версию установленного фреймворка Laravel, тогда не надо дополнительно устанавливать данную библиотеку, а можно сразу использовать весь ее функционал.

Если же у вас нет этой библиотеки, установите ее командой в Консоли:

composer require fzaninotto/faker

|  |  |
| --- | --- |
| Разделы с типами данных, генерируемых библиотекой: | |
| **Base** | простые методы для случайных букв, чисел, обработанных строк и регулярных выражений |
| **Lorem Ipsum Text** | случайный текст на латыни |
| **Person** | имена людей |
| **Address** | почтовые адреса |
| **Phone Number** | номера телефонов |
| **Company** | названия компаний |
| **Real Text** | фактический текст, написанный людьми вместо бессмысленных латинских строк |
| **Date and Time** | случайные даты и время |
| **Internet** | электронная почта, домены и т.д. |
| **User Agent** | строки браузера |
| **Payment** | строки и номера кредитных карт и SWIFT |
| **Color** | случайные цвета |
| **File** | расширения файлов, типы файлов и имена файлов |
| **Image** | URL-адреса изображений-заполнителей разных видов |
| **Uuid** | уникальные IDs |
| **Barcode** | различные типы штрих-кодов, например. ISBN13 |
| **Miscellaneous** | коды шифрования, коды стран и т.д. |
| **Biased** | случайные числа с уклоном |

Фабрика может генерировать данные без записи в БД – метод make() (что используется для тестирования), а может, чем мы и воспользуемся, сохранить эти фейковые данные в базу данных – метод create(). Для этого те, кто используют версию Laravel от 5.5 и далее могут выполнить команду

php artisan make:factory Content

и отредактировать созданный файл database\factories\Content.php и запускать каждую фабрику по-отдельности, но т.к. мы работаем с версией 5.4, то все фабрики создаем в имеющемся файле database\factories\ModelFactory.php. Откроем его и добавим к уже имеющемуся коду следующий фрагмент:

$factory->define(App\Content::class, function (Faker\Generator $faker) {

return [

'title' => $faker-> sentence,

'pageContent' => $faker-> realText . ' || ' . $faker-> realText,

];

});

В строке *'pageContent' => $faker-> realText . ' || ' . $faker-> realText,* я увеличиваю количество данных простой конкатенацией строк (соединением текста). Можно конечно использовать paragraphs, но он выдаст случайный текст на латыни. Т.е. вы можете использовать любой свой вариант, а я предпочел этот.

Создадим seeder класс для наполнения данными

php artisan make:seeder ContentTableSeeder

Отредактируем файл database\seeds\ContentTableSeeder.php

<?php

use Illuminate\Database\Seeder;

use Illuminate\Support\Facades\DB;

use App\Content;

class ContentTableSeeder extends Seeder

{

public function run()

{

DB::statement("SET foreign\_key\_checks=0");

Content::truncate();

DB::statement("SET foreign\_key\_checks=1");

factory(Content::class, 3)->create();

}

}

Снимаем защиту первичного ключа

SET foreign\_key\_checks=0

Очищаем таблицу если в ней уже были какие-то данные, используя нашу модель Content:

Content::truncate();

Устанавливаем защиту обратно

SET foreign\_key\_checks=1

И запускаем фабрику моделей на обращение к модели три раза - цифра 3 указанная вторым параметром, что создаст три записи в БД. Метод create() в отличие от метода make() не только заполнит данными результат запроса, но и внесет эти данные в нашу таблицу.

Приведем файл database\seeds\DatabaseSeeder.php к виду:

<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder

{

public function run()

{

$this->call('ContentTableSeeder');

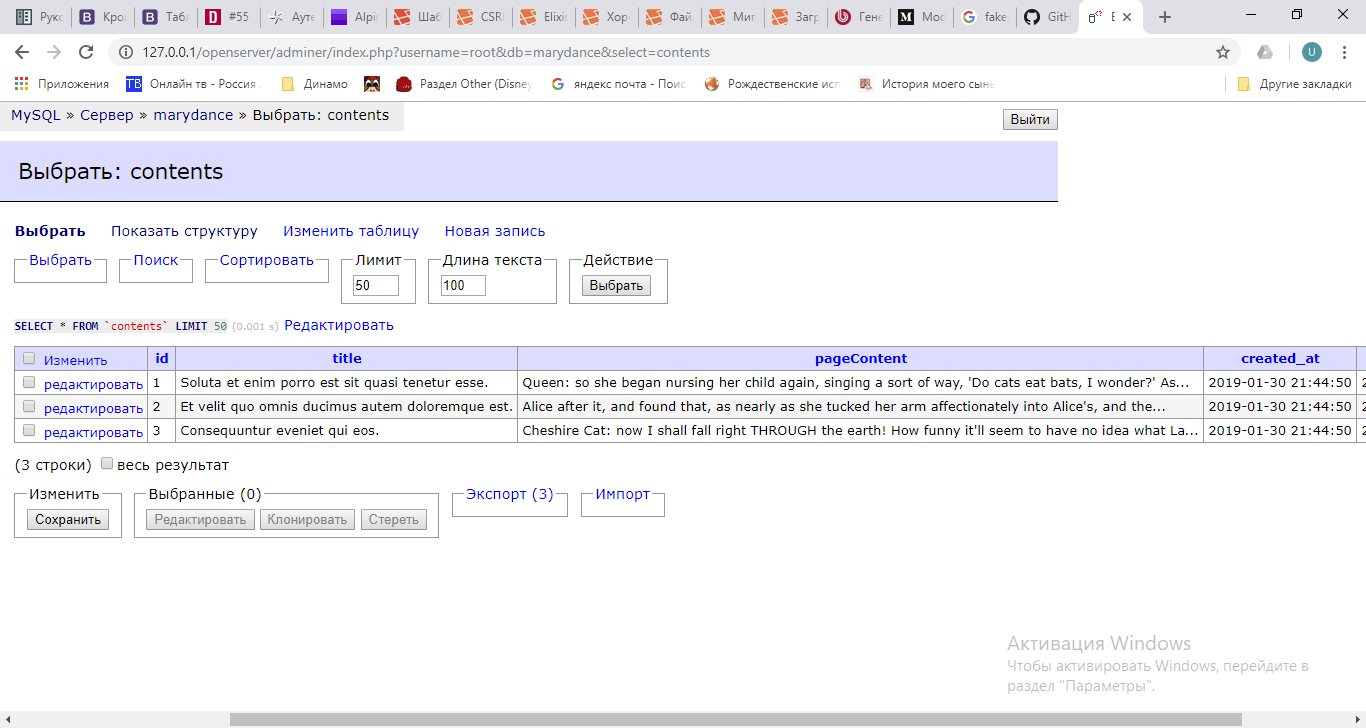
$this->command->info('operation completed successfully!');

}

}

Сохраним его и запустим в командной строке:

php artisan db:seed



На картинке показан результат всех наших действий.

Метод call() класса Seeder вызывает дополнительные классы «наполнителей», указанные аргументами метода; метод же info() просто выведет сообщение, указанное строковым аргументом метода, прямо в командную строку.

Таблица создана, модель готова, данные заполнены, - двигаемся дальше и переходим к созданию маршрута, контроллера и представления.

Открываем файл с маршрутами routes\web.php

Если у нас уже есть действие для маршрута ‘/’, например, Route::get('/', 'DogController@first');

то редактируем его, если нет, то создаем новый маршрут:

Route::get('/', 'MainController@index');

Т.е. запуск в строке адреса вида http://наш\_сайт будет обработан методом index() в контроллере MAinController.

Сохраняем файл с маршрутами и создаем контроллер командой

php artisan make:controller MainController

Если же у нас такой контроллер уже существует, то правим его код (котроллер у нас находится в файле app\Http\Controllers\MainController.php):

<?php

namespace App\Http\Controllers;

class MainController extends Controller

{

public function input() {

return view("main");

}

}

Такое простое действие контроллера будет вызывать файл представления, расположенный в resources\views\main.blade.php

В название файла часть blade показывает, что мы используем Blade-шаблоны для создания представлений.

Напомню изначальный код представления:

@extends('layouts.app')

@section("title", "Главная")

@section("main")

<div id="carouselMainControls" class="container-fluid carousel slide carousel-fade" data-ride="carousel">

<ol class="carousel-indicators">

<li data-target="#carouselExampleIndicators" data-slide-to="0" class="active"></li>

<li data-target="#carouselExampleIndicators" data-slide-to="1"></li>

<li data-target="#carouselExampleIndicators" data-slide-to="2"></li>

</ol>

<div class="carousel-inner">

<div class="carousel-item active">

<img class="d-block w-100" src="carousel/1.jpg" alt="First slide">

<div class="carousel-caption d-none d-md-block">

<h5>ПЕРВЫЙ СЛАЙД</h5>

<p>Всякое описание раздела</p>

</div>

</div>

<div class="carousel-item">

<img class="d-block w-100" src="carousel/2.jpg" alt="First slide">

<div class="carousel-caption d-none d-md-block">

<h5>ВТОРОЙ СЛАЙД</h5>

<p>Всякое описание раздела</p>

</div>

</div>

<div class="carousel-item">

<img class="d-block w-100" src="carousel/3.jpg" alt="First slide">

<div class="carousel-caption d-none d-md-block">

<h5>ТРЕТИЙ СЛАЙД</h5>

<p>Всякое описание раздела</p>

</div>

</div>

</div>

<a class="carousel-control-prev" href="#carouselMainControls" role="button" data-slide="prev">

<span class="carousel-control-prev-icon" aria-hidden="true"></span>

<span class="sr-only">Previous</span>

</a>

<a class="carousel-control-next" href="#carouselMainControls" role="button" data-slide="next">

<span class="carousel-control-next-icon" aria-hidden="true"></span>

<span class="sr-only">Next</span>

</a>

</div>

<div class="container marketing">

<div class="row">

<div class="col-lg-4">

<img class="rounded-circle" src="data:image/gif;base64,R0lGODlhAQABAIAAAHd3dwAAACH5BAAAAAAALAAAAAABAAEAAAICRAEAOw==" alt="Generic placeholder image" width="140" height="140">

<h2>Раздел 1</h2>

<p>Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста Очень много текста Очень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текста</p>

</div>

<div class="col-lg-4">

<img class="rounded-circle" src="data:image/gif;base64,R0lGODlhAQABAIAAAHd3dwAAACH5BAAAAAAALAAAAAABAAEAAAICRAEAOw==" alt="Generic placeholder image" width="140" height="140">

<h2>Раздел 2</h2>

<p>Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста Очень много текста Очень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текста.</p>

</div>

<div class="col-lg-4">

<img class="rounded-circle" src="data:image/gif;base64,R0lGODlhAQABAIAAAHd3dwAAACH5BAAAAAAALAAAAAABAAEAAAICRAEAOw==" alt="Generic placeholder image" width="140" height="140">

<h2>Раздел 3</h2>

<p>Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста Очень много текста Очень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текста.</p>

</div>

</div>

</div>

@endsection

Теперь нам надо вместо статического текста создать вставки из таблицы БД contents. Мы можем, как и в документации, использовать только контроллер (MainController) и создать в только что созданном методе index() запрос в БД и вернуть результат в вызов представления. Код в app\Http\Controllers\MainController.php:

<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\DB;

class MainController extends Controller

{

public function index() {

$cont = DB::table('contents')->select('title', 'pageContent')->get();

return view("main", ["content" => $cont]);

}

}

Используя метод table() фасада DB мы создаем SQL запрос, который с помощью метода get() вернет нам в переменную $cont «коллекцию» полученную запросом из таблицы contents. Добавив метод select() мы ограничиваем запрос только на получение данных из title и pageContent. Затем передаем полученные данные в представление, где мы их сможем вызвать, обратившись к переменной $content в представлении main.blade.php, которой мы передали данные из $cont этой строчкой “content” => $cont

Вот код измененного представления main.blade.php

@extends('layouts.app')

@section("title", "Главная")

@section("main")

<div id="carouselMainControls" class="container-fluid carousel slide carousel-fade" data-ride="carousel">

<ol class="carousel-indicators">

<li data-target="#carouselExampleIndicators" data-slide-to="0" class="active"></li>

<li data-target="#carouselExampleIndicators" data-slide-to="1"></li>

<li data-target="#carouselExampleIndicators" data-slide-to="2"></li>

</ol>

<div class="carousel-inner">

<?php $i=1; ?>

@foreach ($content as $cont)

<div class="carousel-item <?php if ($i==1) { echo 'active'; }?>">

<img class="d-block w-100" src="carousel/<?=$i;?>.jpg" alt="<?=$i;?> слайд">

<div class="carousel-caption d-none d-md-block">

<h5>{{ $cont->title }}</h5>

<p>

<?php echo (html\_entity\_decode(mb\_strimwidth($cont->pageContent, 0,200,"...",mb\_internal\_encoding()))); ?>

</p>

</div>

</div>

<?php $i++; ?>

@endforeach

</div>

<a class="carousel-control-prev" href="#carouselMainControls" role="button" data-slide="prev">

<span class="carousel-control-prev-icon" aria-hidden="true"></span>

<span class="sr-only">Previous</span>

</a>

<a class="carousel-control-next" href="#carouselMainControls" role="button" data-slide="next">

<span class="carousel-control-next-icon" aria-hidden="true"></span>

<span class="sr-only">Next</span>

</a>

</div>

<div class="container marketing">

<div class="row">

<div class="col-lg-4">

<img class="rounded-circle" src="data:image/gif;base64,R0lGODlhAQABAIAAAHd3dwAAACH5BAAAAAAALAAAAAABAAEAAAICRAEAOw==" alt="Generic placeholder image" width="140" height="140">

<h2>Раздел 1</h2>

<p>Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста Очень много текста Очень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текста

Очень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текста

Очень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текста</p>

</div>

<div class="col-lg-4">

<img class="rounded-circle" src="data:image/gif;base64,R0lGODlhAQABAIAAAHd3dwAAACH5BAAAAAAALAAAAAABAAEAAAICRAEAOw==" alt="Generic placeholder image" width="140" height="140">

<h2>Раздел 2</h2>

<p>Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста Очень много текста Очень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текста.</p>

</div>

<div class="col-lg-4">

<img class="rounded-circle" src="data:image/gif;base64,R0lGODlhAQABAIAAAHd3dwAAACH5BAAAAAAALAAAAAABAAEAAAICRAEAOw==" alt="Generic placeholder image" width="140" height="140">

<h2>Раздел 3</h2>

<p>Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста

Очень много текста Очень много текста Очень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текстаОчень много текста.</p>

</div>

</div>

</div>

@endsection

Мы внесли изменение в первую часть шаблона и некоторые моменты надо объяснить более подробно. Во-первых, создаем цикл для прохождения по всему нашему объекту $content, который был передан из модели с результатами запроса в БД. Мы используем синтаксис шаблонов Blade: @foreach ($content as $cont) что соответствуетзаписи на PHP <?php foreach ($content as $cont) { ?>

Так как изображениям у меня изначально были присвоены имена 1.jpg, 2.jpg, 3.jpg, то создаем переменную $i и присваиваем ей значение 1. И далее внутри цикла foreach делаем вызов изображения исходя из изменяющихся значений $i следующим образом:

<img class="d-block w-100" src="carousel/<?=$i;?>.jpg">

потом $i увеличиваем на единицу в каждом прохождении цикла.

Отсюда имеем, что сколько строк из таблицы нам «придет» с запросом, столько прохождений цикла и будет сделано, соответственно столько изображений будет вставлено, если мы не забудем переименовать изображения соответствующим образом. Далее, когда мы начнем создавать форму создания и редактирования разделов главной страницы, мы добавим возможность загрузки новых изображений и сохранение их в нашей БД.

Вызов заголовка раздела производим с помощью синтаксиса Blade вывода на экран {{ $cont->title }}Такой синтаксис подразумевает предварительное экранирование данных функциями php. Для вывода усеченной части основного содержания каждого раздела используем следующую конструкцию php, в которой предварительно декодируем все HTML-сущности функцией html\_entity\_decode(), т.к. в этих полях содержатся HTML-теги для форматирования:

<?php echo (html\_entity\_decode(mb\_strimwidth($cont->pageContent, 0, 200, "...", mb\_internal\_encoding()))); ?>

# Описание функции: mb\_strimwidth

(PHP 4 >= 4.0.6, PHP 5, PHP 7)

mb\_strimwidth — Получение строки, обрезанной до заданного размера

**Описание**

mb\_strimwidth ( string $str , int $start , int $width [, string $trimmarker = "" [, string$encoding = mb\_internal\_encoding() ]] ) : string

Обрезает строку ([string](http://php.net/manual/ru/language.types.string.php)) str до длины width символов.

**Список параметров**

**str** Исходная строка.

**start** Смещение от начала строки. Количество символов от начала строки (первый символ стоит в позиции 0). Если указано отрицательное число, то отсчет будет идти с конца строки.

**width** Размер вырезаемой части в символах. Отрицательные значения отсчитываются с конца.

**trimmarker** Строка, которая заместит конец обрезанной строки.

**encoding** Параметр encoding представляет собой символьную кодировку. Если он опущен, вместо него будет использовано значение внутренней кодировки.

[**mb\_internal\_encoding()**](http://php.net/manual/ru/function.mb-internal-encoding.php)- Установка/получение внутренней кодировки скрипта

Используем функцию для работы с многобайтными символами для корректной обработки кириллицы. Задаем, что из $cont раздела pageContent выводим текст только начиная с позиции 0 до позиции 200, а после добавляется многоточие (…). Данная функция не просто обрезает текст на позиции 200, а обрезает текст возможно даже меньше 200 символа, чтобы последнее слово отобразилось целиком. Это более приятно для восприятия, чем иметь в выводе обрезанный те… (это яркий пример того, что бы было, пойдя мы другим путем).

А теперь поучимся проводить рефакторинг собственного кода. Это необязательно делать с таким небольшим sql-запросом в разрезе получаемых данных, т.е. мы могли бы его оставить и в контроллере, но я рекомендую научиться выносить лишний код из контроллеров в различные сервисные классы, а в нашем случае, в модель Content.php

Википедия:

**Рефа́кторинг** (англ. *refactoring*), или **перепроектирование кода, переработка кода, равносильное преобразование алгоритмов** — процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы. В основе рефакторинга лежит последовательность небольших эквивалентных (то есть сохраняющих поведение) преобразований. Поскольку каждое преобразование маленькое, программисту легче проследить за его правильностью, и в то же время вся последовательность может привести к существенной перестройке программы и улучшению её согласованности и чёткости.

Прежде чем начать редактировать модель, надо понять для себя принципы использования моделей:

- один объект класса модели (т.е. одна модель) представляет одну соответствующую таблицу в БД;

- модель предоставляет инструменты как для получения значений полей, так и для занесения в них новых значений – сохранение данных в таблице БД;

- модель позволяет добраться до связанных записей другой таблицы, если между соответствующими моделями установлена связь.

Говоря про модели надо сделать небольшое отступление и рассказать несколько основных моментов. Laravel предъявляет два основных требования к классам моделей:

1. Модель должна иметь имя, совпадающее с именем сущности (таблицы в БД). У нас имя таблицы – contents, имя модели – Content. Т.е. таблицу мы именуем во множественном числе, а модель соответственно в единственном.
2. Модель должна быть потомком класса Model или его потомка. Этот класс предоставляет все методы (функции) по взаимодействию с базой данных, преобразованию типов данных и прочее.

Модели Laravel следуют особым соглашениям по умолчанию:

* таблица базы данных, с которой взаимодействует модель, имеет служебные поля со следующими именами: ключевое поле – id, поле для хранения даты и времени создания записи – created\_at, поле для хранения даты и времени последнего изменения записи – updated\_at.
* файлы с моделями находятся непосредственно в папке app папки проекта.

Эти соглашения по умолчанию, и они не являются обязательными к исполнению, но желательно следовать им, так как это сильно упростит вам разработку.

Отредактируем ранее созданную модель app\Content.php и приведем ее к такому виду:

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Support\Facades\DB;

class Content extends Model

{

public static function mainPage() {

return DB::table(“contents”)->select(“title”, “pageContent”)->get();

}

}

Контроллер MainController.php приведем к такому виду:

<?php

namespace App\Http\Controllers;

use App\Content;

class MainController extends Controller

{

public function index() {

return view("main", ["content" => Content::mainPage()]);

}

}

Вот так мы сделали наш контроллер «тоньше». Теперь мы из контроллера вызываем статический метод нашей модели, в котором формируется запрос и получается ответ из БД.

Вообще же следование SOLID архитектуре, концепции MVC (модель – представление – контроллер) – все это является благодатной почвой для большого количества споров по поводу: что не надо держать в контроллерах, куда и в какие именно сервисные классы, запросы и т.д. надо выносить код. Иногда большое распыление логики по разным классам и файлам приводит к еще большей путанице при поддержке и доработке такого проекта. Совет один: следуйте своему мнению и не перегибайте палку.

Давайте доработаем вторую часть нашего представления, а потом займемся таким вопросом: а не предоставить ли администратору сайта возможность изменять изображения, используемые в Карусели, и что нам для этого надо.

@extends('layouts.app')

@section("title", "Главная")

@section("main")

<div id="carouselMainControls" class="container-fluid carousel slide carousel-fade" data-ride="carousel">

<ol class="carousel-indicators">

<li data-target="#carouselExampleIndicators" data-slide-to="0" class="active"></li>

<li data-target="#carouselExampleIndicators" data-slide-to="1"></li>

<li data-target="#carouselExampleIndicators" data-slide-to="2"></li>

</ol>

<div class="carousel-inner">

<?php $i=1; ?>

@foreach ($content as $cont)

<div class="carousel-item <?php if ($i==1) { echo 'active'; }?>">

<img class="d-block w-100" src="carousel/<?=$i;?>.jpg" alt="<?=$i;?> слайд">

<div class="carousel-caption d-none d-md-block">

<h5>{{ $cont->title }}</h5>

<p>

<?php echo (html\_entity\_decode(mb\_strimwidth($cont->pageContent, 0, 200, "...", mb\_internal\_encoding()))); ?>

</p>

</div>

</div>

<?php $i++; ?>

@endforeach

</div>

<a class="carousel-control-prev" href="#carouselMainControls" role="button" data-slide="prev">

<span class="carousel-control-prev-icon" aria-hidden="true"></span>

<span class="sr-only">Previous</span>

</a>

<a class="carousel-control-next" href="#carouselMainControls" role="button" data-slide="next">

<span class="carousel-control-next-icon" aria-hidden="true"></span>

<span class="sr-only">Next</span>

</a>

</div>

<div class="container marketing">

<div class="row">

@foreach ($content as $cont)

<div class="col-lg-4">

<img class="rounded-circle" src="data:image/gif;base64,R0lGODlhAQABAIAAAHd3dwAAACH5BAAAAAAALAAAAAABAAEAAAICRAEAOw==" alt="Generic placeholder image" width="140" height="140">

<h2>{{ $cont->title }}</h2>

<p>{!! html\_entity\_decode($cont->pageContent) !!}</p>

</div>

@endforeach

</div>

</div>

@endsection

Код стал намного меньше и более удобочитаемый.

Запускаем нашу главную страницу, и любуемся проделанной работой.

Так как в следующей главе мы будем работать с изменениями записей в таблице БД с помощью модели Content.php, нам необходимо внести в нее кое-какие изменения – обязательно добавить поле protected $fillable = [] чтобы не столкнуться с защитой от проблем безопасности массового присвоения. Уязвимость массового заполнения проявляется, когда пользователь передаёт с помощью запроса неподходящий HTTP-параметр, и вы не ожидаете, что этот параметр изменит столбец в вашей БД. Например, злоумышленник может послать в HTTP-запросе параметр is\_admin, который затем передаётся в метод create() вашей модели, позволяя пользователю повысить свои привилегии до администратора.

Поэтому, для начала надо определить, для каких атрибутов разрешить массовое назначение. Это делается с помощью свойства модели $fillable. Для остальных массовое присвоение не разрешено.

Наш Content.php:

<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Support\Facades\DB;

class Content extends Model

{

protected $fillable = ["id", "title", "pageContent"];

protected $primaryKey = "id";

class Content extends Model

{

public static function mainPage() {

return DB::table(“contents”)->select(“title”, “pageContent”)->get();

}

}