УДК 004.4

Разработка модуля страниц для панели администратора на Laravel

Якимов Антон Сергеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Баженов Руслан Иванович
Приамурский государственный университет имени Шолом-Алейхема
Кандидат педагогических наук, доцент, зав. кафедрой информационных систем, математики и методик обучения

Аннотация
В данной статье описывается пошаговая инструкция по разработке модуля страниц для панели администратора, основанной на фреймворке Laravel.
Ключевые слова: laravel, панель администратора, модуль, страница, контроллер, миграция, модель

Develop module of pages for admin panel on Laravel

Yakimov Anton Sergeevich
Sholom-Aleichem Priamursky State University
Student

Bazhenov Ruslan Ivanovich
Sholom-Aleichem Priamursky State University
Candidate of pedagogical sciences, associate professor, Head of the Department of Information systems, Mathematics and teaching methods

Abstract
This article describes step by step instructions to develop pages for the administrator panel module, based on the framework Laravel.
Keywords: laravel, admin panel, module, pages, controller, migration, model

Почти все у всех сайтов есть и должна быть административная панель, которая позволяет с легкостью взаимодействовать с содержимым сайта, а также редактировать и настраивать элементы с помощью подключаемых модулей. Например, модуль страниц. Из административной панели мы можем с легкостью добавлять нужные нам страницы для сайта, а также редактировать и удалять их. В данной статье мы как раз рассмотрим разработку модуля страниц для административной панели, написанной на фреймворке Laravel.

С данной ситуацией уже сталкивались некоторые исследователи. Например, Е.А. Козлов разработал сайт с поддержкой административной

панели с просмотрами информации о пользователях [1]. О.М. Литвиненко и А.А. Новикова внедрили в своей проект движок WordPress, который удобен из-за наличия многофункционального панели администратора [2]. Н. Н. Лукиных и А. Ю. Сироткин разработали портал Вуза на основе движка MODx, который имеет простую и удобную панель администратора [3]. А также и другие исследователи [4-8].

Для того чтобы реализовать модуль страниц, нужно сначала реализовать маршрутизацию для отображения разрабатываемого модуля со стороны административной панели. Чтобы создать маршрутизацию, необходимо открыть файл */route/web.php*, который находится внутри фреймворка Laravel, и дописать следующий код, размещенный внутри группы административной панели.

```
/**
 * Страницы
 */
Route::group(['prefix' => 'pages'], function()
{
    Route::get('/')
        ->uses('PagesController@index')
        ->name('pages');

    Route::get('/create')
        ->uses('PagesController@create')
        ->name('pages.create');

    Route::post('/store')
        ->uses('PagesController@store')
        ->name('pages.store');

    Route::get('/edit/{id}')
        ->uses('PagesController@edit')
        ->name('pages.edit');

    Route::post('/update/{id}')
        ->uses('PagesController@update')
        ->name('pages.update');

    Route::get('/delete/{id}')
        ->uses('PagesController@delete')
        ->name('pages.delete');
});
```

Рисунок 1 – Маршрутизация страниц

В данном коде содержатся шесть маршрутов. Каждый маршрут принимает свой URL путь и при заходе по данной ссылке, со стороны адмін-панели, он вызывает контроллер *PagesController* с нужными методами, которые будут выполнять ту или иную роль. Создадим контроллер, для этого в папке */app/Http/Controllers* создаем файл *PagesController.php* и впишем следующий код.

```php
<?php
    namespace App\Http\Controllers;

    use Illuminate\Http\Request;
    use App\Http\Requests;
    use App\Pages;

    class PagesController extends Controller
    {

    }
?>
```

Рисунок 2 – Контроллер PagesController

Базовый каркас контроллера готов. Осталось наполнить данного контроллера методами *index*, *create*, *store*, *edit*, *update* и *delete*. Но помимо контроллера нужно еще добавить миграцию и модель страницы. Миграция – это система контроля версии для базы данных, а точнее, в ней мы можем реализовать структуру таблиц базы данных, которая нам необходима. Модель – это система, которая является связывающим звеном между контроллером и базой данных. Модель позволяет запрашивать необходимые данные с базы данных и отдавать контроллеру нужные записи, и наоборот, принимать данные от контроллера и записывать в базу данных. Чтобы реализовать миграцию, нам нужно воспользоваться командной строкой с помощью встроенного пакета Artisan внутри Laravel. Для этого в командной строке нужно вписать следующую команду:

php artisan make:migration create_pages_table

Чтобы использовать данную команду, убедитесь, что вы находитесь в корневой папке Laravel. После выполнения данной команды у вас в папке */database/migrations* создастся миграционный файл страницы. Открываем ее и вписываем следующее содержимое кода.

```php
<?php
    use Illuminate\Support\Facades\Schema;
    use Illuminate\Database\Schema\Blueprint;
    use Illuminate\Database\Migrations\Migration;

    class CreatePagesTable extends Migration
    {
        public function up()
        {
            Schema::create('pages', function (Blueprint $table) {
                $table->increments('id');
                $table->string('title')->nulleable();
                $table->string('slug')->nulleable();
                $table->text('text')->nulleable();
                $table->timestamps();
            });
        }

        public function down()
        {
            Schema::drop('pages');
        }
    }
?>
```

Рисунок 3 – Миграционный файл страницы

Затем применяем миграцию с помощью командной строки, вводя следующую команду:

php artisan migration

Данная команда записывает в базу данных нашу таблицу, которую мы описали в файле миграции. Теперь нужно создать модель. Для этого в папке */app* создаем файл *Pages.php* и впишем в него следующий код.

```php
<?php
    namespace App;

    use Illuminate\Database\Eloquent\Model;

    class Pages extends Model
    {
        protected $table = 'pages';
    }
?>
```

Рисунок 4 – Модель страницы

Теперь вернемся обратно к контроллеру *PagesController* и создадим все необходимые нам методы. Создаем метод *index*:

```php
public function index()
{
    $pages = Pages::all();
    return view('pages.index', ['pages' => $pages]);
}
```

Рисунок 5 – Метод index

Данный метод позволяет выводить список всех страниц, которые есть в таблице базы данных. Далее создаем метод *create*:

```php
public function create()
{
    return view('pages.create');
}
```

Рисунок 6 – Метод create

Данный метод позволяет выводить шаблон для создания страниц. Следующим методом будет *store*:

```php
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required|max:255',
        'slug'  => 'required|alpha_dash|unique:pages|max:255',
        'text'  => 'required'
    ]);

    $pages = [
        'title' => $request->title,
        'slug'  => $request->slug,
        'text'  => $request->text
    ];

    Pages::create($pages);

    return redirect(route('pages'))->with('success', 'Страница успешно добавлена!');
}
```

Рисунок 7 – Метод store

Данный метод позволяет осуществлять создание страниц, занося их в базу данных. Дальше будет метод *edit*:

```php
public function edit($id)
{
    $data['page'] = Pages::findOrFail($id);
    return view('pages.edit', $data);
}
```

Рисунок 8 – Метод edit

Данный метод позволяет выводить конкретную страницу из базы данных, для редактирования данной страницы. Следом у нас будет метод *update*.

```php
public function update(Request $request, $id)
{
    $page = Pages::findOrFail($id);

    $this->validate($request, [
        'title' => 'required|max:255',
        'slug'  => 'required|alpha_dash|unique:pages,slug,'.$page->id.'|max:255',
        'text'  => 'required'
    ]);

    $update = [
        'title' => $request->title,
        'slug'  => $request->slug,
        'text'  => $request->text
    ];

    $page->update($update);
    return redirect(route('pages.edit', $page->id))->with('success', 'Страница успешно обновлена!');
}
```

Рисунок 9 – Метод update

Данный метод позволяет обновлять страницу в базу данных. И последним методом у нас будет *delete*.

```php
public function delete($id)
{
    Pages::find($id)->delete();
    return redirect(route('pages'))->with('success', 'Страница успешно удалена!');
}
```

Рисунок 10 – Метод delete

Данный метод позволяет удалять страницу из базы данных.

Теперь осталось привязать данный разработанный модуль к шаблону в панели администратора и пользоваться ими. Для этого заходим в папку

/resources/views, добавляем в ней еще одну папку – *pages*, и внутри нее создаем три файла: *index.blade.php*, *create.blade.php* и *edit.blade.php*.

Вводим следующий код внутри файла *index.blade.php*.

```
@extends('app')
@section('content')
    @if(session('success'))
        <div class="alert alert-success">
            <p>{{ session('success') }}</p>
        </div>
    @endif
    <table class="table table-hover">
        <thead>
            <tr>
            <th>ID</th>
            <th>Заголовок</th>
            <th>Автор</th>
            <th>Статус</th>
            </tr>
        </thead>
        <tbody>
            @foreach($pages as $page)
                <tr>
                    <td>{{ $page->id }}</td>
                    <td>{{ $page->title }}</td>
                    <td>
                        <a href="{{ route('pages.edit', $page->id) }}" class="btn btn-xs btn-info">Редактировать</a>
                        <a href="{{ route('pages.delete', $page->id) }}" class="btn btn-xs btn-danger">Удалить</a>
                    </td>
                </tr>
            @endforeach
        </tbody>
    </table>
@stop
```

Рисунок 11 – Код шаблона *index.blade.php*

Теперь вносим следующий код внутри файла *create.blade.php*.

```
@extends('app')
@section('content')
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h3 class="panel-title">Добавление страницы</h3>
        </div>
        <div class="panel-body">
            <form action="{{ route('pages.store') }}" enctype="multipart/form-data" method="post">
                {{ csrf_field() }}
                <div class="form-group">
                    <label for="title">Название</label>
                    <input type="text" name="title" class="form-control" id="title" value="{{ old('title') }}">
                </div>
                <div class="form-group">
                    <label for="slug">URL</label>
                    <input type="text" name="slug" class="form-control" id="slug" value="{{ old('slug') }}">
                </div>
                <div class="form-group">
                    <label for="text">Описание</label>
                    <textarea type="text" name="text" class="form-control">{{ old('text') }}</textarea>
                </div>
                <div class="form-group">
                    <button type="submit" class="btn btn-primary">Добавить</button>
                </div>
            </form>
        </div>
    </div>
@stop
```

Рисунок 12 – Код шаблона *create*.blade.php

И последнее, это файл *edit.blade.php*. Вносим следующее содержимое кода.

```
@extends('app')
@section('content')
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h3 class="panel-title">Редактирование страницы</h3>
        </div>
        <div class="panel-body">
            <form action="{{ route('pages.update', $page->id) }}" enctype="multipart/form-data" method="post">
                {{ csrf_field() }}
                <div class="form-group">
                    <label for="title">Название</label>
                    <input type="text" name="title" class="form-control" id="title" value="{{ old('title') ? old('title') : $page->title }}">
                </div>
                <div class="form-group">
                    <label for="slug">URL</label>
                    <input type="text" name="slug" class="form-control" id="slug" value="{{ old('slug') ? old('slug') : $page->slug }}">
                </div>
                <div class="form-group">
                    <label for="text">Описание</label>
                    <textarea type="text" name="text" class="form-control">{{ old('text') ? old('text') : $page->text }}</textarea>
                </div>
                <div class="form-group">
                    <button type="submit" class="btn btn-primary">Сохранить</button>
                </div>
            </form>
        </div>
    </div>
@stop
```

Рисунок 13 – Код шаблона *edit.blade.php*

На этом разработка модуля заканчивается. Можем проверить их в рабочем состоянии, зайдя по ссылке */pages* в панели администратора, увидим следующую картину.

Рисунок 14 – Список страниц

Для данной демонстрации, в базу данных уже заранее были созданы пять страниц. Как видим, модуль успешно отобразил все страницы из базы данных. Попробуем создать новую страницу. Для этого нужно перейти по адресу */pages/create*, и нам откроется следующая страница.

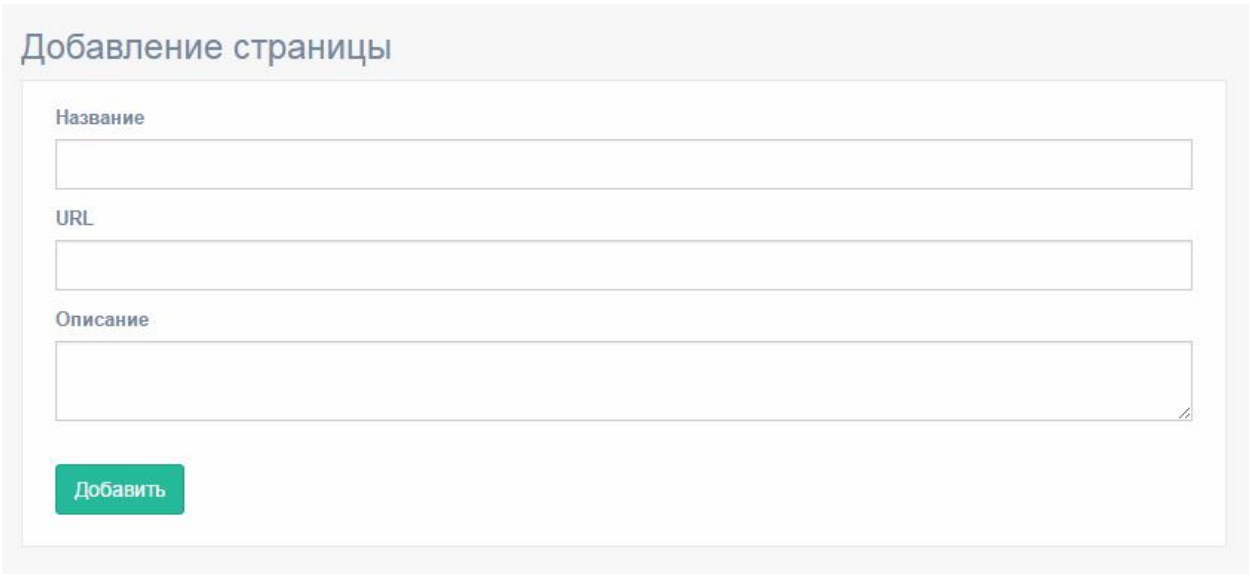

Рисунок 15 – Добавление страницы

Попробуем создать страницу, заполнив все поля. Саму страницу, для примера, мы назовем *«Тестовая страница»*. После того, как заполнили все данные, нужно нажать на кнопку «*Добавить»*, и после этого нас перебросит в список страниц с уведомлением о том, что страница успешно добавлена:

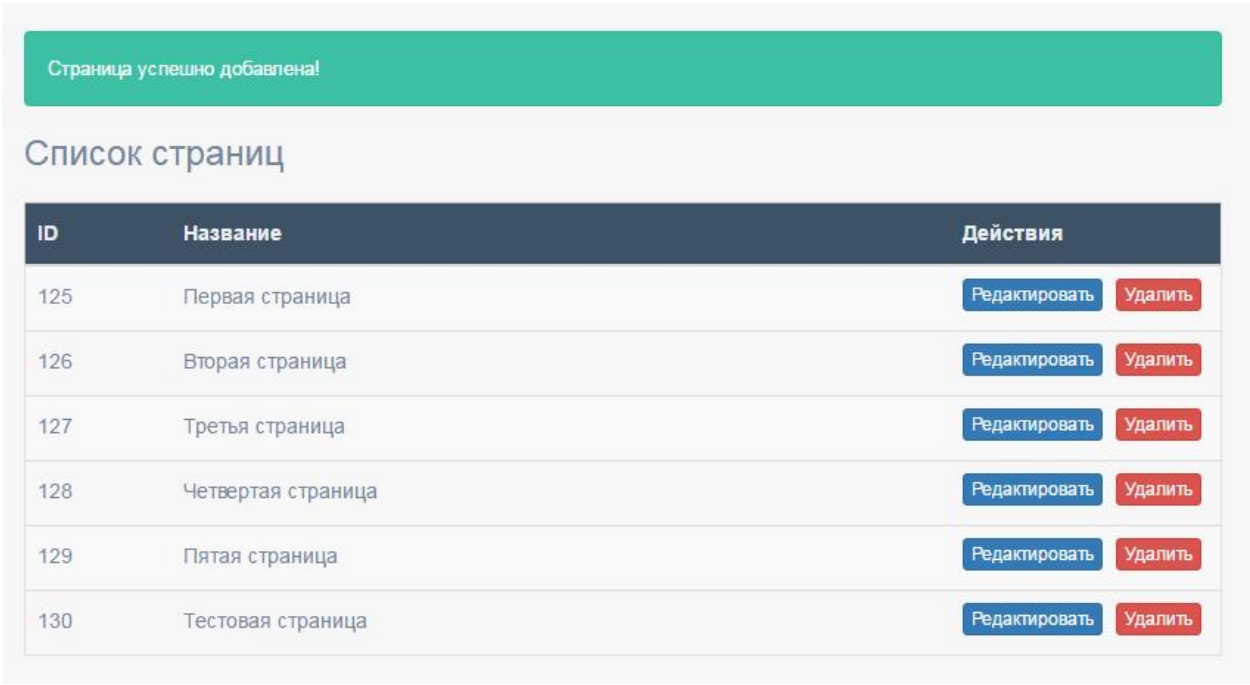

Рисунок 16 – Добавленная страница

Как видим, страница *«Тестовая страница»* была успешно добавлена. Также можно нажать на кнопку *«Редактировать»* у добавленной страницы, и попасть в следующую страницу.

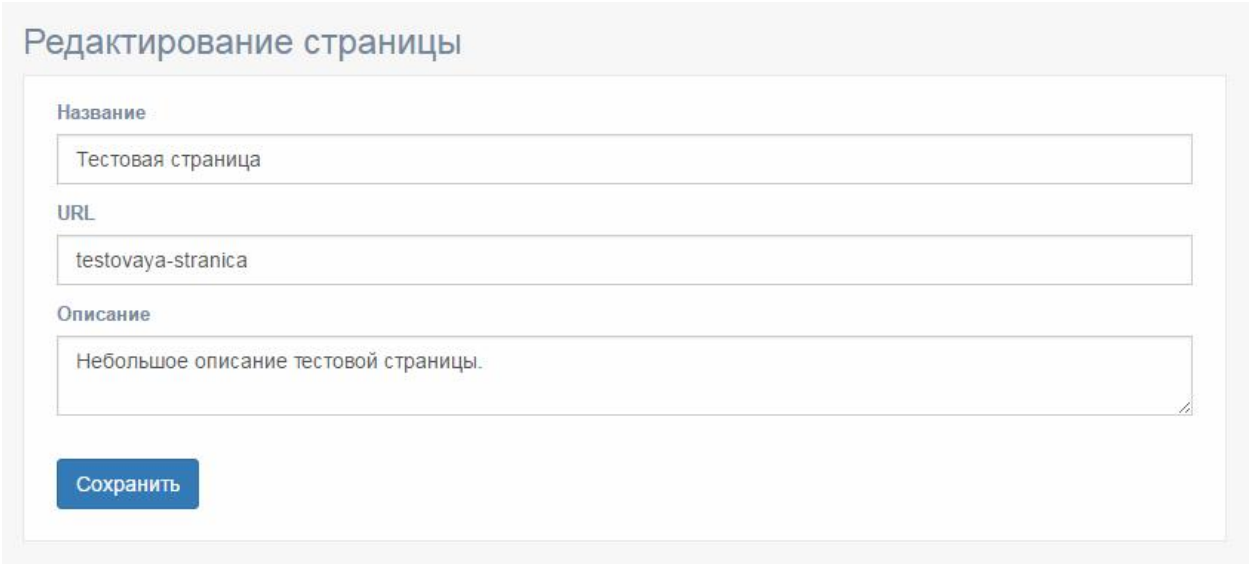

Рисунок 17 – Редактирование страницы

Для проверки, можем изменить название страницы. В нашем случае, заменим старое название на *«Измененная страница»*, и сохраним. После этого нас также перебросит обратно в список страниц, с уведомлением о том, что страница успешно обновлена.

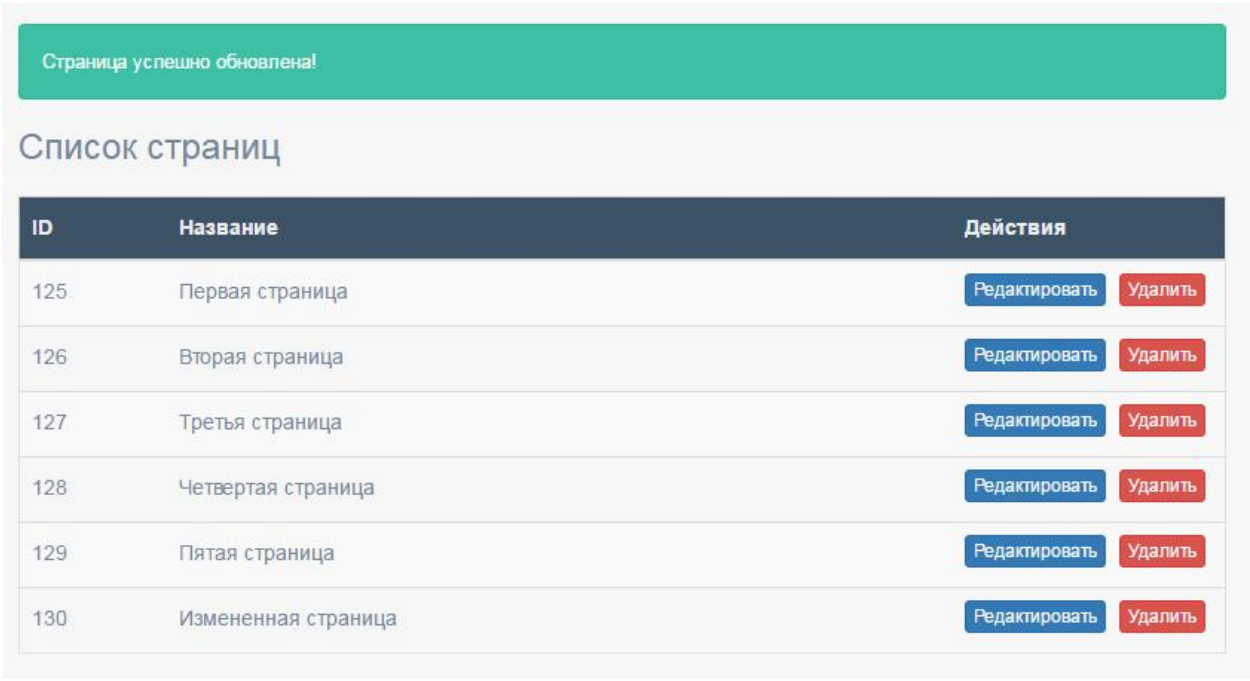

Рисунок 18 – Измененная страница

И напоследок, мы можем проверить удаление страницы. Для этого необходимо нажать на кнопку «Удалить», например, у страницы под названием «Первая страница». После нажатия на кнопку, нас уведомит о том, что страница была успешно удалена.

Рисунок 19 – Удаленная страница

На этом знакомство с модулем заканчивается. В данной статье мы описали поэтапно все шаги для создания модуля страниц. Научились создавать контроллеры. Ознакомились с миграциями, моделью и маршрутизацией. А также продемонстрировали данного модуля в рабочем процессе.

Библиографический список

1. Козлов Е. А. Описание программирования и установки автоматизированной системы контроля знаний при дистанционном обучении // Вестник ВУиТ. 2009. №13 С.120-141.
2. Литвиненко О.М., Новикова А.А. Анализ и формализация биотехнических систем дистанционного обучения // Биомедицинская инженерия и электроника. 2014. №2 (6) С.132-135.
3. Лукиных Н. Н., Сироткин А. Ю. Разработка современного портала вуза на базе CMF MODx // Гаудеамус. 2012. №20 С.204.
4. Королева Н. Л., Печерица В. И. Разработка web-сайта ООО «ИТ-Меридиан» средствами CMS Joomla // Гаудеамус. 2012. №20 С.202-204.
5. Назаров Е., Горбунова К., Галузина С., Рекунов К., Сажин В. Б., Бесков В. С. Разработка интернет-сайта кафедры общей химической технологии // Успехи в химии и химической технологии. 2007. №2 (70) С.107-118.
6. Крючин О.В. Разработка модуля управления скидками для интернет-магазинов // Вестник Тамбовского университета. Серия: Естественные и технические науки. 2014. №2 С.613-616.
7. Борсук Н.А. Разработка модуля администрирования системы on-line бронирования // Инновационная наука. 2016. №5-2 (17) С.25-27.
8. Белов Д. Е., Шалин А. Ф. Разработка модуля авторизации пользователей и разграничения прав доступа к данным // Сборник научных трудов ГНУ СНИИЖК. 2013. №6 (1) С.325-338.