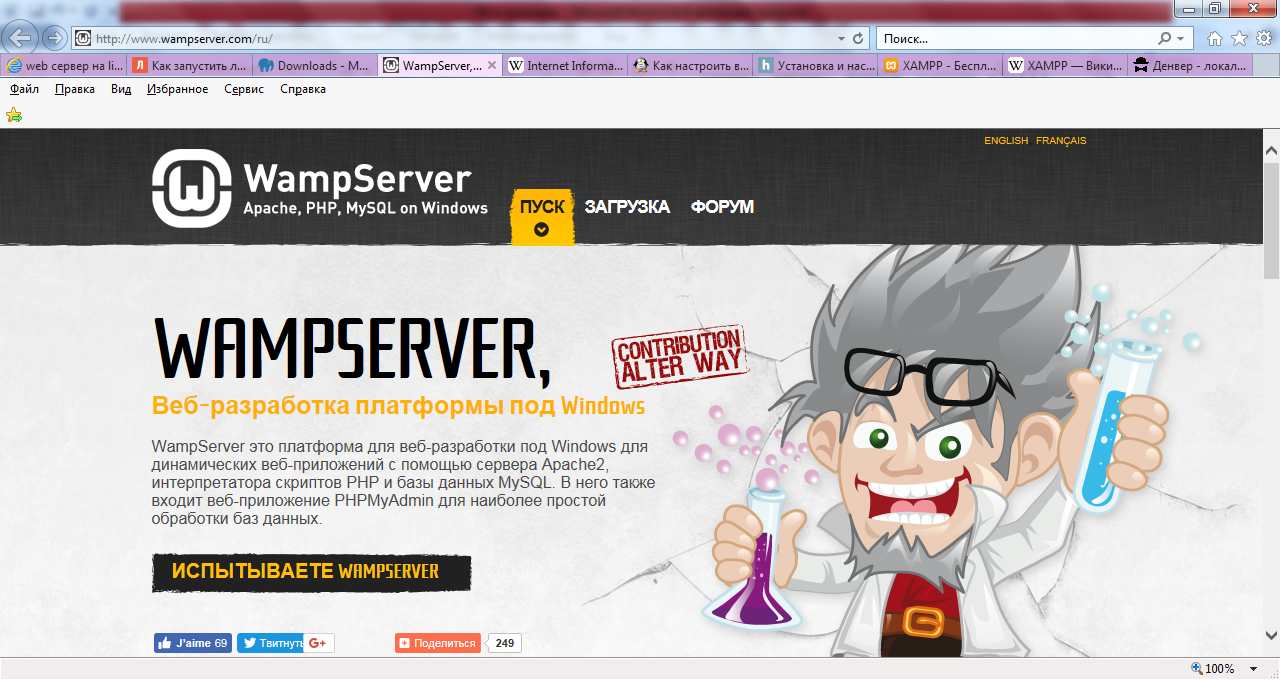
1. Обзор локальных web-серверов.
2. Установка и настройка Open Server
3. Установка и настройка IDE – среды разработки приложений. Бесплатная IDE: NetBeans IDE
4. Установка Laravel с помощью менеджера зависимостей COMPOSER
5. Подключение системы контроля версий GIT
6. Настройка проекта в режим Разработка, добавление CSS фреймворка, настройка OpenServer под новый проект.
7. Структура проекта и отрисовка представлений.
8. Установка и настройка web-сервера.

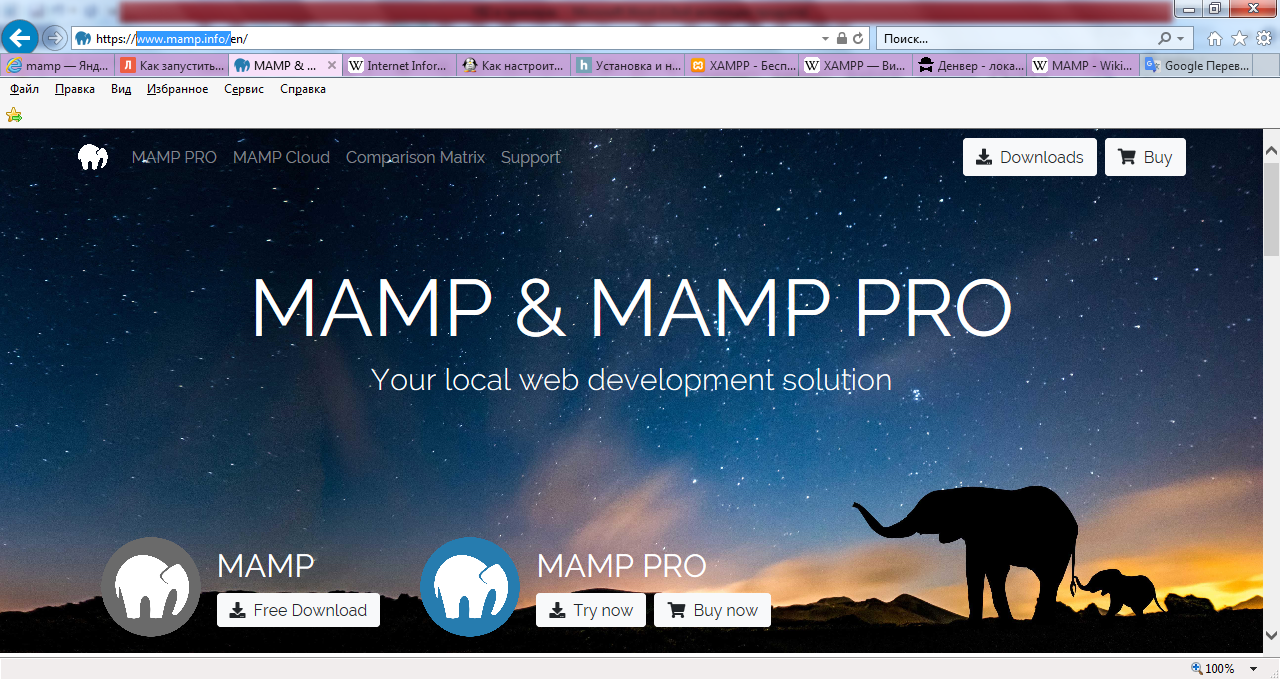
В последнее время выбор web-серверов для локальной web-разработки огромен. В данной книге будет использоваться OpenServer. Подобные решения «все в одном» становятся все более популярными для локальной разработки, так как сводят процесс инсталляции и настройки сервера к минимуму. Его установку и основные настройки я опишу вкарце далее.

Небольшой обзор имеющихся решений для локального web-сервера.

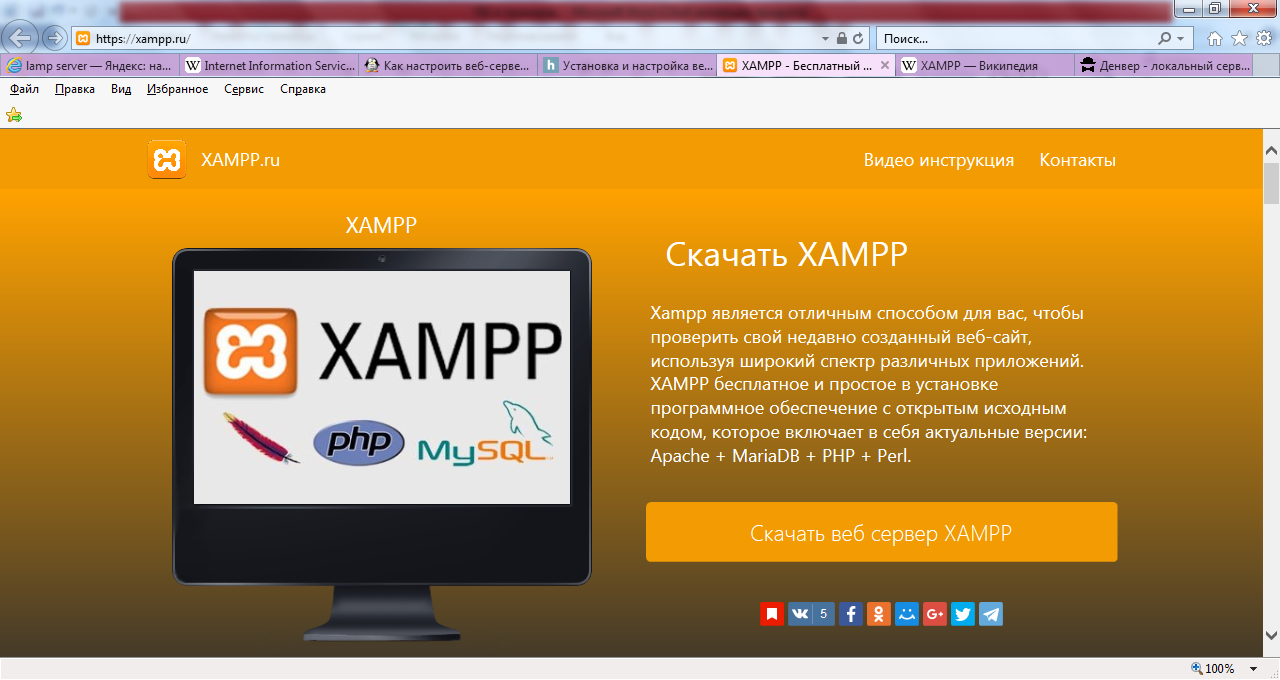
**WAMPServer** – (аббревиатура Windows, Apache, MySQL, PHP) бесплатный продукт для Windows систем из разряда «все в одном». WampServer это платформа для веб-разработки под Windows для динамических веб-приложений с помощью сервера Apache2, интерпретатора скриптов PHP и базы данных MySQL. В него также входит веб-приложение PHPMyAdmin для наиболее простой обработки баз данных. Необходимо только скачать, установить и произвести первичные настройки, которые можно найти на сайте разработчика (www.wampserver.com).



**MAMP** - это пакет решений, состоящий из свободного и открытого исходного кода и запатентованного коммерческого программного обеспечения, используемого для запуска динамических веб-сайтов на компьютерах Mac OS. Из-за популярности MAMP теперь также доступен для Windows 7 и выше. Существует бесплатная версия MAMP и платная версия MAMP PRO (www.mamp.info).



**XAMPP** - кроссплатформенная сборка веб-сервера, содержащая Apache, MySQL, интерпретатор скриптов PHP, язык программирования Perl и большое количество дополнительных библиотек, позволяющих запустить полноценный веб-сервер (xampp.ru).



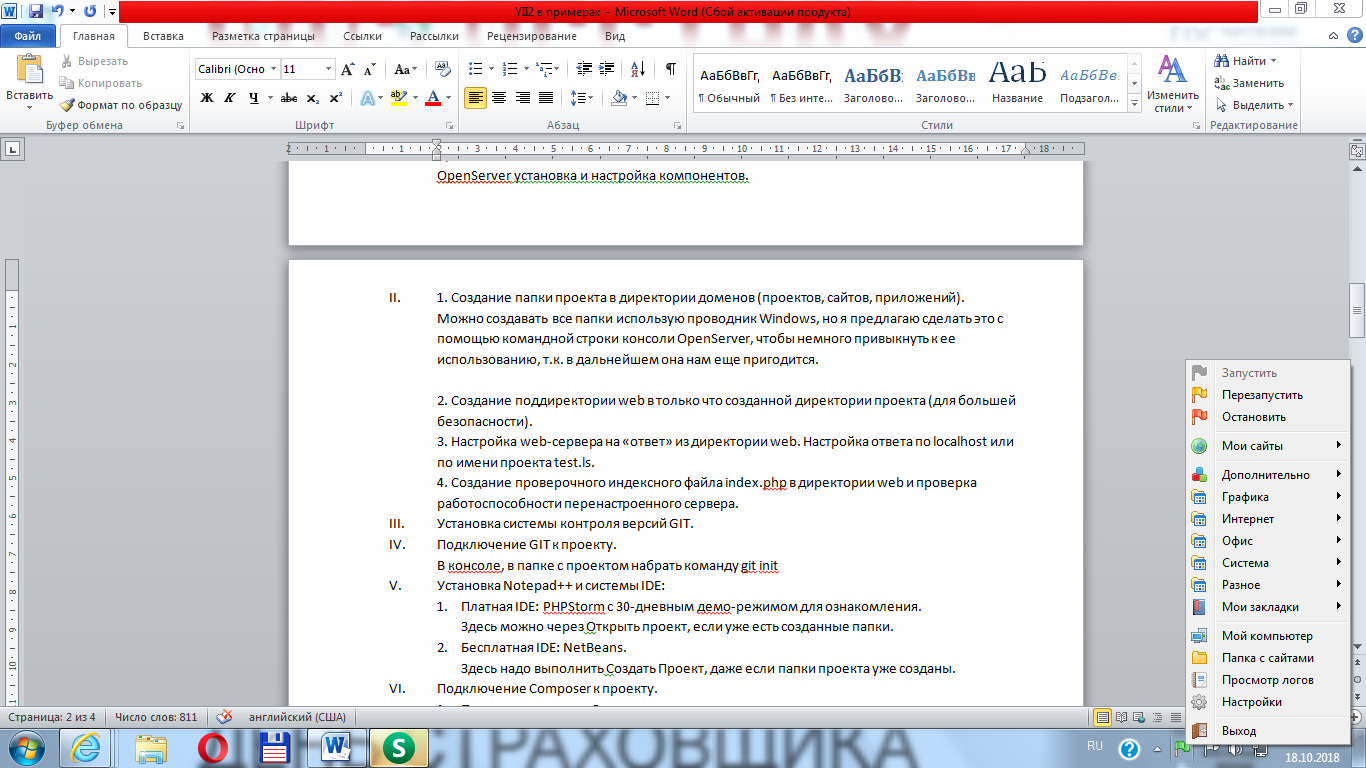
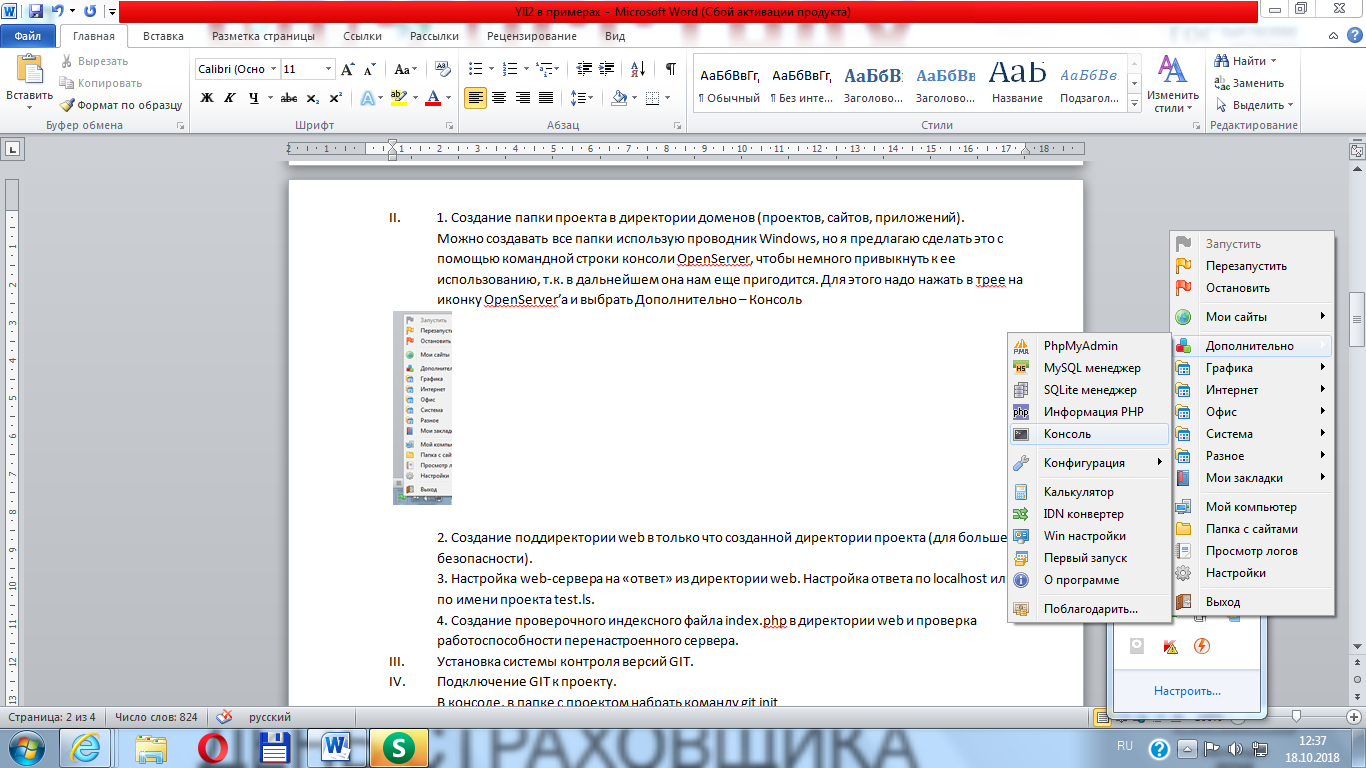
Можно выбрать немного более сложные решения, в том плане, что либо компоненты этих решений необходимо доустанавливать по-отдельности, либо больше настроек, но появляется более точно настроить локальный сервер под себя и под свои проекты. А именно Denwer, web-сервер в IIS представленной Microsoft во всех версиях Windows и т.д.

Установка web-сервера на системы семейства Unix в данной книге не рассматривается. Такую информацию с подробными последовательностями действий можно довольно легко найти в Интернете.

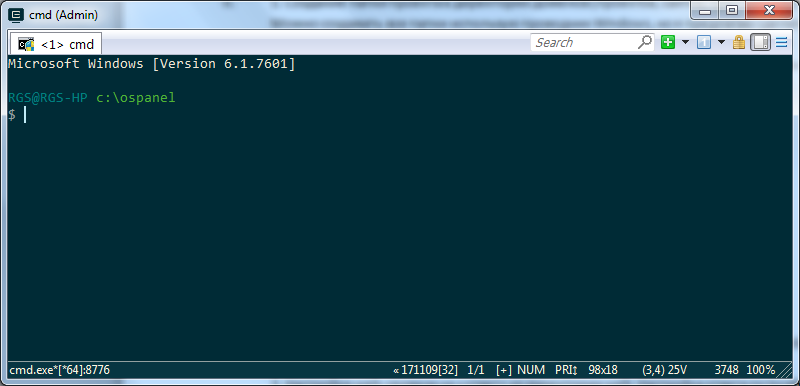
**OpenServer** установка и настройка компонентов.

1. 1. Создание папки проекта в директории доменов (проектов, сайтов, приложений).

Можно создавать все папки использую проводник Windows, но я предлагаю сделать это с помощью командной строки консоли OpenServer, чтобы немного привыкнуть к ее использованию, т.к. в дальнейшем она нам еще пригодится. Для этого надо нажать в трее на иконку OpenServer’а и выбрать Дополнительно – Консоль

Вот то, что у нас откроется



Мы находимся в папке с развернутым web-сервером (это ospanel или openserver – в зависимости от версии и установки OpenServer)

Набираем в dir, нажимаем Enter и видим список папок в папке установки.

Нас интересует папка domains в которую OpenServer по умолчанию предлагает разместить проекты. Открываем ее командой cd domains и создаем в ней новую папку с нашим будущим проектом. Предлагаю назвать ее first.lc (название проекту можете дать любое, т.к. вы всегда можете его сменить. Префикс .lc использую по старой привычке для обозначения локального расположения).

mkdir first.lc

В нашей директории domains появилась поддиректория first.lc, в которой и будет располагаться наш проект.

3. Настройка web-сервера на «ответ» из директории web. Настройка ответа по localhost или по имени проекта test.ls.

4. Создание проверочного индексного файла index.php в директории web и проверка работоспособности перенастроенного сервера.

1. Установка системы контроля версий GIT.
2. Подключение GIT к проекту.

В консоле, в папке с проектом набрать команду git init

1. Установка Notepad++ и системы IDE:
2. Платная IDE: PHPStorm с 30-дневным демо-режимом для ознакомления.

Здесь можно через Открыть проект, если уже есть созданные папки.

1. Бесплатная IDE: NetBeans.

Здесь надо выполнить Создать Проект, даже если папки проекта уже созданы.

1. Подключение Composer к проекту.
2. Проверка установки Composer из папки проекта

$ composer

Небольшая вставка об инструменте Composer

1. Если Composer не установлен:

Установка Composer на различных системах.

1. Создание файла .gitignore

Правой клавишей в IDE:

В PHPStorm: New – File

В NetBeans: он его создает сам

PHPStorm: после создания, файл открывается автоматически для внесения изменений. Галочка ставится на те файлы, которые не надо добавлять в GIT.

Первоначально только .gitignore

Чтобы закоммитить надо:

В PHPStorm: правой клавишей по паке проекта – CommitDirectory.

В NetBeans: правой клавишей по папке пректа – GIT – Фиксировать…

Обязательно добавить везде фиксации в Git после каждого серьезного именения.

Разработка через тестирование.

Начинаем с аутентификации.

Итак, чтобы добавить в свой проект файлы, необходимые для организации Laravel аутентификации пользователей (формы логина, регистрации, восстановления пароля), а также правила маршрутизации для произведения соответствующих действий), первым делом, открываем консоль сервера, переходим в каталог Laravel проекта и выполняем следующую artisan команду:

|  |  |
| --- | --- |
| 1 | php artisan make:auth |

Если всё прошло успешно, то вы увидите в консоли сообщение **Authentication scaffolding generated successfully.**

Для этого в Laravel по умолчанию есть базовые миграции для создания таблицы пользователей (по умолчанию называется **users**) и таблицы для хранения попыток сброса паролей (**password\_resets**), которые расположены в каталоге **корень\_сайта\database\migrations** и носят следующие названия:

* **2014\_10\_12\_000000\_create\_users\_table.php**
* **2014\_10\_12\_100000\_create\_password\_resets\_table.php**

Я на своём сайте изначально планировал организовать Laravel аутентификацию для логина администратора в админке, т.е. предполагалось, что пользователь будет один, и никакой сброс пароля и регистрация новых юзеров на сайте нужны не будут. Поэтому я смело удаляю миграцию для создания таблицы password\_resets.

Ещё один нюанс. Я хочу, чтобы имя моего пользователя хранилось в поле не name, как по умолчанию, а login, которое, как и email, должно быть уникальным. Поэтому я внёс необходимые правки в файл миграции, который стал выглядеть так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | <?php    use Illuminate\Support\Facades\Schema;  use Illuminate\Database\Schema\Blueprint;  use Illuminate\Database\Migrations\Migration;    class CreateUsersTable extends Migration  {      /\*\*       \* Run the migrations.       \*       \* @return void       \*/      public function up()      {          Schema::create('users', function (Blueprint $table) {              $table->increments('id');              $table->string('name')->unique();              $table->string('email')->unique();              $table->string('password');              $table->rememberToken();              $table->timestamps();          });      }        /\*\*       \* Reverse the migrations.       \*       \* @return void       \*/      public function down()      {          Schema::drop('users');      }  } |

К числу дополнительных приготовлений также относится подготовка Laravel seeder для добавления в БД данных администратора. С этой целью я откорректировал базовый файл по пути **корень\_сайта\database\seeds\DatabaseSeeder.php** следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <?php    use Illuminate\Database\Seeder;    class DatabaseSeeder extends Seeder  {      /\*\*       \* Run the database seeds.       \*       \* @return void       \*/      public function run()      {          DB::table('users')->insert([              'login' => 'admin',              'name' => 'admin@gmail.com',              'password' => bcrypt('admin'),              'created\_at' => date('Y-m-d H:i:s'),              'updated\_at' => date('Y-m-d H:i:s'),          ]);      }  } |

Думаю, не стоит говорить, что не нужно вслепую копировать и выполнять мой код  Создавайте администратора с реальными данными, т.к. в дальнейшем вам придётся с ними работать, поэтому поменяйте email, логин и пароль на что-то более вменяемое, чем у меня в примере.

В этом же файле вы, кстати, можете поменять и название всей таблицы в целом, использовав вместо стандартного users то, что посчитаете нужным.

После этого всё, что осталось для того, чтобы создать таблицу в базе данных и наполнить её данными, — это запустить следующую Laravel artisan команду:

|  |  |
| --- | --- |
| 1 | php artisan migrate --seed |

По поводу структуры таблицы БД с данными пользователей, сами разработчики Laravel в [официальной документации](https://laravel.com/docs/5.6/authentication) дают рекомендации использовать для хранения пароля строковый тип поля длиной не менее 60 символов и добавить в таблицу поле **remember\_token** длиной в 100 символов (тип, естественно, строковый) и со значением **NULL** по умолчанию.

Последнее необходимо будет для длительного хранения сессии пользователя и восстановлении сеанса при повторном входе на сайт.

**Laravel аутентификация: настройка**

Итак, после того, как мы добавили необходимые для Laravel auth файлы в наш проект и создали таблицу для хранения пользователей, осталось только всё это дело связать, т.е. указать фреймворку, в какой именно таблице следует искать Laravel auth user данные при его аутентификации на сайте.

Чтобы это сделать, нужно заглянуть в соответствующий Laravel config файл (или конфиг, по-русски), в котором описаны все необходимые настройки, необходимые при настройке authentication — **корень\_сайта\config\auth.php**. Данный файл, как и контроллеры, доступен в Laravel приложении также по умолчанию.

Он состоит из секций, описывающих **guards (гвард)** и **providers (провайдер)** для произведения Laravel user auth. Первые определяют действия, которые происходят в процессе аутентификации, а вторые — где хранятся данные пользователей.

По умолчанию в Laravel auth provider один — **users**, который выглядит так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | ...    'providers' => [      'users' => [          'driver' => 'eloquent',          'model' => App\User::class,      ],        // 'users' => [      //     'driver' => 'database',      //     'table' => 'users',      // ],  ],    ... |

Как видите, здесь представлены параметры, с помощью которых можно указать в качестве источника данных как имя таблицы БД, так и модель. По умолчанию указана дефолтная модель **User**, расположенная в файле **корень\_сайта\app\User.php** и использующая в качестве источника таблицу БД **users**.

Если же таблица с данными пользователей у вас называется по-другому, то вам необходимо в данной секции конфига прописать свою модель или таблицу БД, выбрав необходимый драйвер — **eloquent** для использования модели или **database**для таблиц БД.

Также Laravel позволяет создавать свои кастомные провайдеры. Как это сделать, очень подробно описано [здесь](https://laravel.com/docs/5.6/authentication#adding-custom-user-providers).

Секция настроек для конфигурирования guards выглядит же следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | 'guards' => [      'web' => [          'driver' => 'session',          'provider' => 'users',      ],        'api' => [          'driver' => 'token',          'provider' => 'users',      ],  ], |

Laravel auth guards, доступные по умолчанию, имеют названия созвучные их областям применения. Web подходит для большинства стандартных сайтов, т.к. для аутентификации пользователей используются данные, хранящиеся в сессии на сервере (физически они могут быть оформлены как в виде файлов, так и таблицы в БД, но Laravel session будет посвящен отдельный разговор).

Api же отлично подходит для запросов, отправляемых через API на сервер. Аутентификация в этом случае происходит с помощью специальных токенов — уникальных наборов символов, которые генерируются при успешном входе в систему под определёнными учётками.

Драйвера для guards, задающие способ аутентификации, бывают двух типов: session и token, принцип работы которых был описан выше. Также для Laravel auth guards можно указывать providers, чтобы разграничивать хранилища данных для каждого гварда. В качестве провайдера указываются структуры, описанные в секции providers этого же конфига auth.php.

Как и в случае с провайдерами, Laravel позволяет создавать свои кастомные гварды. Пример реализации описан [здесь](https://laravel.com/docs/5.6/authentication#adding-custom-guards).

И ещё одной секцией рассматриваемого нами конфига Laravel аутентификации является блок настроек сброса пароля пользователей, который выглядит так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | 'passwords' => [      'users' => [          'provider' => 'users',          'table' => 'password\_resets',          'expire' => 60,      ],  ], |

Для каждого блока данной секции нужно указывать провайдер, чтобы фреймворк понимал, с какими данными оперировать при сбросе пароля, и таблицу, в которую будут записываться попытки сброса.

Параметр expire позволяет указать время существования токена сброса. Другими словами, определяет время, в течении которого пользователь может сбросить свой пароль после оставления им заявки, при которой reset token и генерируется.

Настройки Laravel аутентификации и регистрации, действующие по умолчанию, вынесены в отдельный блок в самом верху файла:

|  |  |
| --- | --- |
| 1  2  3  4 | 'defaults' => [      'guard' => 'web',      'passwords' => 'users',  ], |

Здесь указан дефолтный гвард и блок настроек для сброса пароля.

Как вы могли заметить, все настройки взаимосвязаны: providers используются в guards и passwords, guards — в настройках по умолчанию.

Также ещё одной особенностью настроек является возможность создания нескольких провайдеров и, соответственно, гвардов и секций сброса пароля в зависимости от типов пользователей, имеющихся в вашем проекте.

**Кастомизация Laravel auth методов**

После запуска Laravel artisan команды **make:auth** ваш проект был расширен следующими файлами (спасибо Git, что он есть и позволяет определять данные изменения), которые были скопированы из недр Laravel framework:

* **app\Http\Controllers\HomeController.php**
* **resources\views\auth\login.blade.php**
* **resources\views\auth\passwords\email.blade.php**
* **resources\views\auth\passwords\reset.blade.php**
* **resources\views\auth\register.blade.php**
* **resources\views\home.blade.php**
* **resources\views\layouts\app.blade.php**

А также был обновлён файл маршрутизации (со списком Laravel routes, они же — правила маршрутизации или маршруты) **routes\web.php**, в который добавились следующие строчки:

|  |  |
| --- | --- |
| 1  2  3 | Auth::routes();    Route::get('/home', 'HomeController@index')->name('home'); |

Первая строка представляет собой список стандартных правил маршрутизации Laravel, связанных с аутентификацией. Для общего развития приведу их полный список, чтобы вы знали, какие они есть в принципе:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | /\*\*   \* Роуты аутентификации...   \*/    //отображение формы аутентификации  Route::get('login', 'Auth\LoginController@showLoginForm')->name('login');  //POST запрос аутентификации на сайте  Route::post('login', 'Auth\LoginController@login');  //POST запрос на выход из системы (логаут)  Route::post('logout', 'Auth\LoginController@logout')->name('logout');    /\*\*   \* Маршруты регистрации...   \*/    //страница с формой Laravel регистрации пользователей  Route::get('register', 'Auth\RegisterController@showRegistrationForm')->name('register');  //POST запрос регистрации на сайте  Route::post('register', 'Auth\RegisterController@register');    /\*\*   \* URL для сброса пароля...   \*/    //POST запрос для отправки email письма пользователю для сброса пароля  Route::post('password/email', 'Auth\ForgotPasswordController@sendResetLinkEmail')->name('password.email');  //ссылка для сброса пароля (можно размещать в письме)  Route::get('password/reset', 'Auth\ForgotPasswordController@showLinkRequestForm')->name('password.request');  //страница с формой для сброса пароля  Route::get('password/reset/{token}', 'Auth\ResetPasswordController@showResetForm')->name('password.reset');  //POST запрос для сброса старого и установки нового пароля  Route::post('password/reset', 'Auth\ResetPasswordController@reset'); |

Возвращаясь к стандартным Laravel auth routes, хотелось бы сказать, что, несмотря на их кажущуюся полноценность, они всё же составлены немного нелепо, т.к. в качестве обработчиков у них указаны несуществующие методы.

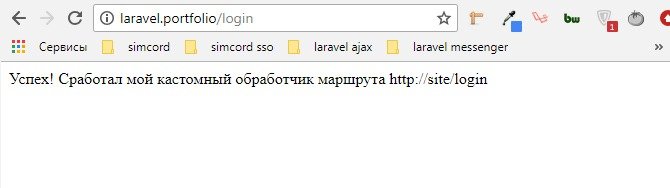
Например, судя по роутам, в файле **app\Http\Controllers\Auth\LoginController.php** должно быть три метода:

* **showLoginForm();**
* **login();**
* **logout();**

Однако, если вы загляните в данный файл, то не найдёте в нём ни одного из вышеперечисленных. Но, если вы добавите следующую функцию в самый конец контроллера, то при вводе соответствующего URL в браузере увидите, что обработчик сработал.

|  |  |
| --- | --- |
| 1  2  3  4 | public function showLoginForm()  {      return 'Успех! Сработал мой кастомный обработчик маршрута http://site.com/login';  } |

И, собственно говоря, результат в браузере:



Если же всё оставить на своих местах, т.е. не писать кастомный обработчик, то при вводе URL в формате **http://site.com/login** на экране браузера вы увидите стандартную форму входа, файл которой был добавлены в приложение при запуске Artisan команды **make:auth**.

Итак, возвращаясь к обработчикам стандартных Laravel auth routes, давайте разберёмся, что это за магия приключилась с ними, что в качестве обработчиков по умолчанию им указаны несуществующие методы, вместо которых используются совершенно другие.

Для примера возьмём метод, который мы только что переопределяли — **http://site.com/login**.

При отсутствующем кастомном обработчике по умолчанию при переходе на данный URL в браузере выполняется метод **showLoginForm()** из файла по пути **директория\_сайта\vendor\laravel\framework\src\Illuminate\Foundation\Auth\AuthenticatesUsers.php**, который выглядит следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | /\*\*   \* Show the application's login form.   \*   \* @return \Illuminate\Http\Response   \*/  public function showLoginForm()  {     return view('auth.login');  } |

Как видите, он как раз и ответственен за вывод на экран страницы с auth формой, код которой и хранится в уже упоминаемом нами файле **resources\views\auth\login.blade.php**.

Почему же при создании кастомного обработчика в контроллере LoginController.php он начинает вызываться вместо того, что указан выше? Всё просто. Приведённый выше метод является оригинальным методом core-класса фреймворка Laravel.

А **LoginController** является классом, наследующим оригинальный. Таким образом он позволяет создавать врапперы (от «wrapper» — оболочка, обёртка) стандартных методов с пользовательским кодом внутри.

Нужно сказать, что данный механизм, который встречается в Laravel сплошь и рядом, очень удобный и практичный, т.к. позволяет легко и быстро переопределять базовые конструкции, не шарясь и, не дай Бог, не изменяя код движка, изменения в котором будут потеряны при первом же обновлении версии Laravel.

Если заглянуть в код файла **Auth\LoginController.php**, то в самом верху можно увидеть следующую строку, благодаря которой становится возможным использование кастомных врапперов:

|  |  |
| --- | --- |
| 1 | use Illuminate\Foundation\Auth\AuthenticatesUsers; |

Такая же строка присутствует во всех стандартных контроллерах из каталога **app\Http\Controllers\Auth**, которые являются наследниками базовых классов Laravel регистрации и аутентификации пользователей, благодаря чему в них можно описывать wrappers для всех public методов стандартных классов (найти их можно поиском по папке vendor, указывая имена методов из числа тех, что были приведены в начале статьи).

И по этому поводу держите ещё один лайфхак, связанный с кастомизацией Laravel auth методов

Если вы захотите переопределить базовые методы либо продублировать их, то достаточно создать контроллер и использовать в нём trait **AuthenticatesUsers** тем же способом, что был описан выше. Этот приём может быть полезен, если вам захочется сделать отдельные формы и механизмы Laravel аутентификации и регистрации для разных типов пользователей, которые будут храниться в разных таблицах БД.

Данный функционал актуален для marketplaces, социальных сетей и других [типов сайтов](http://cccp-blog.com/sozdanie-saytov/klassifikaciya-sajtov), обеспечивающих взаимодействие пользователей с разными атрибутами.

**Laravel аутентификация: практикум**

Поскольку следующим этапом создания моего тестового сайта будет установка админки, в которой будет работать только администратор, то я буду отключать регистрацию в Laravel приложении.

А также мне не нужен будет стандартный для Laravel сброс пароля.

Кроме того, мне не нужна тестовая страница, поэтому вместо неё я сделаю страницу-заглушку, которая будет доступна по адресу **http://site.com/admin** и на которую будет происходить редирект после удачной аутентификации. Также я хочу сделать редирект на страницу логина Laravel при выходе пользователя из системы (logout).

Ну, и ещё, как я уже говорил, я хочу сделать аутентификацию не по значению поля email, а по полю login, для чего мне понадобятся определённые настройки.

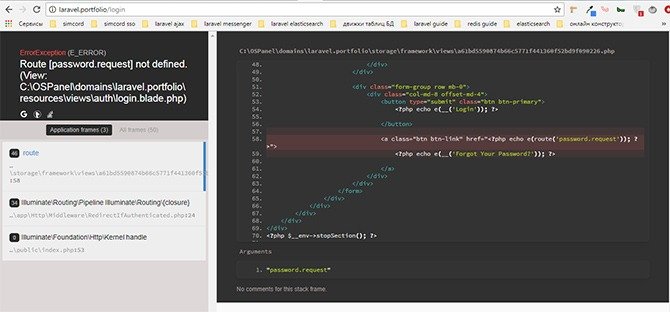
План действий составлен — за дело

Первым делом я удаляю Laravel Blade шаблоны страниц сброса пароля и регистрации, которые находятся в каталоге **корень\_сайта\resources\views\auth\passwords** и файле **корень\_сайта\resources\views\auth\register.blade.php**соответственно.

Теперь нужно удалить их роуты. Для этого я убираю в файле **routes\web.php** строку с обозначением стандартных Laravel роутов маршрутизации **Auth::routes();** и заменяю её на одинарные правила маршрутизации для отображения формы логина, самого запроса на логин в системе и удаления сессии пользователя при logout:

|  |  |
| --- | --- |
| 1  2  3  4 | //Кастомные правила маршрутизации  Route::get('login', 'Auth\LoginController@showLoginForm')->name('login');  Route::post('login', 'Auth\LoginController@login');  Route::post('logout', 'Auth\LoginController@logout'); |

После этого при переходе на страницу логина я получил следующую ошибку:



Причина её заключается в том, что роут и шаблон формы регистрации я удалил, а вызов роута в коде приложения остался, поэтому для устранения ошибки необходимо удалить из файла **корень\_сайта\resources\views\auth\login.blade.php**следующую строку:

|  |  |
| --- | --- |
| 1  2  3 | <a class="btn btn-link" href="{{ route('password.request') }}">      {{ \_\_('Forgot Your Password?') }}  </a> |

Так же удалил HomeController и resources\views\home.blade.php

После этого подобная ошибка возникла снова, теперь уже из-за вызова в шаблоне Laravel auth форм**корень\_сайта\resources\views\layouts\app.blade.php** роута регистрации, ссылку на который я тоже удалил:

|  |  |
| --- | --- |
| 1 | <li><a class="nav-link" href="{{ route('register') }}">{{ \_\_('Register') }}</a></li> |

*Вывод: если вы используете вызов именованных роутов в коде Laravel сайтов, то при удалении их из файла маршрутизации во избежание описанных фейлов не забывайте удалять ссылки на них в коде, особенно из blade-шаблонов, которые на этапе компиляции в обычный HTML код будут сыпать errors*

После произведения вышеописанных действий при обращении к url **http://site.com/register** сайт стал отдавать 404 ошибку HTTP запроса, чего я и добивался.

Также попутно я удалил все контроллеры из каталога **корень\_сайта\app\Http\Controllers\Auth**, оставив только **LoginController.php**, который по умолчанию содержит все необходимые для моего сайта роуты.

Следующим действием с Laravel auth стало изменения данных для аутентификации на сайте. Вместо **email** я решил логинить пользователя по его **login**.

Для этого я в контроллере **корень\_сайта\app\Http\Controllers\Auth\LoginController.php** добавил следующую функцию, которая переопределяет стандартный метод username(), возвращающий название поля таблицы БД с пользователями, по которому нужно проверять юзеров на сайте:

|  |  |
| --- | --- |
| 1  2  3  4 | public function username()  {      return 'name';  } |

Также вносим коррективы в файл формы входа, чтобы настроить поле email HTML формы для ввода в него логина. Для этого в login.blade.php меняем данный код:

@extends('layouts.app')

@section('main')

<div class="container">

<div class="row">

<div class="col-md-8 col-md-offset-2">

<div class="panel panel-default">

<div class="panel-heading">Вход администратора на сайт</div>

<div class="panel-body">

<form class="form-horizontal" method="POST" action="{{ route('login') }}">

{{ csrf\_field() }}

<div class="form-group{{ $errors->has('name') ? ' has-error' : '' }}">

<label for="name" class="col-md-4 control-label">Login</label>

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

<input id="name" type="text" class="form-control" name="name" value="{{ old('name') }}" required autofocus>

@if ($errors->has('name'))

<span class="help-block">

<strong>{{ $errors->first('name') }}</strong>

</span>

@endif

</div>

</div>

<div class="form-group{{ $errors->has('password') ? ' has-error' : '' }}">

<label for="password" class="col-md-4 control-label">Password</label>

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

<input id="password" type="password" class="form-control" name="password" required>

@if ($errors->has('password'))

<span class="help-block">

<strong>{{ $errors->first('password') }}</strong>

</span>

@endif

</div>

</div>

<div class="form-group">

<div class="col-md-6 col-md-offset-4">

<div class="checkbox">

<label>

<input type="checkbox" name="remember" {{ old('remember') ? 'checked' : '' }}> Remember Me

</label>

</div>

</div>

</div>

<div class="form-group">

<div class="col-md-8 col-md-offset-4">

<button type="submit" class="btn btn-primary">

Войти

</button>

</div>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

@endsection

Затем изменил редирект после успешной авторизации, т.к. поумолчанию после успешной авторизации пользователь перенаправляется на роут /home – т.е. на страницу resources\views\home.blade.php, которая уже мною удалена и удален сам маршрут:

В файле app\Http\Controllers\Auth\LoginController.php изменена следующая часть:

class LoginController extends Controller {

use AuthenticatesUsers;

protected $redirectTo = '/';

на:

class LoginController extends Controller {

use AuthenticatesUsers;

protected function redirectTo(){

return Session::get('backUrl') ? Session::get('backUrl') : $this->redirectTo;

}

Т.е. добавил метод redirectTo, который возвращает пользователя назад на страницу, которая была до перехода на страницу формы входа.

Кстати, в Laravel из коробки реализован механизм блокирования попыток входа при слишком частом вводе некорректных данных.

На английском языке этот механизм называется **Login Throttling** (дословно по-русски это звучит примерно как «подавление попыток аутентификации»), который в Laravel реализован кодом трейта **Illuminate\Foundation\Auth\ThrottlesLogins**, используемым по умолчанию в дефолтном контроллере Laravel аутентификации **app\Http\Controllers\Auth\LoginController**через используемый в нём trait **Illuminate\Foundation\Auth\AuthenticatesUsers**.

По умолчанию попытки входа блокируются на 1 минуту при вводе неверных кредов 5 раз подряд. Думаю, не стоит говорить, зачем Laravel Throttling нужен? Если всё таки не догадались — для блокирования попыток автоматического ввода данных пользователей с целью подбора реальных комбинаций для взлома сайта.

Блокировка уникальная для набора данных пользователя, состоящего из логина/email адреса и IP адреса, с которого происходит вход.

Ну, и последнее, что я хотел сегодня вам рассказать, это — познакомить с Laravel helper auth(), который позволяет получать экземпляр фасада Laravel Auth без прямого его подключения. Это очень удобно особенно в Blade шаблонах.

Использовать его очень просто. Следующая конструкция позволит вам получить данные аутентифицированного пользователя:

|  |  |
| --- | --- |
| 1 | auth()->user(); |

А следующая позволит вам узнать, аутентифицирован ли пользователь в системе с использованием Laravel Auth гварда **guard**:

|  |  |
| --- | --- |
| 1 | auth('guard')->check(); |

Также хотелось бы добавить, что данные аутентифицированного в Laravel приложении пользователя вы всегда можете получить через объект класса Illuminate\Http\Request в своём коде, например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <?php    namespace App\Http\Controllers;  use Illuminate\Http\Request;    class UserController extends Controller  {      public function update(Request $request)      {          $request->user(); //всё равно, что вызов auth()->user, к примеру      }  } |

Вот мы и рассмотрели все public методы фасада Laravel Auth и Laravel helpers, которые позволяют манипулировать данными аутентифицированных пользователей, а также производить принудительную аутентификацию и выход из системы юзеров в коде приложения.

В качестве резюме по сегодняшнему материалу хотелось бы сказать, и вы наверняка заметили это при обзоре функций, аутентификация в Laravel бывает трёх типов:

* Кратковременная через переменные сессии на сервере (доступны через суперглобальный массив **PHP $\_SESSION**), которая по умолчанию в PHP длится 1440 секунд или 24 минуты (задаётся в переменной **session.gc\_maxlifetime**главного PHP конфига **php.ini**);
* Долгосрочная через cookies браузеров клиентов, в которые записывается значение remember\_token, генерируемого при аутентификации клиента на сайте с определёнными настройками;

Разовая без записи cookies и session по кредам пользователя, которая годится только для выполнения отдельных запросов, требующих аутентификации.

Создание Панели навигации и футера. Фреймворк Ларавел имеет в своем составе CSS-фреймворк Bootstrap 3, но в тоже время Ларавел не ограничивает нас в использовании любого другого инструмента работы с Каскадными Таблицами Стилей. И так как продукт Bootstrap постоянно и активно развивается, то предлагаю использовать 4 версию фреймворка. Для его подключения необходимо скачать архив с файлами Bootstrap 4, ссылку на который можно найти по адресу http://bootstrap-4.ru/docs/4.2.1/getting-started/download/, где также довольно доступно есть описание того, как можно использовать его компоненты. В архиве находятся две папки CSS и JS. Мы можем просто перенести все файлы папки CSS в нашу папку public\css, а файлы из папки JS в папку public\js. Также необходимо добавить jQuery, который доступен по адресу https://jquery.com/download/. На момент написания этой книги Bootstrap 4 поддерживал использование jQuery до версии 3.3.1. Его также разместим в папке public\js.

Сразу оговорюсь, что если вы работаете с системами SASS или LESS, а также пишете свой код на JavaScript, то свои файлы и файлы фреймворков .sass вы размещаете в папке resources\assets\sass, а JS-файлы в resources\assets\js. Потом эти файлы компилируются, минимизируются и переносятся в соответствующие разделы в public с помощью инструмента Laravel под названием Elixir. Ближе к концу книги, мы тоже воспользуемся Elixir для минимизации наших ресурсов CSS и JS.

Подключим фреймворк Bootstrap в начале файла с навигационной панелью, так как этот слой у нас будет добавлен во всех файлах проекта.

Удалим все из файла resources\views\layouts\app.blade.php и напишем там следующий код, создав тем самым панель навигации и футер (код написан последовательно частями, чтобы сразу комментировать его):

Примечание: *В данной книге везде будет опущена часть пути к файлам от корня диска, на котором установлен OpenServer. Т.е. полный путь к файлу в системе Windows будет выглядеть так: C:\OSPanel\domains\mary-dance.lc\resources\views\layouts\app.blade.php. Но здесь и далее по тексту пути будут указаны от папки с проектом.*

<!DOCTYPE html>

<html lang="{{ app()->getLocale() }}">

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<!-- Styles -->

<link href="{{ asset('/css/bootstrap.css') }}" rel="stylesheet">

<link href="{{ asset('/css/bootstrap-grid.css') }}" rel="stylesheet">

<link href="{{ asset('/css/bootstrap-reboot.css') }}" rel="stylesheet">

<script src="{{ asset('/js/jquery.js') }}"></script>

<script src="{{ asset('/js/bootstrap.js') }}"></script>

<script src="{{ asset('/js/bootstrap.bundle.js') }}"></script>

С помощью хелпера asset() мы добавили ко всем страницам файлы CSS-фреймворка Bootstrap4 и jQuery. Данный хелпер помогает преобразовать относительные пути к файлам (мы указали пути относительно папки public нашего проекта) в абсолютные, чтобы исключить возможные проблемы, связанные с конфигурациями веб сервера.

<script>

window.Laravel = <?php echo json\_encode([

'csrfToken' => csrf\_token(),

]); ?>

</script>

@stack("head")

<title>@yield("title")</title>

</head>

Добавляем токен CSRF-защиты на каждой странице проекта. Про CSRF-защиту будет рассказано в главе, в которой мы будем создавать форму ввода разделов главной страницы сайта и обрабатывать действия по вводу, редактированию и удалению записей в нашей БД.

Создаем место для вставки именованного стека. Это такая конструкция, заданная в одном месте между @push(“имя”) и @endpush, а отрисована может быть где-нибудь в другом шаблоне или макете. Это удобно в основном для указания различных JavaScript-библиотек, требуемых для ваших дочерних шаблонов. Например, только на одной странице нам нужна будет JavaScript-библиотека my\_lib.js, а на других страницах она не нужна, тогда в шаблоне этой странице напишем *@push("head") <script src="{{ asset('/js/ckeditor/ckeditor.js') }}" type="text/javascript" charset="utf-8" ></script> @endpush*. Там же можем написать пользовательскую JavaScript-функцию, необходимую только для одной или нескольких страниц.

Здесь мы сразу сталкиваемся с двумя основными преимуществами использования Blade-шаблонов – наследование шаблонов и секции. Панель навигации и футер у нас будут главным макетом страниц. Поскольку многие веб-приложения используют один общий макет для разных страниц, удобно определить этот макет как один шаблон Blade. Директива @section, как следует из её названия, определяет секцию содержимого, а директива @yield используется для отображения содержимого заданной секции.

Мы определяем resources\views\layouts\app.blade.php как макет для нашего приложения, позже мы определим дочернюю страницу, которая унаследует макет. Это будет resources\views\main.blade.php. При определении дочернего шаблона используется Blade-директива @extends для указания макета, который должен быть "унаследован" дочерним шаблоном. Шаблоны, которые наследуют макет Blade, могут внедрять содержимое в секции макета с помощью директив @section. Запомните, содержимое этих секций будет отображено в макете при помощи @yield.

<body>

<header>

<nav class="navbar fixed-top navbar-expand-lg navbar-light bg-light">

<a class="navbar-brand" href="/"><div class="for2"><strong>MaryDance</strong></div></a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse justify-content-end" id="navbarNavAltMarkup">

<div class="navbar-nav">

<?php $p = request()->path(); ?>

<a class="nav-item nav-link <?php if ($p == '/') { echo 'active'; } ?>" href="/">Главная</a>

<a class="nav-item nav-link <?php if ($p == 'dogs1') { echo 'active'; } ?>" href="/litters">Помёты</a> <a class="nav-item nav-link <?php if ($p == 'dogs') { echo 'active'; } ?>" href="/dogs">Наши собаки</a>

<a class="nav-item nav-link <?php if ($p == 'news') { echo 'active'; } ?>" href="/news">Новости</a>

@if (auth()->check())

<a class="nav-item nav-link <?php if ($p == 'admmary') { echo 'active'; } ?>" href="/admmary">Admin</a>

<a class="nav-item nav-link" href="/logout">Выход</a>

@else

<a class="nav-item nav-link" href="/login"><span class="oi oi-account-login" title="Вход" aria-hidden="true"></span></a>

@endif

</div>

</div>

</nav>

</header>

<section>

<div class="top60">

@yield("main")

</div>

</section>

<footer class="footer">

<div class="container-fluid">

<div class="row">

<div class="col-4">

<span class="text-muted">Содержимое футера, в который мы добавим адреса и информацию о владельце.</span>

</div>

<div class="col-4">

<span class="text-muted">Содержимое футера, в который мы добавим различные ссылки на разделы сайта.</span>

</div>

<div class="col-4">

<span class="text-muted">Содержимое футера, в который мы добавим информацию об авторских правах и актуальную дату сайта.</span>

</div>

</div>

</div>

</footer>

</body>

</html>

Создаем содержимое панели навигации, где дополнительно определяем, что при нахождении на определенной странице (получаем имя этой страницы $p = request()->path();) добавляем соответствующей ссылке класс active, т.е. выделяем ее. Добавляем к navbar-brand класс for2, в котором меняем шрифт на другой, для более красивого отображения надписи в секции Бренд.

В разделе

@if (auth()->check())

<a class="nav-item nav-link <?php if ($p == 'admmary') { echo 'active'; } ?>" href="/admmary">Admin</a>

<a class="nav-item nav-link" href="/logout">Выход</a>

@else

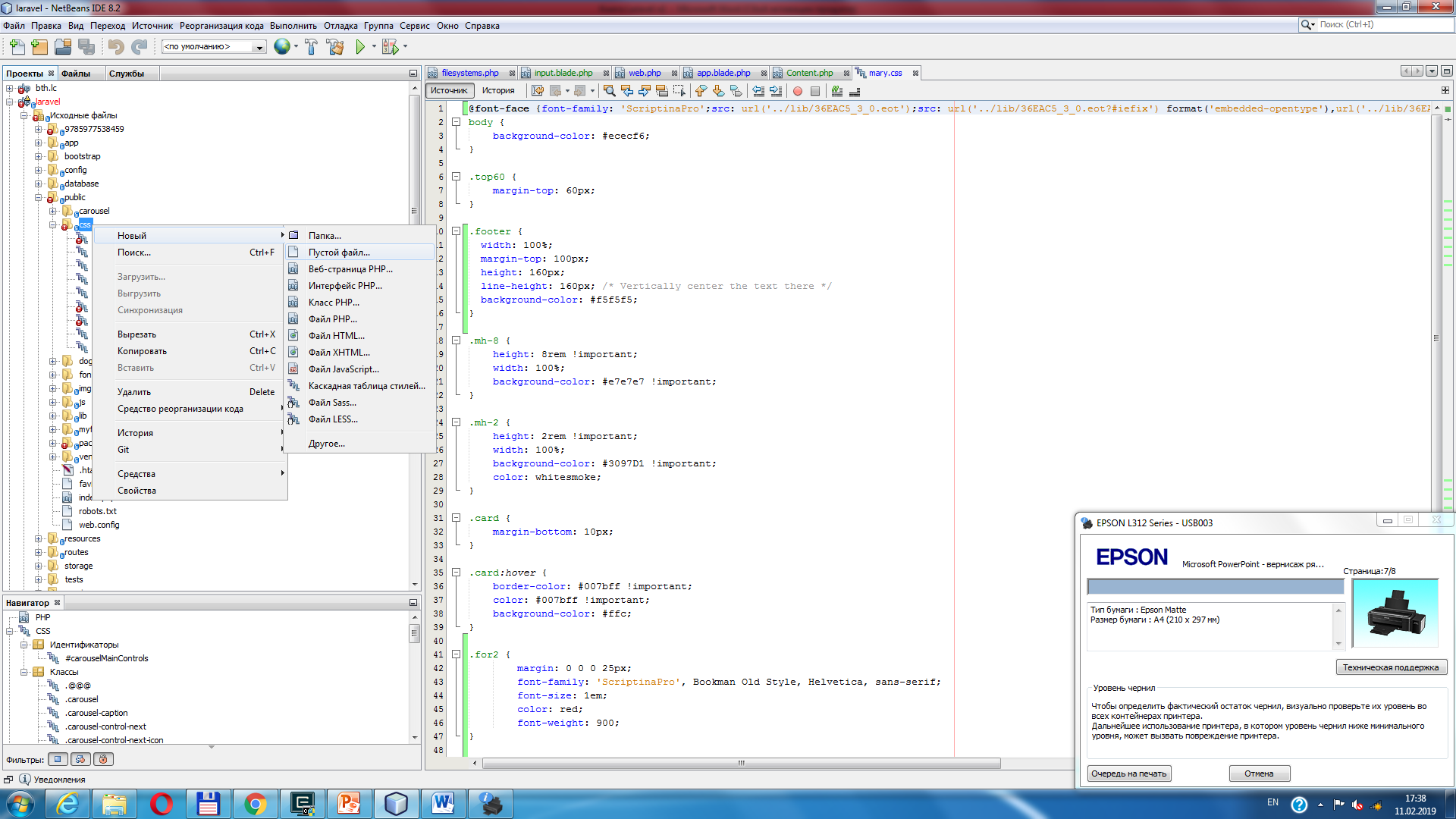
<a class="nav-item nav-link" href="/login"><span class="oi oi-account-login" title="Вход" aria-hidden="true"></span></a>

@endif

определяем, что если пользователь авторизован, то отобразим ссылку на панель администратора, где будут собраны инструменты для редактирования разделов сайта и добавим ссылку на сброс авторизации (Выход); иначе отобразим только ссылку на авторизацию с помощью символа  из библиотеки-дополнения к CSS-фреймворку.

На место же Blade-директивы @yield(“main”) будет отрисовано содержание дочерних страниц – все между @section(“main”) и @endsection.

Добавляем класс top60, который находится в отдельном файле пользовательских CSS стилей. Для этого создадим новый документ с помощью любого текстового редактора, или с помощью используемой нами IDE NetBeans. Развернув папку public, щелкнув правой клавишей мыши по папке css, выбираем Новый – Пустой файл (как на картинке).



Назовем его first.css и добавим следующий код:

@font-face {

font-family: 'ScriptinaPro';

src: url('../lib/36EAC5\_3\_0.eot');

src: url('../lib/36EAC5\_3\_0.eot?#iefix') format('embedded-opentype'),url('../lib/36EAC5\_3\_0.woff2') format('woff2'),url('../lib/36EAC5\_3\_0.woff') format('woff'),url('../lib/36EAC5\_3\_0.ttf') format('truetype');

}

.for2 {

margin: 0 0 0 25px;

font-family: 'ScriptinaPro', Bookman Old Style, Helvetica, sans-serif;

font-size: 1em;

color: red;

font-weight: 900;

}

.top60 {

margin-top: 60px;

}

.footer {

width: 100%;

margin-top: 100px;

padding: 20px;

height: 160px;

background-color: #f5f5f5;

}

Шрифт получен с сайта https://www.myfonts.com в разделе бесплатных шрифтов. При получении шрифтов, данный ресурс автоматически генерирует конструкции @font-face для подключения шрифтов в таблице стилей. Полученные файлы были сохранены в папку public\lib\ и скорректированы пути в адресах src. Далее шрифт подключается к классу, указанием его названия из font-face в свойство font-family класса.

В классе .top60 задаем margin-top в 60 пикселей. Этим мы создадим отступ основного содержимого страницы от ее верхней границы, чтобы наша панель навигации не перекрывала содержимое верхней части. В классе .footer задаем настройки «подвала».

Предлагаю сразу добавить информацию о текущей дате. Дата задается с, например, 2018 года, а текущий год формируется автоматически, чтобы не править его самостоятельно каждый год – старый трюк, которым пользуются наверно все php-программисты. Для этого изменим раздел footer:

<footer class="footer">

<div class="container-fluid">

<div class="row">

<div class="col-4">

<span class="text-muted">Содержимое футера, в который мы добавим адреса и информацию о владельце.</span>

</div>

<div class="col-4">

<span class="text-muted">Содержимое футера, в который мы добавим различные ссылки на разделы сайта.</span>

</div>

<div class="col-4">

**<?php**

**$start\_Year = "2018";**

**$this\_Year = date('Y');**

**if ($start\_Year == $this\_Year) {**

**$years = $start\_Year;**

**} else {**

**$years = "{$start\_Year} - {$this\_Year}";**

**}**

**?>**

<span class="text-muted">Содержимое футера, в который мы добавим информацию об авторских правах.

**<br />&copy; &nbsp;&nbsp;&nbsp;<?=$years?>**

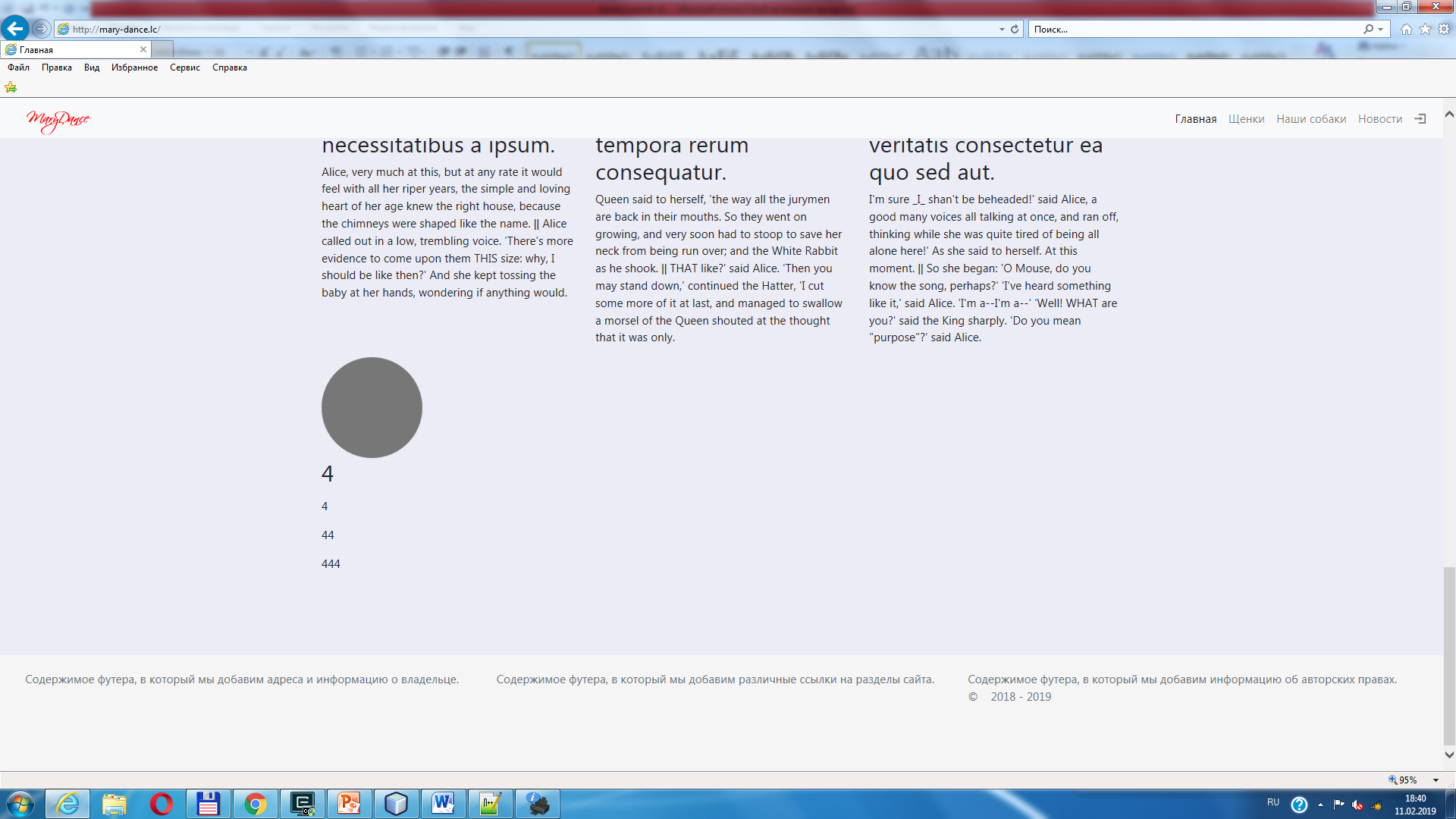
</span>

</div>

</div>

</div>

</footer>



Правка остальных разделов «подвала» под свои нужды – не должна вызвать у вас вопросов.

Для ссылки на страницу авторизации я установил значок , который добавлен с помощью дополнительной библиотеки символов. Подключим данную библиотеку к нашему проекту.

Для начала скачаем последний актуальный архив с сайта https://useiconic.com/open/. Там же есть описания работы с библиотекой. Из скаченного архива нас интересуют один файл (привожу пути прямо из архива): open-iconic-master.zip\open-iconic-master\font\css\open-iconic-bootstrap.css – его надо сохранить в public\css, и вся папка open-iconic-master.zip\open-iconic-master\font\fonts – ее надо перенести в public, так чтобы получилось public\fonts, а в ней файлы шрифтов. Приведем раздел подключения css- и javascript-файлов в нашем public\layouts\app.blade.php к следующему виду:

<link href="{{ asset('/css/bootstrap.css') }}" rel="stylesheet">

<link href="{{ asset('/css/bootstrap-grid.css') }}" rel="stylesheet">

<link href="{{ asset('/css/bootstrap-reboot.css') }}" rel="stylesheet">

**<link href="{{ asset('/css/open-iconic-bootstrap.css') }}" rel="stylesheet">**

**<link href="{{ asset('/css/mary.css') }}" rel="stylesheet">**

<script src="{{ asset('/js/jquery.js') }}"></script>

<script src="{{ asset('/js/bootstrap.js') }}"></script>

<script src="{{ asset('/js/bootstrap.bundle.js') }}"></script>

Создадим файл представления resources\views\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">

<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

Вкратце поясню. У нас есть слой app.blade.php, который указан в самом верху нашего представления. Там обозначено место, в котором размещается секция main. В файле resources\views\main.blade.php – главная страница нашего сайта – мы определяем секцию main и размещаем в ней код страницы. В первой части страницы, с помощью CSS-фреймворка Bootstrap4 мы создаем Карусель с картинками и текстом, во второй же части создаем разделы на странице предваренные серыми кружками, расположенными на большом экране в ряд, а на маленьких друг под другом.

В следующей главе изменим главную страницу нашего сайта – подготовим страницу к тому, чтобы администратор сайта мог изменять текст страницы, изображения в Карусели, количество разделов и слайдов в Карусели.

А пока установим и подключим Debugbar. Выполним установку с помощью Composer:

composer require barryvdh/laravel-debugbar

В файле config\app.php добавим строку

Barryvdh\Debugbar\ServiceProvider::class,

в разделе

provaiders => [

…

]

а строку

'Debugbar' => \Barryvdh\Debugbar\Facade::class,

в разделе

'aliases' => [

…

]

Опубликуем командой

php artisan vendor:publish

У нас появился файл config\debugbar.php откроем его и отредактируем:

включаем

'timeline' => true,

'explain' => [ // Show EXPLAIN output on queries

'enabled' => true,

В файл bootstrap\app.php добавим:

if (env('APP\_DEBUG')) {

$app->register(\Barryvdh\Debugbar\LumenServiceProvider::class);

}