Бесплатная электронная книга

УЧУСЬ

Laravel

Free unaffiliated eBook created from
Stack Overflow contributors.

#laravel

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: laravel

It is an unofficial and free Laravel ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Laravel.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с Laravel

замечания

Сообщество Laravel StackOverflow Slack

Скоро будет

Рекомендуемое учебное пособие

Начало работы с Laravel

Руководства по взносам

Скоро будет

Руководство по стилю вклада

Скоро будет

Информация о Ларавеле

Созданный Taylor Otwell в качестве бесплатной открытой веб-структуры PHP с открытым исходным кодом , Laravel предназначен для облегчения и ускорения процесса разработки веб-приложений с большим вкусом для простоты.

Он следует за архитектурным шаблоном model-view-controller (MVC), а также стандартом кодирования PSR-2 и стандартом автозагрузки PSR-4 .

Запуск тестового развития (TDD) в Laravel - это весело и легко реализовать.

Хостинг на GitHub и доступный по адресу https://github.com/laravel/laravel , Laravel может похвастаться архитектурой микросервисов , что делает его чрезвычайно расширяемым, и это с легкостью использует пользовательские или существующие сторонние пакеты.

Основные функции

MVC

Laravel использует модель MVC, поэтому есть три основные части структуры, которые работают вместе: модели, представления и контроллеры. Контроллеры являются основной частью, где выполняется большая часть работы. Они подключаются к моделям для получения, создания или обновления данных и отображения результатов на представлениях, которые содержат фактическую структуру HTML приложения.

Blade Templating Engine

Laravel поставляется с темплатным двигателем, известным как Blade. Blade довольно прост в использовании, но мощный. Одна особенность Blade templating engine не делится с другими популярными, это ее вседозволенность; позволяя использовать простой PHP-код в файлах шаблонов Blade.

Важно отметить, что файлы движка Blade `.blade` имеют `.blade` добавленные к именам файлов непосредственно перед обычным `.php` который не что иное, как фактическое расширение файла. Таким образом, `.blade.php` является результирующим расширением файла для файлов шаблонов Blade. Файлы движка Blade хранятся в каталоге ресурсов / представлений.

Маршрутизация и промежуточное ПО

Вы можете определить URL-адреса своего приложения с помощью маршрутов. Эти маршруты могут содержать переменные данные, подключаться к контроллерам или быть обернутыми в средние. Middelware - это механизм фильтрации HTTP-запросов. Они могут использоваться для взаимодействия с запросами до того, как они дойдут до контроллеров, и могут, таким образом, изменять или отклонять запросы.

ремесленник

Artisan - инструмент командной строки, который вы можете использовать для управления частями Laravel. Существует множество команд для создания моделей, контроллеров и других ресурсов, необходимых для разработки. Вы также можете написать свои собственные команды, чтобы расширить инструмент командной строки Artisan.

Красноречивый ORM

Чтобы подключить ваши модели к различным типам баз данных, Laravel предлагает собственный ORM с большим набором функций для работы. Структура также обеспечивает миграцию и посев, а также функции откатов.

Обработка событий

Структура способна обрабатывать события в приложении. Вы можете создавать прослушиватели событий и обработчики событий, похожие на те, что были у NodeJ.

Версии

Версия	Дата выхода
1,0	2011-06-09
2,0	2011-11-24
3.0	2012-02-22
3,1	2012-03-27
3,2	2012-05-22
4,0	2013-05-28
4,1	2013-12-12
4,2	2014-06-01
5.0	2015-02-04
5.1 (LTS)	2015-06-09
5,2	2015-12-21
5,3	2016-08-24
5,4	2017-01-24

Examples

Добро пожаловать в документацию по тегам Laravel!

Laravel - это хорошо известная PHP Framework. Здесь вы узнаете все о Laravel. Исходя из всего *прочего* - *как* знать, что такое объектно-ориентированное программирование, к передовой теме развития пакета Laravel.

Это, как и все другие теги документации Stackoverflow, является документацией, основанной на сообществах, поэтому, если у вас уже есть опыт работы в Laravel, поделитесь своими знаниями, добавьте свои собственные темы или примеры! Просто не забудьте ознакомиться с нашим **руководством** по **стилю** в этом разделе, чтобы узнать больше о том, как внести свой вклад, и руководство по стилю, которое мы сделали, чтобы

мы могли дать лучший опыт людям, которые хотят узнать больше о Laravel.

Более того, мы очень рады, что вы пришли, надеюсь, что вас часто можно увидеть здесь!

Руководство для начинающих

Руководство для начинающих - это настраиваемая навигация, которую мы сами заказывали, чтобы облегчить просмотр тем, особенно для начинающих. Эта навигация упорядочена по уровню сложности.

Начиная

Монтаж

Laravel Views

Блейд: Введение

Блейд: переменные и управляющие структуры

Или же

Установка отсюда

1. Получите композитора отсюда и установите его
2. Получите Wamp здесь , установите его и установите переменную среды PHP
3. Получить путь к команде `www` и type:

```
composer create-project --prefer-dist laravel/laravel projectname
```

Чтобы установить конкретную версию Laravel, введите путь к `www` и введите команду:

```
composer create-project --prefer-dist laravel/laravel=DESIRED_VERSION projectname
```

Или же

Установщик Via Laravel

Сначала загрузите установщик Laravel с помощью Composer:

```
composer global require "laravel/installer"
```

Обязательно поместите `$HOME/.composer/vendor/bin` (или эквивалентный каталог для вашей ОС) в ваш $ PATH, чтобы исполняемый файл `laravel` мог быть расположен вашей системой.

После установки `laravel new` команда `laravel new` создаст новую установку Laravel в указанном вами каталоге. Например, `laravel new blog` создаст каталог с `blog` содержащий новую установку Laravel со всеми уже установленными зависимостями Laravel:

```
laravel new blog
```

Прочитайте Начало работы с Laravel онлайн: https://riptutorial.com/ru/laravel/topic/794/начало-работы-с-laravel

глава 2: HTML и Form Builder

Examples

Монтаж

HTML и Form Builder не являются основным компонентом с Laravel 5, поэтому нам нужно установить его отдельно:

```
composer require laravelcollective/html "~5.0"
```

Наконец, в `config/app.php` нам необходимо зарегистрировать поставщика услуг, а псевдонимы фасадов:

```
'providers' => [
    // ...
    Collective\Html\HtmlServiceProvider::class,
    // ...
],

'aliases' => [
   // ...
  'Form' => Collective\Html\FormFacade::class,
  'Html' => Collective\Html\HtmlFacade::class,
   // ...
],
```

Полные документы доступны в формах и HTML

Прочитайте HTML и Form Builder онлайн: https://riptutorial.com/ru/laravel/topic/3672/html-и-form-builder

глава 3: Laravel Docker

Вступление

Задача, с которой сталкиваются все разработчики и разработчики, - согласованность условий. Сегодня Laravel является одной из самых популярных фреймворков PHP. DDocker, с другой стороны, является методом виртуализации, который устраняет проблемы *«работает на моей машине»* при взаимодействии с кодом с другими разработчиками. Эти два вместе создают слияние **полезных** и **мощных** . Хотя оба они делают очень разные вещи, их можно объединить, чтобы создавать удивительные продукты.

Examples

Использование Laradock

Laradock - это проект, который обеспечивает готовность к работе, предназначенную для использования в Laravel.

Загрузите или клонируйте Laradock в корневую папку вашего проекта:

```
git clone https://github.com/Laradock/laradock.git
```

Измените каталог на Laradock и сгенерируйте файл `.env` необходимый для запуска ваших конфигураций:

```
cd laradock
cp .env-example .env
```

Теперь вы готовы запустить докер. При первом запуске контейнера он загрузит все необходимые пакеты из Интернета.

```
docker-compose up -d nginx mysql redis beanstalkd
```

Теперь вы можете открыть свой браузер и просмотреть свой проект на `http://localhost` .

Для полной документации и конфигурации Laradock нажмите здесь .

Прочитайте Laravel Docker онлайн: https://riptutorial.com/ru/laravel/topic/10034/laravel-docker

глава 4: авторизация

Вступление

Laravel предоставляет простой способ авторизации действий пользователя на определенных ресурсах. С помощью авторизации вы можете выборочно разрешать пользователям доступ к определенным ресурсам, не отказывая в доступе к другим. Laravel предоставляет простой API для управления авторизацией пользователей с помощью `Gates` and `Policies` . `Gates` предоставляет простой подход, основанный на закрытии, с использованием `AuthServiceProvider` то время как `Policies` позволяют организовывать логику авторизации вокруг моделей с использованием классов.

Examples

Использование ворот

`Gates` - это замыкания, которые определяют, разрешено ли пользователю выполнять определенное действие на ресурсе. `Gates` обычно определяется в методе загрузки `AuthServiceProvider` и лаконично назван, чтобы отразить то, что он делает. Пример шлюза, который позволяет только премиум-пользователям просматривать некоторый контент, будет выглядеть так:

```
Gate::define('view-content', function ($user, $content){
    return $user->isSubscribedTo($content->id);
});
```

А `Gate` всегда получает экземпляр пользователя в качестве первого аргумента, вам не нужно передавать его при использовании затвора и необязательно получать дополнительные аргументы, такие как вызывающая проблема красноречивая модель.

Авторизация действий с помощью ворот

Чтобы использовать пример выше на шаблоне клинка, чтобы скрыть контент от пользователя, вы обычно делаете что-то вроде этого:

```
@can('view-content', $content)
 <! -- content here -->
@endcan
```

Чтобы полностью предотвратить навигацию по контенту, вы можете сделать следующее в своем контроллере:

```
if(Gate::allows('view-content', $content)){
```

```
    /* user can view the content */
}

        OR

if(Gate::denies('view-content', $content)){
    /* user cannot view content */
}
```

Примечание. Вам не требуется передавать текущий аутентифицированный пользователь в этот метод, Laravel позаботится об этом для вас.

полисы

Политики - это классы, которые помогают организовать логику авторизации вокруг ресурса модели. Используя наш предыдущий пример, у нас может быть `ContentPolicy` который управляет доступом пользователей к модели `Content` .

Для того, чтобы `ContentPolicy` , Laravel обеспечивает команду мастеровых. Просто запустите

```
php artisan make:policy ContentPolicy
```

Это создаст пустой класс политики и разместит его в папке « `app/Policies` ». Если папка не существует, Laravel создаст ее и поместит класс внутрь.

После создания политики необходимо зарегистрировать, чтобы помочь Laravel узнать, какие политики использовать при авторизации действий над моделями. В `AuthServiceProvider` Laravel, который поставляется со всеми новыми установками Laravel, есть свойство политик, которое сопоставляет ваши красноречивые модели с их политиками авторизации. Все, что вам нужно сделать, добавить отображение в массив.

```
protected $policies = [
    Content::class => ContentPolicy::class,
 ];
```

Написание политик

`Policies` написания следуют по той же схеме, что и письмо `Gates` . В качестве политики можно переписать элемент разрешения контента:

```
function view($user, $content)
{
    return $user->isSubscribedTo($content->id);
}
```

Политики могут содержать больше методов по мере необходимости, чтобы заботиться обо всех случаях авторизации для модели.

Авторизация действий с политиками

Через модель пользователя

Модель пользователя Laravel содержит два метода, которые помогают с разрешениями с использованием `Policies` ; `can` и `can't` . Эти два могут использоваться, чтобы определить, имеет ли пользователь авторизацию или нет на модели, соответственно.

Чтобы проверить, может ли пользователь просматривать содержимое или нет, вы можете сделать следующее:

```
if($user->can('view', $content)){
    /* user can view content */
}

        OR

if($user->cant('view', $content)){
    /* user cannot view content */
}
```

Через промежуточное ПО

```
Route::get('/contents/{id}, function(Content $content){
    /* user can view content */
})->middleware('can:view,content');
```

Через контроллеры

Laravel предоставляет вспомогательный метод, называемый `authorize` который принимает имя политики и связанной модели в качестве аргументов и либо разрешает действие на основе вашей логики авторизации, либо отрицает действие, и выдает исключение `AuthorizationException` которое обработчик Laravel Exception преобразует в `403 HTTP response` .

```
pubic function show($id)
{
    $content = Content::find($id);

    $this->authorize('view', $content);

    /* user can view content */
}
```

Прочитайте авторизация онлайн: https://riptutorial.com/ru/laravel/topic/9360/авторизация

глава 5: Аутентификация

Examples

Многопользовательская аутентификация

Laravel позволяет использовать несколько типов аутентификации с определенными защитами.

В laravel 5.3 множественная аутентификация немного отличается от Laravel 5.2

Я объясню, как реализовать функцию мультиаутентификации в 5.3

Сначала вам понадобится две разные модели пользователей

```
cp App/User.php App/Admin.php
```

изменить имя класса на Admin и задать пространство имен, если вы используете разные модели. это должно выглядеть

App \ admin.php

```
<?php

namespace App;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class Admin extends Authenticatable
{
    use Notifiable;

    protected $fillable = ['name', 'email', 'password'];
    protected $hidden   = ['password', 'remember_token'];

}
```

Также вам нужно создать миграцию для администратора

```
php artisan make:migration create_admins_table
```

затем отредактируйте файл миграции с содержимым миграции пользователей по умолчанию. Похож на это

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
```

```
use Illuminate\Support\Facades\Schema;

class CreateAdminsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('admins', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();

            $table->softDeletes();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('admins');
    }
}
```

edit **config / auth.php**

```
'guards'    => [
        'web'   => [
            'driver'   => 'session',
            'provider' => 'users',
        ],

        'api'   => [
            'driver'   => 'token',
            'provider' => 'users',
        ],
        //Add Admin Guard
        'admin' => [
            'driver'   => 'session',
            'provider' => 'admins',
        ],
    ],
```

а также

```
'providers' => [
        'users'  => [
            'driver' => 'eloquent',
            'model'  => App\User::class,
```

```
        ],
        //Add Admins Provider
        'admins' => [
            'driver' => 'eloquent',
            'model'  => App\Admin::class,
        ],
    ],
```

Обратите внимание, что мы добавляем две записи. один из **охранников -** переменный в переменной **поставщика** .

И так вы используете другого охранника, затем «веб»,

Мое приложение \ Http \ Controllers \ Admin \ LoginController

```
<?php

namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use Illuminate\Foundation\Auth\AuthenticatesUsers;
use Illuminate\Support\Facades\Auth;

class AuthController extends Controller
{

    use AuthenticatesUsers;

    protected $guard = 'admin';

    protected $redirectTo = '/admin/';

    public function showLoginForm()
    {
        return view('admin.login');
    }

    protected function guard()
    {
        return Auth::guard($this->guard);
    }

}
```

это мало объясняет.

вкратце **Auth :: guard ('admin')** позволит вам использовать auth-методы (такие как логин, выход из системы, регистрация и т. д.) с вашим администратором.

Например

```
Auth::guard('admin')->login($user)
```

будет искать $ user в таблице admins и логин с пользователем, пока

```
Auth::login($user)
```

будет нормально работать с таблицей пользователей. **Защита по** *умолчанию* указана в **config / auth.php** с массивом по *умолчанию* . В свежем ларавеле это «паутина».

В контроллере вы должны реализовать методы из AuthenticatesUsers, чтобы показать свои настраиваемые пути просмотра. И вам нужно реализовать другие функции, такие как защита, чтобы использовать новые защитники пользователей.

В этом примере мой логин **администратора** - **admin / login.blade**

И, реализуя функцию guard (), чтобы вернуть **Auth :: guard ('admin'),** все методы атрибутов AuthenticatesUsers работают с защитой «admin».

В более ранних версиях laravel это немного отличается от 5.3

в 5.2 функция getGuard возвращает переменную $ guard из класса и основной функции (логин), используя ее в

```
Auth::guard($guard)->attempt(...)
```

в 5.3 функция охраны возвращает весь Auth :: guard (), а основная функция использует его как

```
$this->guard()->attempt(...)
```

Прочитайте Аутентификация онлайн: https://riptutorial.com/ru/laravel/topic/7051/аутентификация

глава 6: База данных

Examples

Несколько соединений с базой данных

Laravel позволяет пользователю работать с несколькими подключениями к базе данных. Если вам необходимо подключиться к нескольким базам данных и заставить их работать вместе, вы можете быть уверены в настройке соединения.

Вы также можете использовать различные типы баз данных в одном приложении, если это необходимо.

Соединение по умолчанию В `config/database.php` вы можете увидеть вызов элемента конфигурации:

```
'default' => env('DB_CONNECTION', 'mysql'),
```

Это имя ссылается на имя соединения `mysql` :

```
'connections' => [

    'sqlite' => [
        'driver' => 'sqlite',
        'database' =>  database_path('database.sqlite'),
        'prefix' => '',
    ],

    'mysql' => [
        'driver' => 'mysql',
        'host' => env('DB_HOST', 'localhost'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'charset' => 'utf8',
        'collation' => 'utf8_unicode_ci',
        'prefix' => '',
        'strict' => false,
        'engine' => null,
    ],
],
```

Если вы не указали название соединения с базой данных в других кодах или командах, Laravel подберет имя подключения к базе данных по умолчанию. однако, в нескольких соединениях с базой данных, даже если вы настроили соединение по умолчанию, вам лучше настроить повсеместно, какое соединение с базой данных вы использовали.

Файл миграции

В файле миграции, если используется одно соединение с базой данных, вы можете использовать:

```
 Schema::create("table",function(Blueprint $table){
     $table->increments('id');
});
```

В нескольких подключениях к базе данных вы будете использовать метод `connection()` чтобы сообщить Laravel, какое соединение с базой данных вы используете:

```
 Schema::connection("sqlite")->create("table",function(Blueprint $table){
     $table->increments('id');
});
```

Artisan Migrate

если вы используете одно соединение с базой данных, вы запустите:

```
php artisan migrate
```

Тем не менее, для множественного подключения к базе данных вам лучше указать, какое соединение с базой данных поддерживает данные миграции. поэтому вы выполните следующую команду:

```
php artisan migrate:install --database=sqlite
```

Эта команда установит таблицу миграции в целевой базе данных для подготовки миграции.

```
php artisan migrate --database=sqlite
```

Эта команда выполнит миграцию и сохранит данные миграции в целевой базе данных

```
php artisan migrate:rollback --database=sqlite
```

Эта команда выполняет откат миграции и сохраняет данные миграции в целевой базе данных

Красноречивая модель

Чтобы указать соединение с базой данных с использованием Eloquent, вам необходимо определить свойство `$connection` :

```
namespace App\Model\Sqlite;
class Table extends Model
{
    protected $table="table";
    protected $connection = 'sqlite';
```

```
}
```

Чтобы указать другое (второе) соединение с базой данных с помощью Eloquent:

```
namespace App\Model\MySql;
class Table extends Model
{
    protected $table="table";
    protected $connection = 'mysql';
}
```

Laravel будет использовать свойство `$connection` определенное в модели, для использования указанного соединения, определенного в `config/database.php` . Если свойство `$connection` не определено в модели, будет использоваться значение по умолчанию.

Вы также можете указать другое соединение, используя метод static `on` :

```
// Using the sqlite connection
Table::on('sqlite')->select(...)->get()
// Using the mysql connection
Table::on('mysql')->select(...)->get()
```

Создание базы данных / запросов

Вы также можете указать другое соединение с помощью построителя запросов:

```
// Using the sqlite connection
DB::connection('sqlite')->table('table')->select(...)->get()
// Using the mysql connection
DB::connection('mysql')->table('table')->select(...)->get()
```

Модульный тест

Laravel предоставляет `seeInDatabase($table,$fielsArray,$connection)` для проверки кода `seeInDatabase($table,$fielsArray,$connection)` к базе данных. В тестовом файле Unit вам нужно сделать следующее:

```
$this
    ->json(
        'GET',
        'result1/2015-05-08/2015-08-08/a/123'
    )
     ->seeInDatabase("log", ["field"=>"value"], 'sqlite');
```

Таким образом, Laravel будет знать, какое соединение с базой данных тестируется.

Транзакции базы данных в модульном тестировании

Laravel позволяет базе данных откатывать все изменения во время тестов. Для тестирования нескольких соединений с базами данных вам необходимо установить

свойства `$connectionsToTransact`

```
use Illuminate\Foundation\Testing\DatabaseMigrations;

class ExampleTest extends TestCase
{
     use DatabaseTransactions;

     $connectionsToTransact =["mysql","sqlite"] //tell Laravel which database need to rollBack

    public function testExampleIndex()
    {
        $this->visit('/action/parameter')
         ->see('items');
    }
}
```

Прочитайте База данных онлайн: https://riptutorial.com/ru/laravel/topic/1093/база-данных

глава 7: Введение в laravel-5.2

Вступление

Laravel - это структура MVC с пакетами, миграциями и CLIS Artisan. Laravel предлагает надежный набор инструментов и архитектуру приложений, которая включает в себя многие из лучших функций таких фреймворков, как CodeIgniter, Yii, ASP.NET MVC, Ruby on Rails, Sinatra и другие. Laravel - это Open Source framework. Он имеет очень богатый набор функций, которые повысят скорость веб-разработки. Если вы знакомы с Core PHP и Advanced PHP, Laravel упростит вашу задачу. Это сэкономит много времени.

замечания

В этом разделе представлен обзор того, что такое laravel-5.1, и почему разработчик может захотеть его использовать.

Следует также упомянуть любые большие предметы в пределах laravel-5.1 и ссылки на связанные темы. Поскольку документация для laravel-5.1 является новой, вам может потребоваться создать начальные версии этих связанных тем.

Examples

Установка или настройка

Инструкции по установке Laravel 5.1 на машине Linux / Mac / Unix.

Перед началом установки проверьте, соблюдены ли следующие требования:

- PHP> = 5.5.9
- Расширение PHP OpenSSL
- Расширение PDO PHP
- Расширение PHP Mbstring
- Расширение Tokenizer PHP

Начнем установку:

1. Установите композитор. Документация композитора
2. Запустить `composer create-project laravel/laravel <folder-name> "5.1.*"`
3. Убедитесь, что папка `storage` папка `bootstrap/cache` доступны для записи.
4. Откройте файл `.env` и установите информацию о конфигурации, такую как учетные данные базы данных, статус отладки, прикладную среду и т. Д.
5. Запустите `php artisan serve` и укажите ваш браузер на `http://localhost:8000` . Если все в

порядке, вы должны получить страницу

Установите Laravel 5.1 Framework на Ubuntu 16.04, 14.04 и LinuxMint

Шаг 1 - Установите LAMP

Чтобы начать с Laravel, сначала нам нужно настроить запущенный сервер LAMP. Если у вас уже запущен стек LAMP, пропустите этот шаг, используйте следующие команды для настройки лампы в системе Ubuntu.

Установить PHP 5.6

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository ppa:ondrej/php
$ sudo apt-get update
$ sudo apt-get install -y php5.6 php5.6-mcrypt php5.6-gd
```

Установка Apache2

```
$ apt-get install apache2 libapache2-mod-php5
```

Установка MySQL

```
$ apt-get install mysql-server php5.6-mysql
```

Шаг 2 - Установите композитор

Композитор необходим для установки зависимостей Laravel. Поэтому используйте нижеприведенные команды для загрузки и использования в качестве команды в нашей системе.

```
$ curl -sS https://getcomposer.org/installer | php
$ sudo mv composer.phar /usr/local/bin/composer
$ sudo chmod +x /usr/local/bin/composer
```

Шаг 3 - Установите Laravel

Чтобы загрузить последнюю версию Laravel, используйте команду ниже для клонирования мастер-репо laravel из github.

```
$ cd /var/www
$ git clone https://github.com/laravel/laravel.git
```

Перейдите в каталог кода Laravel и используйте композитор для установки всех зависимостей, необходимых для рамки Laravel.

```
$ cd /var/www/laravel
```

```
$ sudo composer install
```

Установка зависимостей займет некоторое время. После установки правильных разрешений на файлы.

```
$ chown -R www-data.www-data /var/www/laravel
$ chmod -R 755 /var/www/laravel
$ chmod -R 777 /var/www/laravel/app/storage
```

Шаг 4 - Установите ключ шифрования

Теперь установите 32-битный ключ шифрования случайных чисел, который используется службой шифрования Illuminate.

```
$ php artisan key:generate

Application key [uOHTNu3Au1Kt7Uloyr2Py9blU0J5XQ75] set successfully.
```

Теперь отредактируйте конфигурационный файл `config/app.php` и обновите его выше сгенерированного ключа приложения, как `config/app.php` ниже. Также убедитесь, что шифр установлен правильно.

```
'key' => env('APP_KEY', 'uOHTNu3Au1Kt7Uloyr2Py9blU0J5XQ75'),

'cipher' => 'AES-256-CBC',
```

Шаг 5 - Создайте Apache VirtualHost

Теперь добавьте виртуальный хост в ваш конфигурационный файл Apache для доступа к среде Laravel из веб-браузера. Создайте файл конфигурации Apache в каталоге `/etc/apache2/sites-available/` и добавьте содержимое ниже.

```
$ vim /etc/apache2/sites-available/laravel.example.com.conf
```

Это файловая структура виртуального хоста.

```
<VirtualHost *:80>

        ServerName laravel.example.com
        DocumentRoot /var/www/laravel/public

        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/laravel>
                AllowOverride All
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log
        LogLevel warn
```

```
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Наконец, разрешить работу с сайтом и перезагрузить службу Apache, используя команду ниже.

```
$ a2ensite laravel.example.com
$ sudo service apache2 reload
```

Шаг 6 - Доступ к Laravel

На этом этапе вы успешно завершили фреймворк Laravel 5 PHP в своей системе. Теперь сделайте запись файла хоста для доступа к вашему приложению Laravel в веб-браузере. Измените `127.0.0.1` на свой сервер ip и `laravel.example.com` с вашим доменным именем, настроенным в Apache.

```
$ sudo echo "127.0.0.1  laravel.example.com" >> /etc/hosts
```

И доступ к http://laravel.example.com в вашем любимом веб-браузере, как показано ниже.

Прочитайте Введение в laravel-5.2 онлайн: https://riptutorial.com/ru/laravel/topic/1987/введение-в-laravel-5-2

глава 8: Введение в laravel-5.3

Вступление

Новые функции, улучшения и изменения от Laravel 5.2 до 5.3

Examples

Переменная $ loop

Известно некоторое время, что обработка циклов в Blade ограничена, по 5.3 существует переменная с названием `$loop` available

```
@foreach($variables as $variable)

    // Within here the `$loop` variable becomes available

    // Current index, 0 based
    $loop->index;

    // Current iteration, 1 based
    $loop->iteration;

    // How many iterations are left for the loop to be complete
    $loop->remaining;

    // Get the amount of items in the loop
    $loop->count;

    // Check to see if it's the first iteration ...
    $loop->first;

    // ... Or last iteration
    $loop->last;

    //Depth of the loop, ie if a loop within a loop the depth would be 2, 1 based counting.
    $loop->depth;

    // Get's the parent `$loop` if the loop is nested, else null
    $loop->parent;

@endforeach
```

Прочитайте Введение в laravel-5.3 онлайн: https://riptutorial.com/ru/laravel/topic/9231/введение-в-laravel-5-3

глава 9: Запрос на перекрестный домен

Examples

Вступление

Иногда нам нужен запрос перекрестного домена для нашего API в laravel. Нам нужно добавить соответствующие заголовки для успешного завершения запроса перекрестного домена. Поэтому мы должны убедиться, что любые заголовки, которые мы добавляем, должны быть точными, иначе наш API станет уязвимым. Чтобы добавить заголовки, нам нужно добавить промежуточное программное обеспечение в laravel, которое добавит соответствующие заголовки и пересылает запросы.

CorsHeaders

```
<?php

namespace laravel\Http\Middleware;

class CorsHeaders
{
  /**
   * This must be executed _before_ the controller action since _after_ middleware isn't
executed when exceptions are thrown and caught by global handlers.
   *
   * @param $request
   * @param \Closure $next
   * @param string [$checkWhitelist] true or false Is a string b/c of the way the arguments
are supplied.
   * @return mixed
   */
  public function handle($request, \Closure $next, $checkWhitelist = 'true')
  {
    if ($checkWhitelist == 'true') {
      // Make sure the request origin domain matches one of ours before sending CORS response
headers.
      $origin = $request->header('Origin');
      $matches = [];
      preg_match('/^(https?:\/\/)?([a-zA-Z\d]+\.)*(?<domain>[a-zA-Z\d-\.]+\.[a-z]{2,10})$/',
$origin, $matches);

      if (isset($matches['domain']) && in_array($matches['domain'], ['yoursite.com']) {
        header('Access-Control-Allow-Origin: ' . $origin);
        header('Access-Control-Expose-Headers: Location');
        header('Access-Control-Allow-Credentials: true');

        // If a preflight request comes then add appropriate headers
        if ($request->method() === 'OPTIONS') {
          header('Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS, DELETE, PATCH');
          header('Access-Control-Allow-Headers: ' . $request->header('Access-Control-Request-
Headers'));
            // 20 days
          header('Access-Control-Max-Age: 1728000');
```

```
        }
      }
    } else {
      header('Access-Control-Allow-Origin: *');
    }

    return $next($request);
  }
}
```

Прочитайте Запрос на перекрестный домен онлайн:
https://riptutorial.com/ru/laravel/topic/7425/запрос-на-перекрестный-домен

глава 10: Запрос формы

Вступление

Пользовательские запросы (или Запросы формы) полезны в ситуациях, когда требуется **авторизировать** и **проверять** запрос перед ударом по методу контроллера.

Можно подумать о двух практических применениях, **создании** и **обновлении** записи, в то время как каждое действие имеет другой набор правил валидации (или авторизации).

Использование запросов формы является тривиальным, нужно набирать тип запроса в методе.

Синтаксис

- php artisan make: request name_of_request

замечания

Запросы полезны при отмене проверки с помощью контроллера. Он также позволяет вам проверить, разрешен ли запрос.

Examples

Создание запросов

```
php artisan make:request StoreUserRequest

php artisan make:request UpdateUserRequest
```

> **Примечание:** Вы можете также рассмотреть вопрос об использовании имен , как **StoreUser** или **UpdateUser** (без **запроса** приложения) , так как ваши FormRequests помещаются в папку `app/Http/Requests/` .

Использование запроса формы

Допустим, продолжайте с примером пользователя (у вас может быть контроллер с методом хранения и методом обновления). Чтобы использовать FormRequests, вы используете тип-намек на конкретный запрос.

```
...

public function store(App\Http\Requests\StoreRequest $request, App\User $user) {
```

```
    //by type-hinting the request class, Laravel "runs" StoreRequest
    //before actual method store is hit

    //logic that handles storing new user
    //(both email and password has to be in $fillable property of User model
    $user->create($request->only(['email', 'password']));
    return redirect()->back();
}

...

public function update(App\Http\Requests\UpdateRequest $request, App\User $users, $id) {
    //by type-hinting the request class, Laravel "runs" UpdateRequest
    //before actual method update is hit

    //logic that handles updating a user
    //(both email and password has to be in $fillable property of User model
    $user = $users->findOrFail($id);
    $user->update($request->only(['password']));
    return redirect()->back();
}
```

Обработка перенаправления после проверки

Иногда вам может понадобиться логин для определения того, куда пользователь перенаправляется после отправки формы. Запросы формы предоставляют различные способы.

По умолчанию в Request `$redirect` , `$redirectRoute` и `$redirectAction` объявлены 3 переменных.

Помимо этих трех переменных вы можете переопределить основной обработчик перенаправления `getRedirectUrl()` .

Ниже приведен пример запроса, поясняющий, что вы можете сделать.

```
<?php namespace App;

use Illuminate\Foundation\Http\FormRequest as Request;

class SampleRequest extends Request {

    // Redirect to the given url
    public $redirect;

    // Redirect to a given route
    public $redirectRoute;

    // Redirect to a given action
    public $redirectAction;


    /**
     * Get the URL to redirect to on a validation error.
     *
     * @return string
     */
```

```
    protected function getRedirectUrl()
    {

        // If no path is given for `url()` it will return a new instance of
`Illuminate\Routing\UrlGenerator`

        // If your form is down the page for example you can redirect to a hash
        return url()->previous() . '#contact';

        //`url()` provides several methods you can chain such as

        // Get the current URL
        return url()->current();

        // Get the full URL of the current request
        return url()->full();

        // Go back
        return url()->previous();

        // Or just redirect back
        return redirect()->back();
    }


    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [];
    }

    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
}
```

Прочитайте Запрос формы онлайн: https://riptutorial.com/ru/laravel/topic/6329/запрос-формы

глава 11: Запросы

Examples

Получение ввода

Основной способ получения ввода - это впрыснуть `Illuminate\Http\Request` в ваш контроллер, после чего существует множество способов доступа к данным, 4 из которых приведены в примере ниже.

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

use Illuminate\Http\Request;

class UserController extends Controller
{
    public function store(Request $request)
    {
        // Returns the username value
        $name = $request->input('username');

        // Returns the username value
        $name = $request->username;

        // Returns the username value
        $name = request('username');

        // Returns the username value again
        $name = request()->username;

    }

}
```

При использовании функции `input` также можно добавить значение по умолчанию, когда вход запроса недоступен

```
$name = $request->input('username', 'John Doe');
```

Прочитайте Запросы онлайн: https://riptutorial.com/ru/laravel/topic/3076/запросы

глава 12: Запросы

Examples

Получить экземпляр HTTP-запроса

Чтобы получить экземпляр HTTP-запроса, класс `Illuminate\Http\Request` должен быть намеком типа либо в конструкторе, либо в методе контроллера.

Пример кода:

```
 <?php

namespace App\Http\Controllers;

/* Here how we illuminate the request class in controller */
use Illuminate\Http\Request;

use Illuminate\Routing\Controller;

class PostController extends Controller
{
 /**
  * Store a new post.
  *
  * @param  Request  $request
  * @return Response
  */
 public function store(Request $request)
 {
     $name = $request->input('post_title');

     /*
     * so typecasting Request class in our method like above avails the
     * HTTP GET/POST/PUT etc method params in the controller to use and
     * manipulate
     */
 }
}
```

Запрос экземпляра с другими параметрами из маршрутов в методе контроллера

Иногда нам нужно принять параметры маршрута, а также получить доступ к параметрам запроса HTTP. Мы все еще можем ввести подсказку класса Requests в контроллер laravel и достичь этого, как описано ниже

Например, у нас есть маршрут, который обновляет определенную запись, подобную этой (передача сообщения id i route)

```
Route::put('post/{id}', 'PostController@update');
```

Также, поскольку пользователь отредактировал другие поля формы редактирования, чтобы они были доступны в HTTP-запросе

Вот как получить доступ как к нашему методу

```
    public function update(Request $request,$id){
        //This way we have $id param from route and $request as an HTTP Request object

    }
```

Прочитайте Запросы онлайн: https://riptutorial.com/ru/laravel/topic/4929/запросы

глава 13: засеивание

замечания

Выделение базы данных позволяет вставлять данные, общие тестовые данные в вашу базу данных. По умолчанию в `DatabaseSeeder` `database/seeds` имеется класс `DatabaseSeeder` .

Запуск сеялок можно выполнить с помощью

```
php artisan db:seed
```

Или, если вы хотите обработать только один класс

```
php artisan db:seed --class=TestSeederClass
```

Как и во всех командах мастеров, вы имеете доступ к широкому спектру методов, которые можно найти в документации api

Examples

Вставка данных

Существует несколько способов вставки данных:

Использование DB Facade

```
public function run()
{
    DB::table('users')
        ->insert([
            'name' => 'Taylor',
            'age'  => 21
        ]);
}
```

Визуализация модели

```
public function run()
{
    $user = new User;
    $user->name = 'Taylor';
    $user->save();
}
```

Использование метода create

```
public function run()
{
    User::create([
        'name' => 'Taylor',
        'age'  => 21
    ]);
}
```

Использование фабрики

```
public function run()
{
    factory(App\User::class, 10)->create();
}
```

Seeding && удаление старых данных и автоматическое добавление ресетов

```
public function run()
{
    DB::table('users')->delete();
    DB::unprepared('ALTER TABLE users AUTO_INCREMENT=1;');
    factory(App\User::class, 200)->create();
}
```

Дополнительную информацию о вставке / обновлении данных см. В примере Persisting .

Вызов других сеялок

В вашем классе `DatabaseSeeder` вы можете вызвать другие сеялки

```
$this->call(TestSeeder::class)
```

Это позволяет сохранить один файл, где вы можете легко найти свои сеялки. Имейте в виду, что вам нужно обратить внимание на порядок ваших вызовов в отношении ограничений внешнего ключа. Вы не можете ссылаться на таблицу, которая еще не существует.

Создание сеялки

Чтобы создать сеялки, вы можете использовать команду `make:seeder` Artisan. Все сгенерированные сеялки будут помещены в каталог `database/seeds` .

```
$ php artisan make:seeder MoviesTableSeeder
```

Созданные сеялки будут содержать один метод: `run` . Вы можете вставить данные в свою базу данных в этот метод.

```
<?php

use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;

class MoviesTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        App\Movie::create([
            'name' => 'A New Hope',
            'year' => '1977'
        ]);

        App\Movie::create([
            'name' => 'The Empire Strikes Back',
            'year' => '1980'
        ]);
    }
}
```

Обычно вы хотите вызвать все ваши сеялки внутри класса `DatabaseSeeder` .

Когда вы закончите писать сеялки, используйте команду `db:seed` . Это `run` функцию запуска `DatabaseSeeder` .

```
$ php artisan db:seed
```

Вы также можете указать, чтобы запустить определенный класс сеялки для индивидуальной работы с `--class` опции `--class` .

```
$ php artisan db:seed --class=UserSeeder
```

Если вы хотите откат и повторить все миграции, а затем повторно:

```
$ php artisan migrate:refresh --seed
```

> Команда `migrate:refresh --seed` является ярлыком для этих 3 команд:
>
> ```
> $ php artisan migrate:reset     # rollback all migrations
> $ php artisan migrate           # run migrations
> $ php artisan db:seed           # run seeders
> ```

Безопасное насыщение

Возможно, вы захотите повторно засеять свою базу данных, не затрагивая ранее созданные семена. Для этой цели вы можете использовать `firstOrCreate` в своей `firstOrCreate` :

```
EmployeeType::firstOrCreate([
    'type' => 'manager',
]);
```

Затем вы можете засеять базу данных:

```
php artisan db:seed
```

Позже, если вы хотите добавить другого сотрудника, вы можете просто добавить его в один и тот же файл:

```
EmployeeType::firstOrCreate([
    'type' => 'manager',
]);
EmployeeType::firstOrCreate([
    'type' => 'secretary',
]);
```

И снова запустите свою базу данных без проблем:

```
php artisan db:seed
```

Обратите внимание, что при первом вызове вы извлекаете запись, но ничего не делаете с ней.

Прочитайте засеивание онлайн: https://riptutorial.com/ru/laravel/topic/3272/засеивание

глава 14: Изменение поведения маршрутизации по умолчанию в Laravel 5.2.31 +

Синтаксис

- public function map (Маршрутизатор $ router) // Определение маршрутов для приложения.
- защищенная функция mapWebRoutes (маршрутизатор $ router) // Определите «веб-маршруты» для приложения.

параметры

параметр	заголовок
Маршрутизатор $ router	\ Illuminate \ Routing \ Router $ router

замечания

Среднее программное обеспечение означает, что каждый вызов маршрута будет проходить через промежуточное программное обеспечение, прежде чем на самом деле нанести ваш маршрут конкретному коду. В Laravel сетевое промежуточное программное обеспечение используется для проверки сеанса или проверки маркера csrf, например.

По умолчанию есть другие middlewares, такие как auth или api. Вы также можете легко создать свое собственное промежуточное программное обеспечение.

Examples

Добавление api-маршрутов с другим промежуточным программным обеспечением и сохранение промежуточного ПО по умолчанию

Начиная с версии 5.2.31 Laravel, промежуточное программное обеспечение Интернета применяется по умолчанию в RouteServiceProvider (https://github.com/laravel/laravel/commit/5c30c98db96459b4cc878d085490e4677b0b67ed)

В приложении / Providers / RouteServiceProvider.php вы найдете следующие функции, которые применяют промежуточное ПО на каждом маршруте в вашем приложении / Http / routes.php

```
public function map(Router $router)
{
    $this->mapWebRoutes($router);
}

// ...

protected function mapWebRoutes(Router $router)
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'web',
    ], function ($router) {
        require app_path('Http/routes.php');
    });
}
```

Как вы видите, применяется **промежуточное** web-приложение. Вы можете изменить это здесь. Тем не менее, вы также можете легко добавить еще одну запись, чтобы можно было разместить ваши маршруты api, например, в другой файл (например, route-api.php)

```
public function map(Router $router)
{
    $this->mapWebRoutes($router);
    $this->mapApiRoutes($router);
}

protected function mapWebRoutes(Router $router)
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'web',
    ], function ($router) {
        require app_path('Http/routes.php');
    });
}

protected function mapApiRoutes(Router $router)
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'api',
    ], function ($router) {
        require app_path('Http/routes-api.php');
    });
}
```

С помощью этого вы можете легко отделить ваши маршруты api от маршрутов вашего приложения без беспорядочной групповой оболочки в пределах ваших маршрутов.php

Прочитайте Изменение поведения маршрутизации по умолчанию в Laravel 5.2.31 + онлайн: https://riptutorial.com/ru/laravel/topic/4285/изменение-поведения-маршрутизации-по-умолчанию-в-laravel-5-2-31-plus

глава 15: Именование файлов при загрузке с помощью Laravel на Windows

параметры

Param / Функция	Описание
файл загружен	имя файла `<input>`
$ sampleName	также может быть динамически созданной строкой или именем файла, загруженного пользователем
APP_PATH ()	является помощником Laravel для обеспечения абсолютного пути к приложению
getCLientOriginalExtension ()	Laravel для получения расширения файла, загруженного пользователем, как это было на машине пользователя

Examples

Создание временных файлов имен файлов для файлов, загружаемых пользователями.

Ниже не будет работать на Windows-машине

```
$file = $request->file('file_upload');
$sampleName = 'UserUpload';
$destination = app_path() . '/myStorage/';
$fileName = $sampleName . '-' . date('Y-m-d-H:i:s') . '.' .
$file->getClientOriginalExtension();
$file->move($destination, $fileName);
```

Он выдает ошибку, например «Невозможно переместить файл в / путь ...»

Зачем? - Это отлично работает на сервере Ubuntu
Причина в том, что в Windows `colon ':'` не разрешено в имени файла, где это разрешено в linux. Это такая маленькая вещь, что мы можем не заметить ее заранее и постоянно задаваться вопросом, почему код, который хорошо работает на Ubuntu (Linux), не работает?
Наша первая догадка состояла бы в том, чтобы проверить права доступа к файлам и тому подобное, но мы не можем заметить, что `colon ':'` является виновником здесь.
Поэтому, чтобы загружать файлы в Windows, **не используйте `colon':'` при создании**

временных файлов имен , вместо этого сделайте следующее:

```
$filename = $sampleName . '-' . date('Y-m-d-H_i_s') . '.' . $file-
>getClientOriginalExtension();  //ex output UserUpload-2016-02-18-11_25_43.xlsx

                                    OR

$filename = $sampleName . '-' .date('Y-m-d H i s') . '.' . $file-
>getClientOriginalExtension();  //ex output UserUpload-2016-02-18 11 25 43.xlsx

                                    OR

$filename = $sampleName . '-'.date('Y-m-d_g-i-A').'.' . $file->getClientOriginalExtension();
//ex output UserUpload-2016-02-18_11-25-AM.xlsx
```

Прочитайте Именование файлов при загрузке с помощью Laravel на Windows онлайн: https://riptutorial.com/ru/laravel/topic/2629/именование-файлов-при-загрузке-с-помощью-laravel-на-windows

глава 16: Инструкция по установке

замечания

В этом разделе представлен обзор того, что такое laravel-5.4, и почему разработчик может захотеть его использовать.

Следует также упомянуть любые крупные предметы в пределах laravel-5.4 и ссылки на связанные темы. Поскольку документация для laravel-5.4 является новой, вам может потребоваться создать начальные версии этих связанных тем.

Examples

Монтаж

Подробные инструкции по настройке или установке laravel.

композитор необходим для установки laravel легко.

В вашей системе есть 3 метода установки laravel:

1. **Установщик Via Laravel**

   Загрузите установщик Laravel с помощью `composer`

   ```
   composer global require "laravel/installer"
   ```

   Перед использованием композитора нам нужно добавить `~/.composer/vendor/bin` в `PATH` . По завершении установки мы можем использовать `laravel new` команду `laravel new` для создания нового проекта в `Laravel` .

   Пример:

   ```
   laravel new {folder name}
   ```

   Эта команда создает новый каталог с именем `site` а в каталоге устанавливается новая установка `Laravel` со всеми другими зависимостями.

2. **Создать проект-композитор**

   Вы можете использовать команду в `terminal` для создания нового `Laravel app` :

   ```
   composer create-project laravel/laravel {folder name}
   ```

3. **Via Download**

   Скачайте Laravel и распакуйте его.

   1. `composer install`
   2. Скопируйте `.env.example` в `.env` через `teminal` или вручную.

      ```
      cp .env.example .env
      ```

   3. Откройте файл `.env` и установите свою базу данных, электронную почту, толкатель и т. Д. (При необходимости)
   4. `php artisan migrate` (если база данных настроена)
   5. `php artisan key:generate`
   6. `php artisan serve`
   7. Перейти на localhost: 8000 для просмотра сайта

Документы Laravel

Пример Hello World (базовый)

Доступ к страницам и вывод данных в Laravel довольно прост. Все маршруты страницы расположены в `app/routes.php` . Обычно вы можете начать несколько примеров, но мы собираемся создать новый маршрут. Откройте `app/routes.php` и вставьте следующий код:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

Это говорит Laravel, что, когда кто-то обращается к `http://localhost/helloworld` в браузере, он должен запустить функцию и вернуть предоставленную строку.

Пример Hello World с представлениями и контроллером

Предполагая, что у нас есть работающее приложение laravel, скажем, «mylaravel.com», мы хотим, чтобы наше приложение показывало сообщение «Hello World», когда мы попали по адресу `http://mylaravel.com/helloworld` . Он включает создание двух файлов (представление и контроллер) и изменение существующего файла, маршрутизатора.

Вид

Во-первых, мы открываем новый файл вида лезвия `helloview.blade.php` со `helloview.blade.php` «Hello World». Создайте его в каталоге app / resources / views

```
<h1>Hello, World</h1>
```

Контроллер

Теперь мы создаем контроллер, который будет управлять отображением этого представления с помощью строки «Hello World». Мы будем использовать artisan в командной строке.

```
$> cd your_laravel_project_root_directory
$> php artisan make:controller HelloController
```

Это просто создаст файл (`app/Http/Controllers/HelloController.php`), содержащий класс, который является нашим новым контроллером `HelloController` .

Отредактируйте этот новый файл и напишите `hello` нового метода, которое покажет представление, которое мы создали ранее.

```
public function hello()
{
    return view('helloview');
}
```

Этот аргумент «helloview» в функции просмотра - это просто имя файла представления без конечного «.blade.php». Ларавель будет знать, как его найти.

Теперь, когда мы вызываем метод `hello` контроллера `HelloController` он отображает сообщение. Но как мы связываем это с призывом к `http://mylaravel.com/helloworld` ? С последним шагом - маршрутизация.

Маршрутизатор

Откройте существующее `app/routes/web.php` (в старых версиях версии laravel `app/Http/routes.php`) и добавьте эту строку:

```
Route::get('/helloworld', 'HelloController@hello');
```

который является очень самоочевидной командой, говорящей в нашем приложении laravel: «Когда кто-то использует `GET` глагол для доступа к / helloworld в этом приложении laravel, верните результаты вызова функции `hello` в контроллере `HelloController` .

Прочитайте Инструкция по установке онлайн: https://riptutorial.com/ru/laravel/topic/2187/инструкция-по-установке

глава 17: Интеграция Sparkpost с Laravel 5.4

Вступление

Laravel 5.4 поставляется с предустановленной app lib. Sparkpost lib требует секретного ключа, который можно найти в их учетной записи sparkpost.

Examples

SAMPLE .env данные файла

Чтобы успешно создать настройку api для электронной почты sparkpost, добавьте приведенные ниже данные в файл env, и ваше приложение будет полезно начать отправку писем.

MAIL_DRIVER = sparkpost
SPARKPOST_SECRET =

ПРИМЕЧАНИЕ. Приведенные выше сведения не дают вам кода, написанного на контроллере, который имеет бизнес-логику для отправки писем с использованием функции laravels Mail :: send.

Прочитайте Интеграция Sparkpost с Laravel 5.4 онлайн:
https://riptutorial.com/ru/laravel/topic/10136/интеграция-sparkpost-с-laravel-5-4

глава 18: использовать псевдонимы полей в Eloquent

Прочитайте использовать псевдонимы полей в Eloquent онлайн:
https://riptutorial.com/ru/laravel/topic/7927/использовать-псевдонимы-полей-в-eloquent

глава 19: камердинер

Вступление

Valet - это среда разработки, разработанная для macOS. Он абстрагирует потребность в виртуальных машинах, Homestead или Vagrant. Не нужно постоянно обновлять файл `/etc/hosts` . Вы даже можете публиковать свои сайты публично, используя локальные туннели.

Laravel Valet делает все сайты доступными в домене `*.dev` , связывая имена папок с именами доменов.

Синтаксис

- команда valet [опции] [аргументы]

параметры

параметр	Набор значений
команда	domain , fetch-share-url, забыть, помогать, устанавливать, связывать , ссылки , список, журналы, на последней версии, открывать, парковать , пуски, перезапускать, защищать, запускать, останавливать, деинсталлировать, отсоединять, небезопасно, что
опции	-h, --help, -q, --quiet, -V, --version, -ansi, -no-ansi, -n, -no-взаимодействие, -v, -vv, -vvv, - -подробный
аргументы	*(необязательный)*

замечания

Поскольку Valet для Linux и Windows является неофициальным, поддержка не будет за пределами их соответствующих репозиториев Github.

Examples

Valet link

Эта команда полезна, если вы хотите обслуживать один сайт в каталоге, а не весь каталог.

```
cd ~/Projects/my-blog/
valet link awesome-blog
```

Valet создаст символическую ссылку в `~/.valet/Sites` которая указывает на ваш текущий рабочий каталог.
После запуска команды ссылки вы можете получить доступ к сайту в своем браузере по адресу `http://awesome-blog.dev` .

Чтобы просмотреть список всех ваших связанных каталогов, запустите команду valet links . Для уничтожения символической ссылки вы можете использовать `valet unlink awesome-blog` .

Валет-парк

```
cd ~/Projects
valet park
```

Эта команда зарегистрирует ваш текущий рабочий каталог как путь, по которому Valet должен искать сайты. Теперь любой проект Laravel, который вы создаете в своем «припаркованном» каталоге, будет автоматически подаваться с использованием соглашения `http://folder-name.dev` .

Валетные ссылки

Эта команда отобразит все зарегистрированные ссылки Valet, которые вы создали, и их соответствующие пути к файлу на вашем компьютере.

Команда:

```
valet links
```

Результат выборки:

```
...
site1 -> /path/to/site/one
site2 -> /path/to/site/two
...
```

Примечание 1: Вы можете запустить эту команду из любой точки не только из связанной папки.

Примечание 2: Сайты будут перечислены без окончания **.dev,** но вы все равно будете использовать **site1.dev** для доступа к вашему приложению из браузера.

Монтаж

***ВАЖНЫЙ!!* Valet - это инструмент, предназначенный только для macOS.**

Предпосылки

- Valet использует HTTP-порт вашей локальной машины (порт 80), поэтому вы не сможете использовать, если *Apache* или *Nginx* установлены и запущены на одном компьютере.
- macOS 'неофициальный менеджер пакетов Homebrew должен правильно использовать Valet.
- Убедитесь, что Homebrew обновлен до последней версии, запустив `brew update` в терминале.

Монтаж

- Установите PHP 7.1 с помощью Homebrew через `brew install homebrew/php/php71` .
- Установить Valet с Composer с помощью `composer global require laravel/valet` .
- Добавьте каталог `~/.composer/vendor/bin` в «PATH» вашей системы, если он еще не существует.
- Запустите команду `valet install` .

Post Install В процессе установки Valet установил *DnsMasq* . Он также зарегистрировал демон Valet для автоматического запуска при запуске вашей системы, поэтому вам не нужно запускать запуск `valet start` или `valet install` каждый раз при перезагрузке компьютера.

Домен Valet

Эта команда позволяет вам изменять или просматривать TLD *(домен верхнего уровня),* используемый для привязки доменов к вашей локальной машине.

Получить текущий TLD

```
$ valet domain
> dev
```

Установите TLD

```
$ valet domain local
> Your Valet domain has been updated to [local].
```

Установка (Linux)

ВАЖНЫЙ!! Valet - это инструмент, предназначенный для macOS, версия ниже портирована для ОС Linux.

Предпосылки

- **Не** устанавливайте valet как `root` или с помощью команды `sudo` .
- Valet использует HTTP-порт вашей локальной машины (порт 80), поэтому вы не сможете использовать, если Apache или Nginx установлены и запущены на одном компьютере.
- Для установки и запуска Valet требуется современная версия `composer` .

Монтаж

- Запустить `composer global require cpriego/valet-linux` для установки Valet по всему миру.
- Запустите команду `valet install` чтобы завершить установку.

Post Install

Во время процесса установки Valet установил DnsMasq. Он также зарегистрировал демон Valet для автоматического запуска при запуске вашей системы, поэтому вам не нужно запускать запуск `valet start` или `valet install` каждый раз при перезагрузке компьютера.

Официальную документацию можно найти здесь .

Прочитайте камердинер онлайн: https://riptutorial.com/ru/laravel/topic/1906/камердинер

глава 20: Касса

замечания

Laravel Cashier можно использовать для подписки, предоставляя интерфейс в сервисах подписки как Braintree, так и Stripe. В дополнение к базовому управлению подпиской он может использоваться для обработки купонов, обмена подписками, количествами, периодами отмены аннулирования и составлением счетов в формате PDF.

Examples

Настройка полосы

Начальная настройка

Чтобы использовать Stripe для обработки платежей, нам нужно добавить следующее в `composer.json` затем запустить `composer update` :

```
"laravel/cashier": "~6.0"
```

Следующая строка должна быть добавлена в `config/app.php` , поставщика услуг:

```
Laravel\Cashier\CashierServiceProvider
```

Настройка Databse

Чтобы использовать кассир, нам нужно настроить базы данных, если таблица пользователей еще не существует, нам нужно создать ее, и нам также нужно создать таблицу подписчиков. Следующий пример изменяет существующую таблицу `users` . См. « Элегантные модели» для получения дополнительной информации о моделях.

Чтобы использовать кассир, создайте новую миграцию и добавьте следующее, которое достигнет вышеуказанного:

```
// Adjust users table

Schema::table('users', function ($table) {
    $table->string('stripe_id')->nullable();
    $table->string('card_brand')->nullable();
    $table->string('card_last_four')->nullable();
    $table->timestamp('trial_ends_at')->nullable();
});

//Create subscriptions table
```

```
Schema::create('subscriptions', function ($table) {
    $table->increments('id');
    $table->integer('user_id');
    $table->string('name');
    $table->string('stripe_id');
    $table->string('stripe_plan');
    $table->integer('quantity');
    $table->timestamp('trial_ends_at')->nullable();
    $table->timestamp('ends_at')->nullable();
    $table->timestamps();
});
```

Затем нам нужно запустить `php artisan migrate` для обновления нашей базы данных.

Настройка модели

Затем мы должны добавить оплачиваемый признак к модели пользователя, найденной в `app/User.php` и изменить ее на следующее:

```
use Laravel\Cashier\Billable;

class User extends Authenticatable
{
    use Billable;
}
```

Клавишные клавиши

Чтобы гарантировать, что мы закончим деньги на нашу собственную учетную запись Stripe, мы должны установить ее в файле `config/services.php` , добавив следующую строку:

```
'stripe' => [
    'model'  => App\User::class,
    'secret' => env('STRIPE_SECRET'),
],
```

Замена `STRIPE_SECRET` секретным ключом вашей собственной полосы.

После завершения этого Кассира и Стрипа настроено так, что вы можете продолжить настройку подписки.

Прочитайте Касса онлайн: https://riptutorial.com/ru/laravel/topic/7474/касса

глава 21: Класс CustomException в Laravel

Вступление

Исключения PHP вызывают, когда происходит беспрецедентное событие или ошибка.

Как правило, исключение не должно использоваться для управления логикой приложения, такой как if-statements, и должно быть подклассом класса Exception.

Одним из основных преимуществ наличия всех исключений, которые вызывают один класс, является то, что мы можем создавать собственные обработчики исключений, которые возвращают разные ответы в зависимости от исключения.

Examples

Класс CustomException в laravel

все ошибки и исключения, как пользовательские, так и по умолчанию, обрабатываются классом Handler в приложении / Исключения / Handler.php с помощью двух методов.

- Отчет ()
- рендеринга ()

```
public function render($request, Exception $e)
{
 //check if exception is an instance of ModelNotFoundException.
 if ($e instanceof ModelNotFoundException)
 {
     // ajax 404 json feedback
     if ($request->ajax())
     {
         return response()->json(['error' => 'Not Found'], 404);
     }
     // normal 404 view page feedback
     return response()->view('errors.missing', [], 404);
  }
   return parent::render($request, $e);
}
```

затем создайте представление, связанное с ошибкой в папке с ошибками 404.blade.php

Пользователь не найден.

Вы нарушили баланс Интернета

Прочитайте Класс CustomException в Laravel онлайн:
https://riptutorial.com/ru/laravel/topic/9550/класс-customexception-в-laravel

глава 22: Коллекции

Синтаксис

- $ collection = collect (['Value1', 'Value2', 'Value3']); // Ключи по умолчанию равны 0, 1, 2, ...,

замечания

`Illuminate\Support\Collection` обеспечивает свободный и удобный интерфейс для работы с массивами данных. Возможно, вы использовали их без знания, например, запросы модели, которые извлекают несколько записей, возвращают экземпляр `Illuminate\Support\Collection` .

Для актуальной документации по Коллекциям вы можете найти официальную документацию здесь

Examples

Создание коллекций

Используя помощник `collect()` , вы можете легко создавать новые экземпляры коллекции, передавая их в массив, такой как:

```
$fruits = collect(['oranges', 'peaches', 'pears']);
```

Если вы не хотите использовать вспомогательные функции, вы можете создать новую коллекцию с помощью класса напрямую:

```
$fruits = new Illuminate\Support\Collection(['oranges', 'peaches', 'pears']);
```

Как упоминалось в примечаниях, модели по умолчанию возвращают экземпляр `Collection` , однако вы можете создавать свои собственные коллекции по мере необходимости. Если для создания не задан массив, будет создана пустая коллекция.

где()

Вы можете выбрать определенные элементы из коллекции, используя метод `where()` .

```
$data = [
    ['name' => 'Taylor',  'coffee_drinker' => true],
    ['name' => 'Matt', 'coffee_drinker' => true]
];
```

```
$matt = collect($data)->where('name', 'Matt');
```

Этот бит кода выберет все элементы из коллекции, где имя «Мэтт». В этом случае возвращается только второй элемент.

гнездование

Точно так же, как и большинство методов массива в Laravel, `where()` поддерживает поиск вложенных элементов. Давайте рассмотрим пример выше, добавив второй массив:

```
$data = [
    ['name' => 'Taylor',  'coffee_drinker' => ['at_work' => true, 'at_home' => true]],
    ['name' => 'Matt', 'coffee_drinker' => ['at_work' => true, 'at_home' => false]]
];

$coffeeDrinkerAtHome = collect($data)->where('coffee_drinker.at_home', true);
```

Это вернет Тейлору, поскольку он выпивает кофе дома. Как вы можете видеть, вложенность поддерживается с помощью точечной нотации.

дополнения

При создании коллекции объектов вместо массивов их можно отфильтровать, используя `where()` . Затем коллекция попытается получить все необходимые свойства.

5,3

Обратите внимание: начиная с Laravel 5.3 метод `where()` будет пытаться свободно сравнивать значения по умолчанию. Это означает, что при поиске `(int)1` будут возвращены все записи, содержащие `'1'` . Если вам не нравится это поведение, вы можете использовать метод `whereStrict()` .

Использование Get to lookup value или возврат по умолчанию

Вы часто оказываетесь в ситуации, когда вам нужно найти переменную, соответствующую стоимости, и коллекции вас охватили.

В приведенном ниже примере мы получили три разных локали в массиве с соответствующим назначенным кодом вызова. Мы хотим иметь возможность предоставить локаль и взамен получить связанный код вызова. Второй параметр в `get` - это параметр по умолчанию, если первый параметр не найден.

```
function lookupCallingCode($locale)
{
```

```
    return collect([
        'de_DE' => 49,
        'en_GB' => 44,
        'en_US' => 1,
    ])->get($locale, 44);
}
```

В приведенном выше примере мы можем сделать следующее

```
lookupCallingCode('de_DE'); // Will return 49
lookupCallingCode('sv_SE'); // Will return 44
```

Вы даже можете передать обратный вызов в качестве значения по умолчанию. Результат обратного вызова будет возвращен, если указанный ключ не существует:

```
    return collect([
        'de_DE' => 49,
        'en_GB' => 44,
        'en_US' => 1,
    ])->get($locale, function() {
        return 44;
    });
```

Использование Содержит, чтобы проверить, удовлетворяет ли коллекция определенному условию

Общей проблемой является сбор элементов, которые все должны соответствовать определенным критериям. В приведенном ниже примере мы собрали два пункта для плана диеты, и мы хотим проверить, что диета не содержит вредных продуктов питания.

```
// First we create a collection
$diet = collect([
    ['name' => 'Banana', 'calories' => '89'],
    ['name' => 'Chocolate', 'calories' => '546']
]);

// Then we check the collection for items with more than 100 calories
$isUnhealthy = $diet->contains(function ($i, $snack) {
    return $snack["calories"] >= 100;
});
```

В приведенном выше случае переменная `$isUnhealthy` будет установлена в `true` поскольку шоколад удовлетворяет условию, и диета, таким образом, нездоровая.

Использование Pluck для извлечения определенных значений из коллекции

Вы часто окажетесь с коллекцией данных, где вас интересуют только части данных.

В приведенном ниже примере мы получили список участников на мероприятии, и мы хотим

предоставить экскурсовод с простым списком имен.

```
// First we collect the participants
$participants = collect([
    ['name' => 'John', 'age' => 55],
    ['name' => 'Melissa', 'age' => 18],
    ['name' => 'Bob', 'age' => 43],
    ['name' => 'Sara', 'age' => 18],
]);

// Then we ask the collection to fetch all the names
$namesList = $partcipants->pluck('name')
// ['John', 'Melissa', 'Bob', 'Sara'];
```

Вы также можете использовать `pluck` для коллекций объектов или вложенных массивов / объектов с точечной нотацией.

```
$users = User::all(); // Returns Eloquent Collection of all users
$usernames = $users->pluck('username'); // Collection contains only user names

$users->load('profile'); // Load a relationship for all models in collection

// Using dot notation, we can traverse nested properties
$names = $users->pluck('profile.first_name'); // Get all first names from all user profiles
```

Использование карты для управления каждым элементом в коллекции

Часто вам нужно изменить способ структурирования набора данных и манипулировать определенными значениями.

В приведенном ниже примере мы получили сборник книг с прилагаемой скидкой. Но у нас есть скорее список книг с ценой, которая уже уценена.

```
$books = [
    ['title' => 'The Pragmatic Programmer', 'price' => 20, 'discount' => 0.5],
    ['title' => 'Continuous Delivery', 'price' => 25, 'discount' => 0.1],
    ['title' => 'The Clean Coder', 'price' => 10, 'discount' => 0.75],
];

$discountedItems =  collect($books)->map(function ($book) {
   return ['title' => $book["title"], 'price' => $book["price"] * $book["discount"]];
});

//[
//    ['title' => 'The Pragmatic Programmer', 'price' => 10],
//    ['title' => 'Continuous Delivery', 'price' => 12.5],
//    ['title' => 'The Clean Coder', 'price' => 5],
//]
```

Это также может быть использовано для изменения ключей, скажем , мы хотели изменить ключевое `title` , чтобы `name` это будет подходящим решением.

Использование суммы, avg, min или max для коллекции для

статистических расчетов

Коллекции также предоставляют вам простой способ сделать простые статистические вычисления.

```
$books = [
    ['title' => 'The Pragmatic Programmer', 'price' => 20],
    ['title' => 'Continuous Delivery', 'price' => 30],
    ['title' => 'The Clean Coder', 'price' => 10],
]

$min = collect($books)->min('price'); // 10
$max = collect($books)->max('price'); // 30
$avg = collect($books)->avg('price'); // 20
$sum = collect($books)->sum('price'); // 60
```

Сортировка коллекции

Существует несколько способов сортировки коллекции.

Сортировать()

Метод `sort` сортирует коллекцию:

```
$collection = collect([5, 3, 1, 2, 4]);

$sorted = $collection->sort();

echo $sorted->values()->all();

returns : [1, 2, 3, 4, 5]
```

Метод `sort` также позволяет передавать пользовательский обратный вызов с помощью вашего собственного алгоритма. Под капотом сортировка использует `usort` php.

```
$collection = $collection->sort(function ($a, $b) {
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
});
```

Сортировать по()

Метод `sortBy` сортирует коллекцию по данному ключу:

```
$collection = collect([
['name' => 'Desk', 'price' => 200],
```

```
['name' => 'Chair', 'price' => 100],
['name' => 'Bookcase', 'price' => 150],
]);

$sorted = $collection->sortBy('price');

echo $sorted->values()->all();

returns:  [
        ['name' => 'Chair', 'price' => 100],
        ['name' => 'Bookcase', 'price' => 150],
        ['name' => 'Desk', 'price' => 200],
    ]
```

Метод `sortBy` позволяет использовать формат точечной нотации для доступа к более глубокому ключу, чтобы сортировать многомерный массив.

```
$collection = collect([
    ["id"=>1,"product"=>['name' => 'Desk', 'price' => 200]],
    ["id"=>2, "product"=>['name' => 'Chair', 'price' => 100]],
    ["id"=>3, "product"=>['name' => 'Bookcase', 'price' => 150]],
    ]);

$sorted = $collection->sortBy("product.price")->toArray();

return: [
  ["id"=>2, "product"=>['name' => 'Chair', 'price' => 100]],
  ["id"=>3, "product"=>['name' => 'Bookcase', 'price' => 150]],
  ["id"=>1,"product"=>['name' => 'Desk', 'price' => 200]],
]
```

SortByDesc ()

Этот метод имеет ту же подпись, что и метод `sortBy` , но сортирует коллекцию в обратном порядке.

Использование reduce ()

Метод `reduce` сводит коллекцию к одному значению, передавая результат каждой итерации в последующую итерацию. См. Метод уменьшения .

Метод `reduce` пересекает каждый элемент с помощью коллекции и создает новый результат для следующей итерации. Каждый результат последней итерации передается через первый параметр (в следующих примерах, как `$carry`).

Этот метод может выполнять большую обработку больших наборов данных. Например, следующие примеры, мы будем использовать следующие примеры данных студента:

```
 $student = [
    ['class' => 'Math', 'score' => 60],
    ['class' => 'English', 'score' => 61],
```

```
    ['class' => 'Chemistry', 'score' => 50],
    ['class' => 'Physics', 'score' => 49],
];
```

Суммарная сумма студента

```
$sum = collect($student)
    ->reduce(function($carry, $item){
        return $carry + $item["score"];
    }, 0);
```

Результат: `220`

Объяснение:

- `$carry` - результат последней итерации.
- Второй параметр - значение по умолчанию для $ carry в первом раунде итерации. В этом случае значение по умолчанию равно 0

Передайте студенту, если все его баллы> 50

```
$isPass = collect($student)
    ->reduce(function($carry, $item){
        return $carry && $item["score"] >= 50;
    }, true);
```

Результат: `false`

Объяснение:

- Значение по умолчанию $ carry равно true
- Если весь балл больше 50, результат вернет true; если они меньше 50, верните false.

Отказ студента, если оценка равна <50

```
$isFail = collect($student)
    ->reduce(function($carry, $item){
        return $carry || $item["score"] < 50;
    }, false);
```

Результат: `true`

Объясните:

- значение по умолчанию $ carry равно false
- если любой балл меньше 50, верните true; если все баллы превышают 50, верните false.

Возвращаемый объект с наивысшей оценкой

```
$highestSubject = collect($student)
    ->reduce(function($carry, $item){
        return $carry === null || $item["score"] > $carry["score"] ? $item : $carry;
    });
```

результат: `[ "subject" => "English", "score" => 61 ]`

Объясните:

- В этом случае второй параметр не предоставляется.
- Значение по умолчанию $ carry равно нулю, поэтому мы проверяем это в нашем условном выражении.

Использование макроса () для расширения коллекций

Функция `macro()` позволяет добавлять новые функции в объекты `Illuminate\Support\Collection`

Использование:

```
Collection::macro("macro_name", function ($parameters) {
    // Your macro
});
```

Например:

```
Collection::macro('uppercase', function () {
    return $this->map(function ($item) {
        return strtoupper($item);
    });
});

collect(["hello", "world"])->uppercase();
```

Результат: `["HELLO", "WORLD"]`

Использование синтаксиса Array

Объект `Collection` реализует интерфейс `ArrayAccess` и `IteratorAggregate` , позволяя использовать его как массив.

Элемент доступа Access:

```
 $collection = collect([1, 2, 3]);
 $result = $collection[1];
```

Результат: `2`

Назначить новый элемент:

```
$collection = collect([1, 2, 3]);
$collection[] = 4;
```

Результат: `$collection [1, 2, 3, 4]`

Коллекция петли:

```
$collection = collect(["a" => "one", "b" => "two"]);
$result = "";
foreach($collection as $key => $value){
    $result .= "(".$key.": ".$value.") ";
}
```

Результат: `$result` is `(a: one) (b: two)`

Преобразование массива в коллекцию:

Чтобы преобразовать коллекцию в собственный PHP-массив, используйте:

```
$array = $collection->all();
//or
$array = $collection->toArray()
```

Чтобы преобразовать массив в коллекцию, используйте:

```
$collection = collect($array);
```

Использование коллекций с функциями массива

Имейте в виду, что коллекции - это обычные объекты, которые не будут правильно преобразованы при использовании функциями, явно требующими массивы, например `array_map($callback)` .

Обязательно сначала конвертируйте коллекцию или, если доступно, используйте метод, предоставляемый классом `Collection` : `$collection->map($callback)`

Прочитайте Коллекции онлайн: https://riptutorial.com/ru/laravel/topic/2358/коллекции

глава 23: Константы

Examples

пример

Сначала вам нужно создать файл constants.php, и это хорошая практика для создания этого файла внутри папки app / config /. Вы также можете добавить файл constants.php в файл compose.json.

Файл примера:

> приложение / Config / constants.php

Константы на основе массива внутри файла:

```
return [
    'CONSTANT' => 'This is my first constant.'
];
```

И вы можете получить эту константу, включив фасад `Config` :

```
use Illuminate\Support\Facades\Config;
```

Затем получите значение по постоянному имени `CONSTANT` как `CONSTANT` ниже:

```
echo Config::get('constants.CONSTANT');
```

И результатом будет значение:

> Это моя первая постоянная.

Прочитайте Константы онлайн: https://riptutorial.com/ru/laravel/topic/9192/константы

глава 24: Контроллеры

Вступление

Вместо того, чтобы определять всю логику обработки запросов как Closures в файлах маршрутов, вы можете организовать это поведение с помощью классов Controller. Контроллеры могут группировать связанную логику обработки запросов в один класс. По умолчанию контроллеры хранятся в каталоге `app/Http/Controllers` .

Examples

Основные контроллеры

```
<?php

namespace App\Http\Controllers;

use App\User;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return Response
     */
    public function show($id)
    {
        return view('user.profile', ['user' => User::findOrFail($id)]);
    }
}
```

Вы можете определить маршрут для этого действия контроллера следующим образом:

```
Route::get('user/{id}', 'UserController@show');
```

Теперь, когда запрос соответствует указанному URI маршрута, будет запущен метод `show` в классе `UserController` . Конечно, параметры маршрута также будут переданы методу.

Контроллер Middleware

Средство промежуточного уровня может быть назначено маршрутам контроллера в файлах маршрутов:

```
Route::get('profile', 'UserController@show')->middleware('auth');
```

Однако удобнее указывать промежуточное ПО в конструкторе вашего контроллера.

Используя метод промежуточного программного обеспечения из конструктора вашего контроллера, вы можете легко назначить промежуточное программное обеспечение для действия контроллера.

```
class UserController extends Controller
{
    /**
     * Instantiate a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');

        $this->middleware('log')->only('index');

        $this->middleware('subscribed')->except('store');
    }
}
```

Контроллер ресурсов

Маршрутизация ресурсов Laravel назначает типичные маршруты «CRUD» контроллеру с одной строкой кода. Например, вы можете создать контроллер, который обрабатывает все HTTP-запросы для «фотографий», хранящихся в вашем приложении. Используя команду `make:controller` Artisan, мы можем быстро создать такой контроллер:

```
php artisan make:controller PhotoController --resource
```

Эта команда будет генерировать контроллер в `app/Http/Controllers/PhotoController.php` . Контроллер будет содержать метод для каждой из доступных операций с ресурсами.

Пример того, как выглядит диспетчер ресурсов

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PhotoController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }
```

```
    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
```

```
    public function destroy($id)
    {
        //
    }
}
```

Пример контроллера ресурсов разделяет имя метода, приведенное в таблице ниже.

Затем вы можете зарегистрировать находчивый маршрут к контроллеру:

```
Route::resource('photos', 'PhotoController');
```

Это одно объявление маршрута создает несколько маршрутов для обработки различных действий на ресурсе. Сгенерированный контроллер уже будет иметь методы, заглушенные для каждого из этих действий, включая заметки, информирующие вас о HTTP-глаголах и URI, которые они обрабатывают.

Действия, управляемые контроллером ресурсов

глагол	URI	действие	Название маршрута
ПОЛУЧИТЬ	`/photos`	индекс	photos.index
ПОЛУЧИТЬ	`/photos/create`	Создайте	photos.create
СООБЩЕНИЕ	`/photos`	хранить	photos.store
ПОЛУЧИТЬ	`/photos/{photo}`	шоу	photos.show
ПОЛУЧИТЬ	`/photos/{photo}/edit`	редактировать	photos.edit
PUT / PATCH	`/photos/{photo}`	Обновить	photos.update
УДАЛЯТЬ	`/photos/{photo}`	уничтожить	photos.destroy

Прочитайте Контроллеры онлайн: https://riptutorial.com/ru/laravel/topic/10604/контроллеры

глава 25: красноречивый

Вступление

The Eloquent - это ORM (реляционная модель объекта), включенная в Laravel. Он реализует активный шаблон записи и используется для взаимодействия с реляционными базами данных.

замечания

Именование таблиц

Соглашение заключается в использовании множественных «snake_case» для имен таблиц и уникальной «StudlyCase» для имен моделей. Например:

- В таблице `cats` будет модель `Cat`
- В таблице `jungle_cats` была бы модель `JungleCat`
- Таблица `users` будет иметь `User` модель
- Таблица `people` будет иметь модель `Person`

«Красноречивый» автоматически попытается привязать вашу модель таблицей с множественным числом имени модели, как указано выше.

Однако вы можете указать имя таблицы, чтобы переопределить соглашение по умолчанию.

```
class User extends Model
{
    protected $table = 'customers';
}
```

Examples

Вступление

Красноречивый - это ORM, встроенный в структуру Laravel. Он позволяет взаимодействовать с вашими таблицами базы данных объектно-ориентированным способом с использованием шаблона ActiveRecord .

Один класс модели обычно сопоставляется с одной таблицей базы данных, а также отношения разных типов (один-к-одному , один-ко-многим , много-ко-многим , полиморфные) могут быть определены между различными классами моделей.

Раздел « Создание модели» описывает создание и определение классов моделей.

Прежде чем вы сможете начать использовать модели Eloquent, убедитесь, что в конфигурационном файле `config/database.php` указано как минимум одно соединение с `config/database.php` .

Чтобы понять использование конструктора запросов красноречивых запросов во время разработки, вы можете использовать команду `php artisan ide-helper:generate` . Вот ссылка .

Подкатегория Навигация

Красноречивые отношения

Упорно

Помимо чтения данных с помощью Eloquent, вы также можете использовать его для вставки или обновления данных с помощью метода `save()` . Если вы создали новый экземпляр модели, запись будет *вставлена* ; в противном случае, если вы извлекли модель из базы данных и установили новые значения, она будет *обновлена* .

В этом примере мы создаем новую запись `User` :

```
$user = new User();
$user->first_name = 'John';
$user->last_name = 'Doe';
$user->email = 'john.doe@example.com';
$user->password = bcrypt('my_password');
$user->save();
```

Вы также можете использовать метод `create` для заполнения полей с помощью массива данных:

```
User::create([
    'first_name'=> 'John',
    'last_name' => 'Doe',
    'email'     => 'john.doe@example.com',
    'password'  => bcrypt('changeme'),
]);
```

При использовании метода create ваши атрибуты должны быть объявлены в `fillable` массиве в вашей модели:

```
class User extends Model
{
    protected $fillable = [
         'first_name',
         'last_name',
         'email',
         'password',
    ];
}
```

В качестве альтернативы, если вы хотите, чтобы все атрибуты были назначены массами, вы можете определить свойство $ guarded как пустой массив:

```
class User extends Model
{
    /**
    * The attributes that aren't mass assignable.
    *
    * @var array
    */
    protected $guarded = [];
}
```

Но вы также можете создать запись даже без изменения `fillable` атрибута в модели, используя `forceCreate` метод , а не `create` метод

```
User::forceCreate([
    'first_name'=> 'John',
    'last_name' => 'Doe',
    'email'     => 'john.doe@example.com',
    'password'  => bcrypt('changeme'),
]);
```

Ниже приведен пример обновления существующего `User` модели по первой загрузке (с помощью `find`), изменяя его, а затем сохранить его:

```
$user = User::find(1);
$user->password = bcrypt('my_new_password');
$user->save();
```

Чтобы выполнить тот же подвиг с помощью одного вызова функции, вы можете использовать метод `update` :

```
$user->update([
    'password' => bcrypt('my_new_password'),
]);
```

Способы `create` и `update` делают работу с большими наборами данных намного проще, чем устанавливать каждую пару ключ / значение отдельно, как показано в следующих примерах:

> Обратите внимание на использование `only` и `except` сбора данных запроса. Важно указать точные ключи, которые вы хотите разрешить / запретить для обновления, в противном случае злоумышленник может отправить дополнительные поля с их запросом и вызвать непреднамеренные обновления.

```
// Updating a user from specific request data
$data = Request::only(['first_name', 'email']);
$user->find(1);
$user->update($data);
```

```
// Create a user from specific request data
$data = Request::except(['_token', 'profile_picture', 'profile_name']);
$user->create($data);
```

Удаление

Вы можете удалить данные после их записи в базу данных. Вы можете либо удалить экземпляр модели, если вы ее получили, либо указать условия для удаления записей.

Чтобы удалить экземпляр модели, извлеките его и вызовите метод `delete()` :

```
$user = User::find(1);
$user->delete();
```

В качестве альтернативы вы можете указать первичный ключ (или массив первичных ключей) записей, которые вы хотите удалить с помощью метода `destroy()` :

```
User::destroy(1);
User::destroy([1, 2, 3]);
```

Вы также можете комбинировать запрос с удалением:

```
User::where('age', '<', 21)->delete();
```

Это приведет к удалению всех пользователей, которые соответствуют условию.

> Примечание: При выполнении массового удаления заявления с помощью красноречивого, то `deleting` и `deleted` моделей событий не будут срабатывать для удаленных моделей. Это связано с тем, что модели никогда не извлекаются при выполнении инструкции delete.

Мягкое удаление

Иногда вы не хотите окончательно удалять запись, но сохраняйте ее для целей аудита или отчетности. Для этого Eloquent предоставляет функции *мягкого удаления* .

Чтобы добавить функциональность мягких удалений в вашу модель, вам нужно импортировать свойство `SoftDeletes` и добавить его в класс модели Eloquent:

```
namespace Illuminate\Database\Eloquent\Model;
namespace Illuminate\Database\Eloquent\SoftDeletes;

class User extends Model
{
     use SoftDeletes;
}
```

При удалении модели она установит `deleted_at` метку в столбце timestamp `deleted_at` в таблице для вашей модели, поэтому `deleted_at` столбец `deleted_at` в своей таблице. Или в процессе миграции вы должны вызвать `softDeletes()` в своем `deleted_at` чтобы добавить `deleted_at` времени `deleted_at` . Пример:

```
Schema::table('users', function ($table) {
    $table->softDeletes();
});
```

В любых запросах будут отсутствовать записи с мягким удалением. Вы можете принудительно показать их, если хотите, используя область `withTrashed()` :

```
User::withTrashed()->get();
```

Если вы хотите разрешить пользователям *восстанавливать* запись после мягкого удаления (т. Е. В области типа корзины), вы можете использовать метод `restore()` :

```
$user = User::find(1);
$user->delete();
$user->restore();
```

Чтобы принудительно удалить запись, используйте метод `forceDelete()` который действительно удалит запись из базы данных:

```
$user = User::find(1);
$user->forceDelete();
```

Изменение первичного ключа и временных меток

По умолчанию модели Eloquent ожидают, что первичный ключ будет называться `'id'` . Если это не ваше дело, вы можете изменить имя вашего первичного ключа, указав свойство `$primaryKey` .

```
class Citizen extends Model
{
    protected $primaryKey = 'socialSecurityNo';

    // ...
}
```

Теперь все методы Eloquent, которые используют ваш первичный ключ (например, `find` или `findOrFail`), будут использовать это новое имя.

Кроме того, Eloquent ожидает, что первичный ключ будет автоматически увеличивающимся целым числом. Если ваш основной ключ не является автоматически увеличивающимся целым числом (например, GUID), вам нужно сообщить Eloquent, обновив свойство `$incrementing` на `false` :

```
class Citizen extends Model
{
    protected $primaryKey = 'socialSecurityNo';

    public $incrementing = false;

    // ...
}
```

По умолчанию Eloquent ожидает, что в ваших таблицах `created_at updated_at` столбцы `created_at` и `updated_at` . Если вы не хотите, чтобы эти столбцы автоматически управлялись Eloquent, установите для свойства `$timestamps` на вашей модели значение false:

```
class Citizen extends Model
{
    public $timestamps = false;

    // ...
}
```

Если вам нужно настроить имена столбцов, используемых для хранения временных меток, вы можете установить `CREATED_AT` и `UPDATED_AT` в своей модели:

```
class Citizen extends Model
{
    const CREATED_AT = 'date_of_creation';
    const UPDATED_AT = 'date_of_last_update';

    // ...
}
```

Выбросить 404, если объект не найден

Если вы хотите автоматически генерировать исключение при поиске записи, которая не найдена в модальном, вы можете использовать либо

```
Vehicle::findOrFail(1);
```

или же

```
Vehicle::where('make', 'ford')->firstOrFail();
```

Если запись с первичным ключом `1` не найдена, `ModelNotFoundException` . Это по сути то же самое, что и запись (источник просмотра):

```
$vehicle = Vehicle::find($id);

if (!$vehicle) {
    abort(404);
}
```

Модели клонирования

Возможно, вам нужно клонировать строку, возможно, изменить несколько атрибутов, но вам нужен эффективный способ держать вещи сухими. Laravel предоставляет своего рода «скрытый» метод, позволяющий вам выполнять эту функцию. Хотя он полностью недокументирован, вам нужно найти через API, чтобы его найти.

Используя `$model->replicate()` вы можете легко клонировать запись

```
$robot = Robot::find(1);
$cloneRobot = $robot->replicate();
// You can add custom attributes here, for example he may want to evolve with an extra arm!
$cloneRobot->arms += 1;
$cloneRobot->save();
```

Вышеупомянутый найдет робота, который имеет идентификатор 1, а затем клонирует его.

Прочитайте красноречивый онлайн: https://riptutorial.com/ru/laravel/topic/865/красноречивый

глава 26: Красноречивый: Аксессоры и мутаторы

Вступление

Аксессоры и мутаторы позволяют вам форматировать значения атрибутов Eloquent при их извлечении или установке в экземплярах модели. Например, вы можете использовать шифр Laravel для шифрования значения, когда он хранится в базе данных, а затем автоматически расшифровывать атрибут при доступе к нему в модели Eloquent. Помимо пользовательских аксессуаров и мутаторов, Eloquent также может автоматически создавать поля даты для экземпляров Carbon или даже отправлять текстовые поля в JSON.

Синтаксис

- set {ATTRIBUTE} Атрибут (атрибут $) // в верблюжьем футляре

Examples

Определение аксессуаров

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the user's first name.
     *
     * @param  string  $value
     * @return string
     */
    public function getFirstNameAttribute($value)
    {
        return ucfirst($value);
    }
}
```

Получение Accessor:

Как вы можете видеть, исходное значение столбца передается в accessor, что позволяет

вам манипулировать и возвращать значение. Чтобы получить доступ к значению аксессора, вы можете просто получить доступ к атрибуту `first_name` в экземпляре модели:

```
$user = App\User::find(1);
$firstName = $user->first_name;
```

Определение мутатора

```
class User extends Model
{
    public function setPasswordAttribute($password)
    {
        $this->attributes['password'] = bcrypt($password);
    }
    ...
}
```

Над кодом «bcrypting» каждый раз, когда установлен пароль.

```
$user = $users->first();
$user->password = 'white rabbit'; //laravel calls mutator on background
$user->save(); // password is bcrypted and one does not need to call bcrypt('white rabbit')
```

Прочитайте Красноречивый: Аксессоры и мутаторы онлайн:
https://riptutorial.com/ru/laravel/topic/8305/красноречивый--аксессоры-и-мутаторы

глава 27: Красноречивый: Модель

Examples

Создание модели

Создание модели

Модельные классы должны расширять `Illuminate\Database\Eloquent\Model` . По умолчанию для моделей используется каталог `/app` .

Класс модели может быть легко сгенерирован командой Artisan :

```
php artisan make:model [ModelName]
```

Это создаст новый файл PHP в `app/` по умолчанию, который называется `[ModelName].php` и будет содержать все шаблоны для вашей новой модели, включая класс, пространство имен и использование, необходимые для базовой настройки.

Если вы хотите создать файл миграции вместе с вашей моделью, используйте следующую команду, где `-m` также будет генерировать файл миграции:

```
php artisan make:model [ModelName] -m
```

В дополнение к созданию модели это создает миграцию базы данных, которая подключается к модели. Файл PHP миграции базы данных по умолчанию находится в `database/migrations/` . Это не по умолчанию - включает в себя все, кроме столбцов id и `created_at` / `updated_at` , поэтому вам нужно будет отредактировать файл, чтобы предоставить дополнительные столбцы.

Обратите внимание, что вам потребуется выполнить миграцию (после того, как вы настроили файл миграции), чтобы модель начала работать, используя `php artisan migrate` из корня проекта

Кроме того, если вы хотите добавить переход позже, после создания модели вы можете сделать это, запустив:

```
php artisan make:migration [migration name]
```

Скажем, например, вы хотели создать модель для своих кошек, у вас было бы два варианта, чтобы создать с миграцией или без нее. Вы бы решили создать без миграции, если у вас уже была таблица кошек или вы не хотели ее создавать в это время.

В этом примере мы хотим создать миграцию, потому что у нас еще нет таблицы, поэтому будет выполняться следующая команда.

```
php artisan make:model Cat -m
```

Эта команда создаст два файла:

1. В папке «Приложение»: `app/Cat.php`
2. В папке `database/migrations/timestamp_creat_cats_table.php` данных: `database/migrations/timestamp_creat_cats_table.php`

Файл, который нам интересен, является последним, так как именно этот файл мы можем решить, к чему мы хотим, чтобы таблица выглядела и включалась. Для любой предопределенной миграции нам присваивается столбец с индексом auto incrementing id и столбцы timestamps.

Нижеприведенный пример извлечения файла миграции включает в себя вышеуказанные предопределенные столбцы, а также добавление имени кота, возраста и цвета:

```
public function up()
    {
        Schema::create('cats', function (Blueprint $table) {

            $table->increments('id');     //Predefined ID
            $table->string('name');       //Name
            $table->integer('age');       //Age
            $table->string('colour');     //Colour
            $table->timestamps();         //Predefined Timestamps

        });
    }
```

Таким образом, вы можете легко создать модель и выполнить миграцию для таблицы. Затем, чтобы выполнить миграцию и создать ее в своей базе данных, вы выполните следующую команду:

```
php artisan migrate
```

Которая перенесет любые выдающиеся миграции в вашу базу данных.

Расположение файла модели

Модели можно хранить в любом месте благодаря PSR4 .

По умолчанию модели создаются в каталоге `app` с пространством имен `App` . Для более сложных приложений обычно рекомендуется хранить модели в своих собственных папках в структуре, которая имеет смысл для архитектуры ваших приложений.

Например, если у вас было приложение, в котором использовалась серия фруктов в качестве моделей, вы могли бы создать папку под названием `app/Fruits` и в этой папке вы создадите `Banana.php` (поддерживая соглашение об именах StudlyCase), тогда вы можете создать класс Banana в пространстве имен `App\Fruits` :

```
namespace App\Fruits;

use Illuminate\Database\Eloquent\Model;

class Banana extends Model {
    // Implementation of "Banana" omitted
}
```

Конфигурация модели

«Красноречивый» следует за «соглашением по конфигурации». Расширяя базовый класс `Model` , все модели наследуют перечисленные ниже свойства. Если не переопределены, применяются следующие значения по умолчанию:

Имущество	Описание	По умолчанию
`protected $connection`	Имя соединения БД	Соединение по умолчанию DB
`protected $table`	Название таблицы	По умолчанию имя класса преобразуется в `snake_case` и pluralized. Например, `SpecialPerson` становится `special_people`
`protected $primaryKey`	Таблица PK	`id`
`public $incrementing`	Указывает, будут ли идентификаторы автоматически увеличиваться	`true`
`public $timestamps`	Указывает, должна ли модель быть временной отметкой	`true`
`const CREATED_AT`	Имя столбца временной метки создания	`created_at`
`const UPDATED_AT`	Имя столбца временной отметки модификации	`updated_at`
`protected $dates`	Атрибуты, которые должны быть изменены на DateTime, в дополнение к атрибутам	`[]`

Имущество	Описание	По умолчанию
	timestamps	
`protected $dateFormat`	Формат, в котором атрибуты даты сохраняются	По умолчанию для текущего диалекта SQL.
`protected $with`	Отношения к eagerload с моделью	`[]`
`protected $hidden`	Атрибуты опущены в сериализации модели	`[]`
`protected $visible`	Атрибуты, разрешенные в сериализации модели	`[]`
`protected $appends`	Атрибуты атрибутов добавлены в сериализацию модели	`[]`
`protected $fillable`	Атрибуты, которые могут быть распределены по массе	`[]`
`protected $guarded`	Атрибуты, которые перечислены в черном списке из массового присвоения	`[*]` (Все атрибуты)
`protected $touches`	Отношения, которые следует затронуть	`[]`
`protected $perPage`	Количество моделей, возвращаемых для разбивки на страницы.	`15`

5.0

Имущество	Описание	По умолчанию
`protected $casts`	Атрибуты, которые должны быть отнесены к родным типам	`[]`

Обновление существующей модели

```
$user = User::find(1);
$user->name = 'abc';
$user->save();
```

Вы также можете обновлять сразу несколько атрибутов, используя `update` , которое не

требует использования `save` впоследствии:

```
$user = User::find(1);
$user->update(['name' => 'abc', 'location' => 'xyz']);
```

Вы также можете обновить модель (ы) без предварительного запроса:

```
User::where('id', '>', 2)->update(['location' => 'xyz']);
```

Если вы не хотите запускать изменение в timestamp `updated_at` на модели, вы можете передать `touch` параметр:

```
$user = User::find(1);
$user->update(['name' => 'abc', 'location' => 'xyz'], ['touch' => false]);
```

Прочитайте Красноречивый: Модель онлайн: https://riptutorial.com/ru/laravel/topic/7984/красноречивый--модель

глава 28: Красноречивый: Отношения

Examples

Запрос на отношения

Eloquent также позволяет вам запрашивать определенные отношения, как показано ниже:

```
User::whereHas('articles', function (Builder $query) {
    $query->where('published', '!=', true);
})->get();
```

Это требует, чтобы в этом случае ваше имя метода взаимодействия - это `articles` . Аргумент, переданный в закрытие, представляет собой Query Builder для связанной модели, поэтому вы можете использовать любые запросы, которые вы можете найти в другом месте.

Веселая загрузка

Предположим, что модель пользователя имеет отношение к модели Article, и вы хотите, чтобы она загружала связанные статьи. Это означает, что статьи пользователя будут загружаться при извлечении пользователя.

`articles` - это имя отношения (метод) в модели пользователя.

```
User::with('articles')->get();
```

если у вас много отношений. например, статей и сообщений.

```
User::with('articles','posts')->get();
```

и выбрать вложенные отношения

```
User::with('posts.comments')->get();
```

Вызов более чем одной вложенной связи

```
User::with('posts.comments.likes')->get()
```

Вставка похожих моделей

Предположим, у вас есть модель `Post` с отношениями `hasMany` с `Comment` . Вы можете вставить объект `Comment` связанный с сообщением, выполнив следующие действия:

```
$post = Post::find(1);

$commentToAdd = new Comment(['message' => 'This is a comment.']);

$post->comments()->save($commentToAdd);
```

Вы можете сохранить сразу несколько моделей с помощью функции `saveMany` :

```
$post = Post::find(1);

$post->comments()->saveMany([
    new Comment(['message' => 'This a new comment']),
    new Comment(['message' => 'Me too!']),
    new Comment(['message' => 'Eloquent is awesome!'])
]);
```

Кроме того, существует также метод `create` который принимает простой PHP-массив вместо экземпляра модели Eloquent.

```
$post = Post::find(1);

$post->comments()->create([
    'message' => 'This is a new comment message'
]);
```

Вступление

Яркие отношения определяются как функции в классах модели Eloquent. Поскольку, как и сами модели Eloquent, отношения также служат мощными строителями запросов, определяя отношения как функции, обеспечивая мощные возможности цепочки и запросов. Например, мы можем связать дополнительные ограничения в отношении этих сообщений:

```
$user->posts()->where('active', 1)->get();
```

Перейти к родительской теме

Типы отношений

Один ко многим

Допустим, что каждый пост может иметь один или несколько комментариев, и каждый комментарий принадлежит только одному сообщению.

поэтому в таблице комментариев будет `post_id` . В этом случае отношения будут следующими.

Почтовая модель

```
public function comments()
{
   return $this->belongsTo(Post::class);
}
```

Если внешний ключ отличается от `post_id` , например, внешний ключ - `example_post_id` .

```
public function comments()
{
   return $this->belongsTo(Post::class, 'example_post_id');
}
```

и плюс, если локальный ключ отличается от `id` , например, локальным ключом является `other_id`

```
public function comments()
{
   return $this->belongsTo(Post::class, 'example_post_id', 'other_id');
}
```

Модель комментария

определение обратного от одного к многим

```
public function post()
{
   return $this->hasMany(Comment::class);
}
```

Один к одному

Как установить связь между двумя моделями (пример: модель `User` и `Phone`)

`App\User`

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the phone record associated with the user.
     */
    public function phone()
    {
        return $this->hasOne('Phone::class', 'foreign_key', 'local_key');
    }
```

```
}
```

`App\Phone`

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Phone extends Model
{
    /**
     * Get the user that owns the phone.
     */
    public function user()
    {
        return $this->belongsTo('User::class', 'foreign_key', 'local_key');
    }
}
```

`foreign_key` : По умолчанию Eloquent примет это значение как `other_model_name_id` (в этом случае `user_id` и `phone_id`), измените его, если это не так.

`local_key` : По умолчанию Eloquent будет считать это значение `id` (текущий первичный ключ модели), измените его, если это не так.

> Если ваша база данных указала имя по стандарту laravel, вам не нужно предоставлять внешний ключ и локальный ключ в объявлении отношений

объяснение

Многим многим

Допустим, есть роли и разрешения. Каждая роль может принадлежать многим разрешениям, и каждое разрешение может принадлежать ко многим ролям. поэтому будет 3 таблицы. две модели и одна сводная таблица. `roles` , `users` и таблица `permission_role` .

Ролевая модель

```
public function permissions()
{
   return $this->belongsToMany(Permission::class);
}
```

Модель разрешения

```
public function roles()
{
   return $this->belongsToMany(Roles::class);
}
```

Примечание: 1

рассмотрите следующее, используя другое имя таблицы для сводной таблицы.

Предположим, если вы хотите использовать `role_permission` вместо `permission_role` , поскольку красноречивый использует буквенный порядок для создания имен ключевых ключей. вам нужно будет передать имя сводной таблицы в качестве второго параметра следующим образом.

Ролевая модель

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission');
}
```

Модель разрешения

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission');
}
```

Заметка 2

рассмотрите следующие при использовании разных имен ключей в сводной таблице.

Красноречивый предполагает, что если никакие ключи не передаются третьим и четвертым параметрами, это будут уникальные имена таблиц с `_id` . поэтому он предполагает, что в pivot будут поля `role_id` и `permission_id` . Если ключи, отличные от них, должны использоваться, они должны быть переданы как третий, так и четвертый параметры.

Допустим , если `other_role_id` вместо `role_id` и `other_permission_id` вместо `permission_id` будет использоваться. Так было бы так.

Ролевая модель

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id');
}
```

Модель разрешения

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission', 'other_permission_id',
'other_role_id');
```

```
}
```

полиморфный

Полиморфные отношения позволяют модели относиться к более чем одной другой модели в одной ассоциации. Хорошим примером могут быть изображения, как пользователь, так и продукт могут иметь изображение. Структура таблицы может выглядеть следующим образом:

```
user
    id - integer
    name - string
    email - string

product
    id - integer
    title - string
    SKU - string

image
    id - integer
    url - string
    imageable_id - integer
    imageable_type - string
```

Важные столбцы для просмотра находятся в таблице изображений. `imageable_id` будет содержать значение идентификатора пользователя или продукта, в то время `imageable_type` столбец `imageable_type` будет содержать имя класса модели владельца. В ваших моделях вы устанавливаете отношения следующим образом:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Image extends Model
{
    /**
     * Get all of the owning imageable models.
     */
    public function imageable()
    {
        return $this->morphTo();
    }
}

class User extends Model
{
    /**
     * Get all of the user's images.
     */
    public function images()
    {
```

```
        return $this->morphMany('Image::class', 'imageable');
    }
}

class Product extends Model
{
    /**
     * Get all of the product's images.
     */
    public function images()
    {
        return $this->morphMany('Image::class', 'imageable');
    }
}
```

Вы также можете получить владельца полиморфного отношения из полиморфной модели, обратившись к имени метода, который выполняет вызов `morphTo` . В нашем случае это `imageable` метод на модели изображения. Таким образом, мы получим доступ к этому методу как динамическое свойство

```
$image = App\Image::find(1);

$imageable = $image->imageable;
```

Этот `imageable` будет возвращать либо Пользователь, либо Продукт.

Много для многих

Допустим, есть роли и разрешения. Каждая роль может принадлежать многим разрешениям, и каждое разрешение может принадлежать ко многим ролям. поэтому будет 3 таблицы. две модели и одна сводная таблица. `roles` , `users` и таблица `permission_role` .

Ролевая модель

```
public function permissions()
{
   return $this->belongsToMany(Permission::class);
}
```

Модель разрешения

```
public function roles()
{
   return $this->belongsToMany(Roles::class);
}
```

Примечание: 1

рассмотрите следующее, используя другое имя таблицы для сводной таблицы.

Предположим, если вы хотите использовать `role_permission` вместо `permission_role` ,

поскольку красноречивый использует буквенный порядок для создания имен ключевых ключей. вам нужно будет передать имя сводной таблицы в качестве второго параметра следующим образом.

Ролевая модель

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission');
}
```

Модель разрешения

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission');
}
```

Заметка 2

рассмотрите следующие при использовании разных имен ключей в сводной таблице.

Красноречивый предполагает, что если никакие ключи не передаются третьим и четвертым параметрами, это будут уникальные имена таблиц с `_id` . поэтому он предполагает, что в pivot будут поля `role_id` и `permission_id` . Если ключи, отличные от них, должны использоваться, они должны быть переданы как третий, так и четвертый параметры.

Допустим , если `other_role_id` вместо `role_id` и `other_permission_id` вместо `permission_id` будет использоваться. Так было бы так.

Ролевая модель

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id');
}
```

Модель разрешения

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission', 'other_permission_id',
'other_role_id');
}
```

Доступ к промежуточной таблице с помощью команды pivot ()

Предположим, у вас есть третий столбец « **permission_assigned_date** » в сводной

таблице. По умолчанию на сводный объект будут присутствовать только ключи модели. Теперь, чтобы получить этот столбец в результате запроса, вам нужно добавить имя в функцию withPivot ().

```
   public function permissions()
        {
           return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id')->withPivot('permission_assigned_date');
        }
```

Прикрепление / отсоединение

Eloquent также предоставляет несколько дополнительных вспомогательных методов, чтобы сделать работу с родственными моделями более удобной. Например, давайте представим, что у пользователя может быть много ролей, а роль может иметь много разрешений. Чтобы прикрепить роль к разрешению, вставив запись в промежуточную таблицу, которая присоединяется к моделям, используйте метод attach:

```
$role= App\Role::find(1);
$role->permissions()->attach($permissionId);
```

При присоединении отношения к модели вы также можете передать массив дополнительных данных, которые необходимо вставить в промежуточную таблицу:

```
$rol->roles()->attach($permissionId, ['permission_assigned_date' => $date]);
```

Аналогично, Чтобы удалить конкретное разрешение на роль, используйте функцию отладки

```
$role= App\Role::find(1);
//will remove permission 1,2,3 against role 1
$role->permissions()->detach([1, 2, 3]);
```

Синхронизация ассоциаций

Вы также можете использовать метод sync для построения ассоциаций «многие-ко-многим». Метод синхронизации принимает массив идентификаторов для размещения на промежуточной таблице. Любые идентификаторы, которые не находятся в данном массиве, будут удалены из промежуточной таблицы. Таким образом, после завершения этой операции в промежуточной таблице будут существовать только идентификаторы в данном массиве:

```
//will keep permission id's 1,2,3 against Role id 1

$role= App\Role::find(1)
$role->permissions()->sync([1, 2, 3]);
```

Прочитайте Красноречивый: Отношения онлайн: https://riptutorial.com/ru/laravel/topic/7960/красноречивый--отношения

глава 29: Макросы в красноречивых отношениях

Вступление

У нас есть новые возможности для Eloquent Relationship в Laravel версии 5.4.8. Мы можем получить один экземпляр отношения hasMany (это всего лишь один пример), определив его на месте и он будет работать для всех отношений

Examples

Мы можем получить один экземпляр отношений hasMany

В нашем AppServiceProvider.php

```
public function boot()
{
  HasMany::macro('toHasOne', function() {
      return new HasOne(
          $this->query,
          $this->parent,
          $this->foreignKey,
          $this->localKey
      );
  });
}
```

Предположим, что у нас есть магазин модальный, и мы получаем список приобретенных продуктов. Предположим, что у нас есть всеприобретенные отношения для Shop modal

```
public function allPurchased()
{
    return $this->hasMany(Purchased::class);
}

public function lastPurchased()
{
    return $this->allPurchased()->latest()->toHasOne();
}
```

Прочитайте Макросы в красноречивых отношениях онлайн:
https://riptutorial.com/ru/laravel/topic/8998/макросы-в-красноречивых-отношениях

глава 30: маршрутизация

Examples

Основная маршрутизация

Маршрутизация определяет карту между методами HTTP и URI с одной стороны, а действия - с другой. Маршруты обычно записываются в файле `app/Http/routes.php` .

В своей простейшей форме маршрут определяется путем вызова соответствующего метода HTTP на фасаде маршрута, передавая в качестве параметров строку, которая соответствует URI (относительно корня приложения), и обратный вызов.

Например: маршрут к корневому URI сайта , который возвращает вид `home` выглядит следующим образом :

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

Маршрут для почтового запроса, который просто перекликается с пост-переменными:

```
Route::post('submit', function() {
   return Input::all();
});

//or

Route::post('submit', function(\Illuminate\Http\Request $request) {
   return $request->all();
});
```

Маршруты, указывающие на метод контроллера

Вместо определения встроенного обратного вызова маршрут может ссылаться на метод контроллера в синтаксисе [ControllerClassName @ Method]:

```
Route::get('login', 'LoginController@index');
```

Маршрут для нескольких глаголов

Метод `match` может использоваться для сопоставления массива методов HTTP для заданного маршрута:

```
Route::match(['GET', 'POST'], '/', 'LoginController@index');
```

Также вы можете использовать `all` чтобы соответствовать любому методу HTTP для заданного маршрута:

```
Route::all('login', 'LoginController@index');
```

Группы маршрутов

Маршруты можно сгруппировать, чтобы избежать повторения кода.

Предположим, что все URI с префиксом `/admin` используют определенное промежуточное программное обеспечение, называемое `admin` и все они живут в пространстве имен `App\Http\Controllers\Admin` .

Чистый способ представления этого с использованием групп маршрутов выглядит следующим образом:

```
Route::group([
    'namespace' => 'Admin',
    'middleware' => 'admin',
    'prefix' => 'admin'
    ], function () {

        // something.dev/admin
        // 'App\Http\Controllers\Admin\IndexController'
        // Uses admin middleware
        Route::get('/', ['uses' => 'IndexController@index']);

        // something.dev/admin/logs
        // 'App\Http\Controllers\Admin\LogsController'
        // Uses admin middleware
        Route::get('/logs', ['uses' => 'LogsController@index']);
});
```

Именованный маршрут

Именованные маршруты используются для создания URL-адреса или перенаправления на определенный маршрут. Преимущество использования именованного маршрута состоит в том, что если мы изменим URI маршрута в будущем, нам не нужно будет изменять URL-адрес или перенаправления, указывающие на этот маршрут, если мы используем именованный маршрут. Но если ссылки были сгенерированы с использованием URL-адреса [например. `url('/admin/login')`], тогда нам придется менять везде, где он используется.

Именованные маршруты создаются с использованием в `as` ключа массива

```
Route::get('login', ['as' => 'loginPage', 'uses' => 'LoginController@index']);
```

или используя `name` метода

```
Route::get('login', 'LoginController@index')->name('loginPage');
```

Создание URL-адреса с использованием именованного маршрута

Чтобы создать URL-адрес, используя имя маршрута

```
$url = route('loginPage');
```

Если вы используете имя маршрута для перенаправления

```
$redirect = Redirect::route('loginPage');
```

Параметры маршрута

Вы можете использовать параметры маршрута, чтобы получить часть сегмента URI. Вы можете определить необязательный или требуемый параметр / s маршрута при создании маршрута. Необязательные параметры имеют `?` добавляется в конце имени параметра. Это имя заключено в фигурные скобки `{}`

Необязательный параметр

```
Route::get('profile/{id?}', ['as' => 'viewProfile', 'uses' => 'ProfileController@view']);
```

К этому маршруту можно обращаться по `domain.com/profile/23` где 23 - это параметр id. В этом примере `id` передается в качестве параметра в `view` метода `ProfileController` . Поскольку это необязательный параметр, доступ к `domain.com/profile` работает просто отлично.

Требуемый параметр

```
Route::get('profile/{id}', ['as' => 'viewProfile', 'uses' => 'ProfileController@view']);
```

Обратите внимание, что имя требуемого параметра не имеет `?` в конце имени параметра.

Доступ к параметру в контроллере

В вашем контроллере ваш метод представления принимает параметр с тем **же** именем, что и в `routes.php` и может использоваться как обычная переменная. Laravel заботится о том, чтобы вводить значение:

```
public function view($id){
    echo $id;
}
```

Поймать все маршруты

Если вы хотите поймать все маршруты, вы можете использовать регулярное выражение, как показано:

```
Route::any('{catchall}', 'CatchAllController@handle')->where('catchall', '.*');
```

Важно: если у вас есть другие маршруты, и вы не хотите, чтобы все это мешало, вы должны положить его в конец. Например:

Захват всех маршрутов, кроме уже определенных

```
Route::get('login',  'AuthController@login');
Route::get('logout', 'AuthController@logout');
Route::get('home',   'HomeController@home');

// The catch-all will match anything except the previous defined routes.
Route::any('{catchall}', 'CatchAllController@handle')->where('catchall', '.*');
```

Маршруты сопоставляются в том порядке, в котором они объявлены

Это обычная гора с маршрутами Laravel. Маршруты сопоставляются в том порядке, в котором они объявлены. Первый соответствующий маршрут - тот, который используется.

Этот пример будет работать, как ожидалось:

```
Route::get('/posts/{postId}/comments/{commentId}', 'CommentController@show');
Route::get('/posts/{postId}', 'PostController@show');
```

Запрос на получение `/posts/1/comments/1` вызовет `CommentController@show` . Запрос на получение `/posts/1` будет вызывать `PostController@show` .

Однако этот пример не будет работать таким же образом:

```
Route::get('/posts/{postId}', 'PostController@show');
Route::get('/posts/{postId}/comments/{commentId}', 'CommentController@show');
```

Запрос на получение `/posts/1/comments/1` вызовет `PostController@show` . Запрос на получение `/posts/1` будет вызывать `PostController@show` .

Поскольку Laravel использует первый согласованный маршрут, запрос `/posts/1/comments/1` соответствует `Route::get('/posts/{postId}', 'PostController@show');` и присваивает переменной `$postId` значение `1/comments/1` . Это означает, что `CommentController@show` никогда не будет вызываться.

Нечувствительные к регистру маршруты

Маршруты в Laravel чувствительны к регистру. Это означает, что такой маршрут, как

```
Route::get('login', ...);
```

будет соответствовать запросу GET для `/login` но не будет соответствовать запросу GET в `/Login` .

Чтобы ваши маршруты были нечувствительны к регистру, вам необходимо создать новый класс проверки, который будет соответствовать запрошенным URL-адресам в отношении определенных маршрутов. Единственное различие между новым валидатором и существующим заключается в том, что он добавит модификатор **i** в конце регулярного выражения для скомпилированного маршрута для переключения разрешить нечувствительность к регистру.

```
<?php namespace Some\Namespace;

use Illuminate\Http\Request;
use Illuminate\Routing\Route;
use Illuminate\Routing\Matching\ValidatorInterface;

class CaseInsensitiveUriValidator implements ValidatorInterface
{
  public function matches(Route $route, Request $request)
  {
    $path = $request->path() == '/' ? '/' : '/'.$request->path();
    return preg_match(preg_replace('/$/','i', $route->getCompiled()->getRegex()),
rawurldecode($path));
  }
}
```

Чтобы Laravel использовал ваш новый валидатор, вам необходимо обновить список совпадений, которые используются для сопоставления URL-адреса маршруту и заменить оригинальный UriValidator на ваш.

Чтобы сделать это, добавьте следующее в начало файла routes.php:

```
<?php
use Illuminate\Routing\Route as IlluminateRoute;
use Your\Namespace\CaseInsensitiveUriValidator;
use Illuminate\Routing\Matching\UriValidator;

$validators = IlluminateRoute::getValidators();
$validators[] = new CaseInsensitiveUriValidator;
IlluminateRoute::$validators = array_filter($validators, function($validator) {
  return get_class($validator) != UriValidator::class;
});
```

Это удалит исходный валидатор и добавит ваш список в список валидаторов.

Прочитайте маршрутизация онлайн: https://riptutorial.com/ru/laravel/topic/1284/маршрутизация

глава 31: Миграции баз данных

Examples

Миграции

Для управления вашей базой данных в Laravel используется миграция. Создание миграции с помощью мастера:

```
php artisan make:migration create_first_table --create=first_table
```

Это создаст класс CreateFirstTable. Внутри метода вверх вы можете создать свои столбцы:

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateFirstTable extends Migration
{
    public function up()
    {
        Schema::create('first_table', function (Blueprint $table) {
            $table->increments('id');
            $table->string('first_string_column_name');
            $table->integer('secont_integer_column_name');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('first_table');
    }
}
```

В конце, чтобы запустить все ваши классы миграции, вы можете запустить команду artisan:

```
php artisan migrate
```

Это создаст ваши таблицы и столбцы в вашей базе данных. Другая полезная команда:

- `php artisan migrate:rollback` - `php artisan migrate:rollback` последней миграции базы данных
- `php artisan migrate:reset` - Откат всех миграций баз данных
- `php artisan migrate:refresh` - Сбросить и повторно запустить все миграции
- `php artisan migrate:status` - показать статус каждой миграции

Изменение существующих таблиц

Иногда вам необходимо изменить существующую структуру таблиц, например, `renaming/deleting` столбцов. Что вы можете достичь путем создания нового migration.And В `up` метода миграции.

```
//Renaming Column.

public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->renameColumn('email', 'username');
    });
}
```

В приведенном выше примере переименуйте `email column users table` в `username` . В то время как приведенный ниже код удаляет `username users` столбца из таблицы `users` .

IMPROTANT: для изменения столбцов вам нужно добавить зависимость `doctrine/dbal` к файлу `composer.json` проекта и запустить `composer update` чтобы отразить изменения.

```
//Droping Column
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('username');
    });
}
```

Файлы миграции

Миграции в приложении Laravel 5 находятся в каталоге `database/migrations` . Их имена файлов соответствуют определенному формату:

```
<year>_<month>_<day>_<hour><minute><second>_<name>.php
```

Один файл миграции должен представлять собой обновление схемы для решения конкретной проблемы. Например:

```
2016_07_21_134310_add_last_logged_in_to_users_table.php
```

Переходы базы данных хранятся в хронологическом порядке, чтобы Laravel знал, в каком порядке их выполнять. Laravel всегда будет выполнять миграции от самых старых до новейших.

Создание файлов миграции

Создание нового файла миграции с правильным именем файла каждый раз, когда вам нужно изменить схему, было бы сложной задачей. К счастью, команда `artisan` Laravel может произвести миграцию для вас:

```
php artisan make:migration add_last_logged_in_to_users_table
```

Вы также можете использовать флаги `--table` и `--create` с приведенной выше командой. Они являются необязательными и только для удобства, и вставьте соответствующий код шаблона в файл миграции.

```
php artisan make:migration add_last_logged_in_to_users_table --table=users

php artisan make:migration create_logs_table --create=logs
```

Вы можете указать настраиваемый путь вывода для сгенерированной миграции с `--path` опции `--path` . Путь относится к базовому пути приложения.

```
php artisan make:migration --path=app/Modules/User/Migrations
```

Внутри миграции базы данных

Каждая миграция должна иметь метод `up()` метод `down()` . Цель метода `up()` - выполнить требуемые операции, чтобы поместить схему базы данных в ее новое состояние, а цель метода `down()` - отменить любые операции, выполняемые методом `up()` . Убедиться, что метод `down()` корректно меняет ваши операции, имеет решающее значение для возможности изменения схемы отката базы данных.

Пример файла миграции может выглядеть следующим образом:

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class AddLastLoggedInToUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dateTime('last_logged_in')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('users', function (Blueprint $table) {
```

```
            $table->dropColumn('last_logged_in');
        });
    }
}
```

При выполнении этой миграции Laravel создаст следующий SQL для работы с вашей базой данных:

```
ALTER TABLE `users` ADD `last_logged_in` DATETIME NULL
```

Выполнение миграций

Как только ваша миграция будет записана, ее запуск будет применяться к вашей базе данных.

```
php artisan migrate
```

Если все пойдет хорошо, вы увидите результат, похожий на приведенный ниже:

```
Migrated: 2016_07_21_134310_add_last_logged_in_to_users_table
```

Laravel достаточно умен, чтобы знать, когда вы выполняете миграцию в рабочей среде. Если он обнаруживает, что вы выполняете деструктивную миграцию (например, тот, который удаляет столбец из таблицы), команда `php artisan migrate` попросит вас подтвердить. В условиях непрерывной доставки это может не потребоваться. В этом случае используйте флаг `--force` чтобы пропустить подтверждение:

```
php artisan migrate --force
```

Если вы только запускаете миграцию, вы можете быть смущены, чтобы увидеть наличие таблицы `migrations` в вашей базе данных. Эта таблица - это то, что Laravel использует для отслеживания того, какие миграции уже выполнены. При выдаче команды `migrate` Laravel определит, какие миграции еще предстоит выполнить, а затем выполнить их в хронологическом порядке, а затем обновить таблицу `migrations` .

Вы никогда не должны вручную редактировать таблицу `migrations` если не знаете, что делаете. Очень легко непреднамеренно оставить вашу базу данных в неисправном состоянии, когда ваши миграции не удастся.

Перемещение назад

Что делать, если вы хотите отменить последнюю миграцию, т.е. недавнюю операцию, вы можете использовать команду awesome `rollback` . Но помните, что эта команда откатывает только последнюю миграцию, которая может включать в себя несколько файлов миграции

```
php artisan migrate:rollback
```

Если вы заинтересованы в откате всех миграций приложений, вы можете использовать следующую команду

```
php artisan migrate:reset
```

Более того, если вы ленитесь, как я, и хотите откат и переход с помощью одной команды, вы можете использовать эту команду

```
php artisan migrate:refresh
php artisan migrate:refresh --seed
```

Вы также можете указать количество шагов для отката с `step` . Как это будет откат 1 шаг.

```
php artisan migrate:rollback --step=1
```

Прочитайте Миграции баз данных онлайн: https://riptutorial.com/ru/laravel/topic/1131/миграции-баз-данных

глава 32: Монтаж

Examples

Монтаж

Приложения Laravel устанавливаются и управляются с помощью Composer , популярного менеджера зависимостей PHP. Существует два способа создания нового приложения Laravel.

Via Composer

```
$ composer create-project laravel/laravel [foldername]
```

Или же

```
$ composer create-project --prefer-dist laravel/laravel [foldername]
```

Замените **[имя_папки]** на имя каталога, в который вы хотите установить новое приложение Laravel. Он не должен существовать до установки. Вам также может потребоваться добавить исполняемый файл Composer к вашему системному пути.

Если вы хотите создать проект Laravel с использованием определенной версии фреймворка, вы можете указать шаблон версии, иначе ваш проект будет использовать последнюю доступную версию.

Например, если вы хотите создать проект в Laravel 5.2, вы должны запустить:

```
$ composer create-project --prefer-dist laravel/laravel 5.2.*
```

Почему -prefer-dist

Существует два способа загрузки пакета: `source` и `dist` . Для стабильных версий Composer будет использовать `dist` по умолчанию. `source` является репозиторий управления версиями. Если `--prefer-source` включен, Composer будет устанавливать из источника, если он есть.

`--prefer-dist` - противоположность `--prefer-source` , и говорит Composer, чтобы установить с `dist` если это возможно. Это может ускорить установку на серверах сборки и в других случаях, когда вы обычно не запускаете обновления поставщиков. Это также позволяет избежать проблем с Git, если у вас нет надлежащей настройки.

С помощью установщика Laravel

Laravel предоставляет полезную утилиту командной строки для быстрого создания приложений Laravel. Сначала установите установщик:

```
$ composer global require laravel/installer
```

> Вы должны убедиться, что папка двоичных файлов Composer находится в переменной $ PATH для выполнения установщика Laravel.
>
> Во-первых, посмотрите, уже ли это в переменной $ PATH
>
> `echo $PATH`
>
> Если все правильно, вывод должен содержать следующее:
>
> `Users/yourusername/.composer/vendor/bin`
>
> Если нет, отредактируйте свой `.bashrc` или, если вы используете ZSH, ваш `.zshrc` чтобы он содержал путь к каталогу поставщика Composer.

После установки эта команда создаст новую установку Laravel в указанном вами каталоге.

```
laravel new [foldername]
```

Вы также можете использовать . (точка) вместо **[имя_папки]** для создания проекта в текущем рабочем каталоге без создания подкаталога.

Запуск приложения

Laravel поставляется в комплекте с веб-сервером на основе PHP, который можно запустить с помощью

```
$ php artisan serve
```

По умолчанию HTTP-сервер будет использовать порт 8000, но если порт уже используется или если вы хотите запускать сразу несколько приложений Laravel, вы можете использовать флаг `--port` для указания другого порта:

```
$ php artisan serve --port=8080
```

HTTP-сервер будет использовать `localhost` в качестве домена по умолчанию для запуска приложения, но вы можете использовать флаг `--host` для указания другого адреса:

```
$ php artisan serve --host=192.168.0.100 --port=8080
```

Использование другого сервера

Если вы предпочитаете использовать другое программное обеспечение веб-сервера, некоторые файлы конфигурации предоставляются вам в `public` каталоге вашего проекта; `.htaccess` для Apache и `web.config` для ASP.NET. Для других программ, таких как NGINX, вы можете конвертировать конфигурации Apache с помощью различных онлайн-инструментов.

Рамочной основе требуется, чтобы пользователь веб-сервера имел права на запись в следующих каталогах:

- `/storage`
- `/bootstrap/cache`

В операционных системах * nix это может быть достигнуто путем

```
chown -R www-data:www-data storage bootstrap/cache
chmod -R ug+rwx storage bootstrap/cache
```

(где `www-data` - это имя и группа пользователей веб-сервера)

Веб-сервер по вашему выбору должен быть настроен для обслуживания контента из каталога вашего проекта `/public` доступа, что обычно делается путем установки его в качестве корня документа. Остальная часть вашего проекта не должна быть доступна через ваш веб-сервер.

Если вы настроите все правильно, для навигации по URL вашего сайта должна отображаться целевая страница по умолчанию Laravel.

Требования

Рамка Laravel имеет следующие требования:

5,3

- PHP> = 5.6.4
- Расширение XML PHP
- Расширение PDO PHP
- Расширение PHP OpenSSL
- Расширение PHP Mbstring
- Расширение Tokenizer PHP

5.1 (LTS) 5.2

- PHP> = 5.5.9
- Расширение PDO PHP
- Laravel 5.1 - первая версия Laravel для поддержки PHP 7.0.

5.0

- PHP> = 5.4, PHP <7
- Расширение PHP OpenSSL
- Расширение Tokenizer PHP
- Расширение PHP Mbstring
- Расширение JSON PHP (только на PHP 5.5)

4,2

- PHP> = 5.4
- Расширение PHP Mbstring
- Расширение JSON PHP (только на PHP 5.5)

Пример Hello World (использование контроллера и представления)

1. Создайте приложение Laravel:

```
$ composer create-project laravel/laravel hello-world
```

2. Перейдите в папку проекта, например

```
$ cd C:\xampp\htdocs\hello-world
```

3. Создать контроллер:

```
$ php artisan make:controller HelloController --resource
```

Это создаст файловое **приложение** / **Http / Controllers / HelloController.php** . Опция `--resource` будет генерировать CRUD-методы для контроллера, например, индексировать, создавать, показывать, обновлять.

4. Зарегистрируйте маршрут к методу `index` HelloController. Добавьте эту строку в **app / Http / routes.php** *(версия 5.0 до 5.2)* или **route / web.php** *(версия 5.3)* :

```
Route::get('hello', 'HelloController@index');
```

Чтобы просмотреть новые добавленные маршруты, вы можете запустить `$ php artisan route:list`

5. Создание шаблона клинка в `views` каталоге:

ресурсы / **просмотров** / **hello.blade.php:**

```
<h1>Hello world!</h1>
```

6. Теперь мы **указываем** индексный метод для отображения шаблона **hello.blade.php** :

приложение / Http / Контроллеры / HelloController.php

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

class HelloController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        return view('hello');
    }

    // ... other resources are listed below the index one above
```

Вы можете использовать свое приложение, используя следующую команду PHP Artisan: **`php artisan serve`** ; он покажет вам адрес, по которому вы можете получить доступ к своему приложению *(обычно по адресу http: // localhost: 8000 по умолчанию)* .

Кроме того, вы можете перейти непосредственно в соответствующее место в своем браузере; *в случае, если вы используете такой сервер, как XAMPP (либо: http: // localhost / hello-world / public / hello, если вы установили свой экземпляр Laravel,* **`hello-world`** *, прямо в ваш* ***каталог xampp / htdocs,*** *как в: выполнив шаг 1 этого слова Hello из вашего интерфейса командной строки, указывая на ваш* ***каталог xampp / htdocs*** *)* .

Пример Hello World (базовый)

Открыть файл маршрутов. Вставьте следующий код в:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

после перехода на маршрут `localhost/helloworld` отображает **Hello World** .

Файл маршрутов находится:

5,3

Для Web

```
routes/web.php
```

Для API

```
routes/api.php
```

5.2 5.1 (LTS) 5.0

```
app/Http/routes.php
```

4,2

```
app/routes.php
```

Установка с использованием LaraDock (Laravel Homestead for Docker)

LaraDock - это усадьба Laravel, такая как среда разработки, но для Docker вместо Vagrant.
https://github.com/LaraDock/laradock

Монтаж

* Требуется Git и Docker

Клонировать хранилище LaraDock:

A. Если у вас уже есть проект Laravel, клонируйте этот репозиторий в корневой каталог Laravel:

```
git submodule add https://github.com/LaraDock/laradock.git
```

B. Если у вас нет проекта Laravel, и вы хотите установить Laravel из Docker, клонируйте это репо где угодно на вашем компьютере:

```
git clone https://github.com/LaraDock/laradock.git
```

Основное использование

1. Run Containers: (Перед запуском команд docker-compose убедитесь, что вы находитесь в папке laradock).

   Пример: Запуск NGINX и MySQL: `docker-compose up -d nginx mysql`

   Существует список доступных контейнеров, которые вы можете выбрать для создания своих собственных комбинаций.

   `nginx` , `hhvm` , `php-fpm` , `mysql` , `redis` , `postgres` , `mariadb` , `neo4j` , `mongo` , `apache2` , `caddy` ,

```
memcached , beanstalkd , beanstalkd-console , workspace
```

2. Войдите в контейнер рабочей области, чтобы выполнять команды типа (Artisan, Composer, PHPUnit, Gulp, ...).

```
docker-compose exec workspace bash
```

3. Если у вас еще не установлен проект Laravel, следуйте инструкциям по установке Laravel из контейнера Docker.

   а. Войдите в контейнер рабочей области.

   б. Установите Laravel. `composer create-project laravel/laravel my-cool-app "5.3.*"`

4. Измените настройки Laravel. Откройте файл .env Laravel и установите DB_HOST в ваш mysql:

```
DB_HOST=mysql
```

5. Откройте браузер и перейдите на адрес localhost.

Прочитайте Монтаж онлайн: https://riptutorial.com/ru/laravel/topic/7961/монтаж

глава 33: наблюдатель

Examples

Создание наблюдателя

Наблюдатели используются для прослушивания обратных вызовов жизненного цикла определенной модели в Laravel.
Эти слушатели могут прослушивать одно из следующих действий:

- создание
- созданный
- обновление
- обновленный
- экономия
- сохранены
- удаление
- удаленный
- восстановление
- восстановлен

Вот пример наблюдателя.

UserObserver

```
<?php

namespace App\Observers;

/**
 * Observes the Users model
 */
class UserObserver
{
    /**
     * Function will be triggerd when a user is updated
     *
     * @param Users $model
     */
     public function updated($model)
     {
         // execute your own code
     }
}
```

Как показано в пользовательском наблюдателе, мы прислушиваемся к обновленному действию, однако, прежде чем этот класс фактически прослушает модель пользователя, нам сначала нужно зарегистрировать его внутри `EventServiceProvider` .

EventServiceProvider

```
<?php

namespace App\Providers;

use Illuminate\Contracts\Events\Dispatcher as DispatcherContract;
use Illuminate\Foundation\Support\Providers\EventServiceProvider as ServiceProvider;

use App\Models\Users;
use App\Observers\UserObserver;

/**
 * Event service provider class
 */
class EventServiceProvider extends ServiceProvider
{
    /**
     * Boot function
     *
     * @param DispatcherContract $events
     */
    public function boot(DispatcherContract $events)
    {
        parent::boot($events);

        // In this case we have a User model that we want to observe
        // We tell Laravel that the observer for the user model is the UserObserver
        Users::observe(new UserObserver());
    }
}
```

Теперь, когда мы зарегистрировали нашего наблюдателя, обновленная функция будет вызываться каждый раз после сохранения модели пользователя.

Прочитайте наблюдатель онлайн: https://riptutorial.com/ru/laravel/topic/7128/наблюдатель

глава 34: Начало работы с laravel-5.3

замечания

В этом разделе представлен обзор того, что такое laravel-5.3, и почему разработчик может захотеть его использовать.

Следует также упомянуть любые крупные предметы в рамках laravel-5.3 и ссылки на связанные темы. Поскольку документация для laravel-5.3 является новой, вам может потребоваться создать начальные версии этих связанных тем.

Examples

Установка Laravel

Требования:
Вам нужен `PHP >= 5.6.4` и `Composer` установленный на вашем компьютере. Вы можете проверить версию как с помощью команды:
Для PHP:

```
php -v
```

Вывод выглядит следующим образом:

```
PHP 7.0.9 (cli) (built: Aug 26 2016 06:17:04) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

Для композитора
Вы можете запустить команду на вашем терминале / CMD:

```
composer --version
```

Вывод выглядит следующим образом:

```
composer version 1.2.1 2016-09-12 11:27:19
```

Laravel использует Composer для управления своими зависимостями. Поэтому, прежде чем использовать Laravel, убедитесь, что на вашем компьютере установлен Composer.

Установщик Via Laravel

Сначала загрузите установщик Laravel с помощью Composer:

```
composer global require "laravel/installer"
```

Обязательно поместите `$HOME/.composer/vendor/bin` (или эквивалентный каталог для вашей ОС) в ваш $ PATH, чтобы исполняемый файл `laravel` мог быть расположен вашей системой.

После установки `laravel new` команда `laravel new` создаст новую установку Laravel в указанном вами каталоге. Например, `laravel new blog` создаст каталог с `blog` содержащий новую установку Laravel со всеми уже установленными зависимостями Laravel:

```
laravel new blog
```

Создать проект-композитор

Кроме того, вы также можете установить Laravel, выпустив команду `create-project` Composer в своем терминале:

```
composer create-project --prefer-dist laravel/laravel blog
```

Настроить

После того, как вы закончите установку Laravel, вам нужно будет установить `permissions` для папок хранения и Bootstrap.

> **Примечание.** Установка `permissions` является одним из наиболее важных процессов для завершения установки Laravel.

Сервер локального развития

Если у вас установлен PHP локально , и вы хотели бы использовать встроенный сервер разработки PHP, чтобы служить приложение, вы можете использовать `serve` Artisan команды. Эта команда запустит сервер разработки по адресу `http://localhost:8000` :

```
php artisan serve
```

Откройте URL-адрес вашего браузера URL `http://localhost:8000`

Требования к серверу

Рамка Laravel имеет несколько системных требований. Конечно, все эти требования удовлетворяются виртуальной машиной Laravel Homestead , поэтому настоятельно рекомендуется использовать Homestead в качестве локальной среды разработки Laravel.

Однако, если вы не используете Homestead, вам необходимо убедиться, что ваш сервер отвечает следующим требованиям:

- PHP> = 5.6.4
- Расширение PHP OpenSSL
- Расширение PDO PHP
- Расширение PHP Mbstring
- Расширение Tokenizer PHP
- Расширение XML PHP

Сервер локального развития

Если у вас установлен PHP локально , и вы хотели бы использовать встроенный сервер разработки PHP, чтобы служить приложение, вы можете использовать `serve` Artisan команды. Эта команда запустит сервер разработки по адресу `http://localhost:8000` :

```
php artisan serve
```

Разумеется, более надежные локальные варианты разработки доступны через Homestead и Valet .

Также можно использовать собственный порт, например `8080` . Вы можете сделать это с помощью опции `--port` .

```
php artisan serve --port=8080
```

Если у вас есть локальный домен в вашем файле hosts, вы можете установить имя хоста. Это можно сделать с помощью опции `--host` .

```
php artisan serve --host=example.dev
```

Вы также можете запускать собственный хост и порт, это можно сделать с помощью следующей команды.

```
php artisan serve --host=example.dev --port=8080
```

Hello World Example (Basic) и с использованием вида

Основной пример
Откройте файл `routes/web.php` и вставьте следующий код в файл:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

здесь « **helloworld** » будет действовать как имя страницы, к которой вы хотите получить доступ,

и если вы не хотите создавать файл клика и хотите получить доступ к странице напрямую, вы можете использовать маршрутизацию laravel таким образом

теперь введите `localhost/helloworld` в адресную строку браузера, и вы можете получить доступ к странице, отображающей Hello World.

Следующий шаг.
Итак, вы научились создавать очень простой Hello World! , вернув приветственное предложение мира. Но мы можем сделать это немного лучше!

Шаг 1.
Мы снова начнем в наших `routes/web.php` файле `routes/web.php` вместо использования вышеприведенного кода, мы будем использовать следующий код:

```
Route::get('helloworld', function() {
    return view('helloworld');
});
```

Возвращаемое значение на этот раз - это не просто простой текст helloworld, а вид. Вид в Laravel - это просто новый файл. Этот файл «helloworld» содержит HTML и, возможно, позже даже некоторый PHP текста Helloworld.

Шаг 2.
Теперь, когда мы скорректировали наш маршрут, чтобы вызвать представление, мы собираемся сделать представление. Laravel работает с файлами blade.php в представлениях. Итак, в этом случае наш маршрут называется helloworld. Поэтому наш взгляд будет называться `helloworld.blade.php`

Мы создадим новый файл в каталоге `resources/views` и назовем его helloworld.blade.php

Теперь мы откроем этот новый файл и отредактируем его, создав предложение Hello World. Мы можем добавить несколько разных способов получить наше предложение, как в приведенном ниже примере.

```
<html>
    <body>
        <h1> Hello World! </h1>

        <?php
            echo "Hello PHP World!";
        ?>

    </body>
</html>
```

теперь перейдите в свой браузер и снова введите свой маршрут, как в базовом примере: `localhost/helloworld` вы увидите новое созданное представление со всем содержимым!

Пример Hello World (базовый)

Открыть файл маршрутов. Вставьте следующий код в:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

после перехода на маршрут `http://localhost/helloworld` он отображает Hello World.

Файл маршрутов расположен `/routes/web.php`

Конфигурация веб-сервера для довольных URL-адресов

Если вы установили `Laravel` через `Composer or the Laravel installer` , вам понадобится настройка ниже.

Конфигурация для Apache Laravel включает файл `public/.htaccess` который используется для предоставления URL-адресов без переднего контроллера `index.php` в пути. Прежде чем отправлять Laravel с Apache, обязательно включите модуль `mod_rewrite` чтобы файл `.htaccess` был удостоен сервера.

Если файл `.htaccess` который поставляется с Laravel, не работает с вашей установкой Apache, попробуйте эту альтернативу:

```
Options +FollowSymLinks
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]
```

Конфигурация для Nginx Если вы используете Nginx, следующая директива в конфигурации вашего сайта будет направлять все запросы на передний контроллер `index.php` :

```
location / {
    try_files $uri $uri/ /index.php?$query_string;
}
```

Конечно, при использовании Homestead или Valet будут автоматически настроены красивые URL-адреса.

Прочитайте Начало работы с laravel-5.3 онлайн: https://riptutorial.com/ru/laravel/topic/8602/начало-работы-с-laravel-5-3

глава 35: Несколько соединений DB в Laravel

Examples

Начальные шаги

В файле конфигурации базы данных можно определить несколько соединений с базами данных любого типа (вероятно, `app/config/database.php`). Например, чтобы вытащить данные из 2 баз данных MySQL, определите их как отдельно:

```
<?php
return array(

    'default' => 'mysql',

    'connections' => array(

        # Our primary database connection
        'mysql' => array(
            'driver'    => 'mysql',
            'host'      => 'host1',
            'database'  => 'database1',
            'username'  => 'user1',
            'password'  => 'pass1'
            'charset'   => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix'    => '',
        ),

        # Our secondary database connection
        'mysql2' => array(
            'driver'    => 'mysql',
            'host'      => 'host2',
            'database'  => 'database2',
            'username'  => 'user2',
            'password'  => 'pass2'
            'charset'   => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix'    => '',
        ),
    ),
);
```

По умолчанию соединение по-прежнему установлено в `mysql` . Это означает, что если не указано иное, приложение использует соединение `mysql` .

Использование Schema builder

Внутри построителя схем используйте фасад схемы с любым подключением. Запустите

метод `connection()` чтобы указать, какое соединение использовать:

```
Schema::connection('mysql2')->create('some_table', function($table)
{
    $table->increments('id');
});
```

Использование построителя запросов БД

Подобно Schema Builder, определите соединение в Query Builder:

```
$users = DB::connection('mysql2')->select(...);
```

Использование эвкалипта

Существует несколько способов определить, какое соединение использовать в моделях Eloquent. Один из способов - установить переменную $ connection в модели:

```
<?php

class SomeModel extends Eloquent {

    protected $connection = 'mysql2';

}
```

Соединение также может быть определено во время выполнения с помощью метода `setConnection` .

```
<?php

class SomeController extends BaseController {

    public function someMethod()
    {
        $someModel = new SomeModel;

        $someModel->setConnection('mysql2');

        $something = $someModel->find(1);

        return $something;
    }
}
```

Из документации Laravel

Доступ к каждому отдельному соединению возможен через метод соединения на фасаде `DB` , даже если определено несколько подключений. `name` переданное методу `connection` должно соответствовать одному из соединений, перечисленных в конфигурационном

файле `config/database.php` :

```
$users = DB::connection('foo')->select(...);
```

Необработанный может быть также доступен, базовый экземпляр PDO, используя метод getPdo для экземпляра соединения:

```
$pdo = DB::connection()->getPdo();
```

https://laravel.com/docs/5.4/database#using-multiple-database-connections

Прочитайте Несколько соединений DB в Laravel онлайн:
https://riptutorial.com/ru/laravel/topic/9605/несколько-соединений-db-в-laravel

глава 36: Обработка ошибок

замечания

Не забудьте настроить приложение для отправки по электронной почте, `config/mail.php` правильную конфигурацию `config/mail.php`

Также проверьте, правильно ли установлены переменные ENV.

Этот пример является ориентиром и минимален. Изучите, измените и настройте представление по своему усмотрению. Измените код, чтобы он соответствовал вашим потребностям. Например, установите получателя в файл *.env*

Examples

Отправить сообщение об ошибке

Исключения в Laravel обрабатываются *App \ Exceptions \ Handler.php*

По умолчанию этот файл содержит две функции. Отчет и рендеринг. Мы будем использовать только первый

```
public function report(Exception $e)
```

> Метод report используется для регистрации исключений или отправки их во внешнюю службу, например BugSnag. По умолчанию метод отчета просто передает исключение базовому классу, в котором регистрируется исключение. Однако вы можете регистрировать исключения, как пожелаете.

По сути эта функция просто пересылает ошибку и ничего не делает. Поэтому мы можем вставить бизнес-логику для выполнения операций на основе ошибки. В этом примере мы отправим электронное письмо с информацией об ошибке.

```
public function report(Exception $e)
{
    if ($e instanceof \Exception) {
        // Fetch the error information we would like to
        // send to the view for emailing
        $error['file']    = $e->getFile();
        $error['code']    = $e->getCode();
        $error['line']    = $e->getLine();
        $error['message'] = $e->getMessage();
        $error['trace']   = $e->getTrace();

        // Only send email reports on production server
        if(ENV('APP_ENV') == "production"){
            #1. Queue email for sending on "exceptions_emails" queue
```

```
            #2. Use the emails.exception_notif view shown below
            #3. Pass the error array to the view as variable $e
            Mail::queueOn('exception_emails', 'emails.exception_notif', ["e" => $error],
function ($m) {
                $m->subject("Laravel Error");
                $m->from(ENV("MAIL_FROM"), ENV("MAIL_NAME"));
                $m->to("webmaster@laravelapp.com", "Webmaster");
            });

        }
    }

    // Pass the error on to continue processing
    return parent::report($e);
}
```

Представление для электронной почты ("emails.exception_notif") ниже

```
<?php
$action = (\Route::getCurrentRoute()) ? \Route::getCurrentRoute()->getActionName() : "n/a";
$path = (\Route::getCurrentRoute()) ? \Route::getCurrentRoute()->getPath() : "n/a";
$user = (\Auth::check()) ? \Auth::user()->name : 'no login';
?>

There was an error in your Laravel App<br />

<hr />
<table border="1" width="100%">
    <tr><th >User:</th><td>{{ $user }}</td></tr>
    <tr><th >Message:</th><td>{{ $e['message'] }}</td></tr>
    <tr><th >Action:</th><td>{{ $action }}</td></tr>
    <tr><th >URI:</th><td>{{ $path }}</td></tr>
    <tr><th >Line:</th><td>{{ $e['line'] }}</td></tr>
    <tr><th >Code:</th><td>{{ $e['code'] }}</td></tr>
</table>
```

Широкое охват приложения ModelNotFoundException

Приложение \ Исключения \ handler.php

```
public function render($request, Exception $exception)
{
    if ($exception instanceof ModelNotFoundException) {
        abort(404);
    }

    return parent::render($request, $exception);
}
```

Вы можете поймать / обработать любое исключение, которое бросается в Laravel.

Прочитайте Обработка ошибок онлайн: https://riptutorial.com/ru/laravel/topic/2858/обработка-ошибок

глава 37: Общие проблемы и быстрые исправления

Вступление

В этом разделе перечислены общие проблемы и быстрые решения разработчиков (особенно начинающих).

Examples

Исключение TokenMisMatch

Вы получаете это исключение, главным образом, с помощью форм. Laravel защищает приложение от `CSRF` и проверяет каждый запрос и гарантирует, что запрос возник из приложения. Эта проверка выполняется с использованием `token` . Если этот токен не соответствует этому исключению.

Быстрая починка

Добавьте это в свой элемент формы. Это отправляет `csrf_token` сгенерированный laravel вместе с другими данными формы, поэтому laravel знает, что ваш запрос действителен

```
<input type="hidden" name="_token" value="{{ csrf_token() }}">
```

Прочитайте Общие проблемы и быстрые исправления онлайн: https://riptutorial.com/ru/laravel/topic/9971/общие-проблемы-и-быстрые-исправления

глава 38: Основы Cron

Вступление

Cron - это демон планировщика задач, выполняющий запланированные задачи через определенные промежутки времени. Cron использует файл конфигурации crontab, также известный как cron table, для управления процессом планирования.

Examples

Создать работу Cron

Crontab содержит задания cron, каждый из которых связан с конкретной задачей. Задачи Cron состоят из двух частей: выражения cron и команды оболочки для запуска:

```
* * * * * команда / to / run
```

Каждое поле в приведенном выше выражении `* * * * *` является опцией для установки частоты расписания. Он состоит из минуты, часа, месяца месяца, месяца и дня недели в порядке размещения. Символ звездочки относится ко всем возможным значениям для соответствующего поля. В результате вышеуказанное задание cron будет выполняться каждую минуту в день.

Следующее задание cron выполняется в **12:30** каждый день:

```
30 12 * * * команда / to / run
```

Прочитайте Основы Cron онлайн: https://riptutorial.com/ru/laravel/topic/9891/основы-cron

глава 39: Очереди

Вступление

Очереди позволяют вашему приложению резервировать бит работы, требующий много времени для обработки фоновым процессом.

Examples

Случаи применения

Например, если вы отправляете электронное письмо клиенту после запуска задачи, лучше сразу перенаправить пользователя на следующую страницу во время очередности отправки электронной почты в фоновом режиме. Это ускорит время загрузки для следующей страницы, так как отправка электронной почты может занять несколько секунд или дольше.

Другим примером может быть обновление системы инвентаризации после того, как клиент проверит их заказ. Вместо того, чтобы ждать завершения вызовов API, что может занять несколько секунд, вы можете сразу перенаправить пользователя на страницу успешной проверки, в то время как очереди вызовов API выполняются в фоновом режиме.

Конфигурация драйвера очереди

Каждый из драйверов очереди Laravel настроен из файла `config/queue.php` . Драйвер очереди - это обработчик для управления запуском задания в очереди, определение того, были ли задания успешными или неудачными, и повторить попытку задания, если оно настроено для этого.

Из коробки Laravel поддерживает следующие драйверы очереди:

`sync`

Синхронизация или синхронный - это драйвер очереди по умолчанию, который выполняет задание в очереди в рамках существующего процесса. Если этот драйвер включен, у вас фактически нет очереди, поскольку работа в очереди выполняется немедленно. Это полезно для локальных или тестовых целей, но явно не рекомендуется для производства, поскольку оно снижает производительность при настройке очереди.

`database`

Этот драйвер хранит задания в очереди в базе данных. Перед включением этого драйвера вам необходимо создать таблицы базы данных для хранения заданий в очереди и

неудачных заданий:

```
php artisan queue:table
php artisan migrate
```

sqs

Этот драйвер очереди использует Amazon Simple Queue Service для управления заданиями в очереди. Перед включением этого задания необходимо установить следующий пакет `aws/aws-sdk-php ~3.0` : `aws/aws-sdk-php ~3.0`

Также обратите внимание, что если вы планируете использовать задержки для заданий в очереди, Amazon SQS поддерживает максимальную задержку в 15 минут.

iron

Эти драйверы очереди используют Iron для управления заданиями в очереди.

redis

Этот драйвер очереди использует экземпляр Redis для управления заданиями в очереди. Перед использованием этого драйвера очереди вам нужно будет настроить копию Redis и установить следующую зависимость композитора: `predis/predis ~1.0`

beanstalkd

Этот драйвер очереди использует экземпляр Beanstalk для управления заданиями в очереди. Перед использованием этого драйвера очереди вам нужно будет настроить копию Beanstalk и установить следующую зависимость композитора: `pda/pheanstalk ~3.0`

null

Указание нулевого значения в качестве драйвера очереди будет отбрасывать любые задания в очереди.

Прочитайте Очереди онлайн: https://riptutorial.com/ru/laravel/topic/2651/очереди

глава 40: Ошибка несоответствия токена в AJAX

Вступление

Я проанализировал, что отношение получения TokenMismatch Error очень велико. И эта ошибка возникает из-за некоторых глупых ошибок. Есть много причин, по которым разработчики делают ошибки. Вот некоторые из примеров, то есть No _token в заголовках, No _token передал данные при использовании Ajax, проблема разрешения на пути хранения, недопустимый путь хранения сеанса.

Examples

Установить токен в заголовке

Установите токен на `<head>` вашего `default.blade.php` .

```
<meta name="csrf-token" content="{{csrf_token()}}">
```

Добавьте `ajaxSetup` в начало вашего скрипта, который будет доступен везде. Это задает заголовки для каждого вызова `ajax`

```
$.ajaxSetup({
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

Установите токен на тег

Добавьте функцию ниже в свой `<form>` . Эта функция будет генерировать скрытое поле с именем `_token` и заполнять значение с помощью токена.

```
{{csrf_field()}}
```

Добавьте `csrf_token ()` в ваш скрытый `_token` в атрибуте value. Это создаст только зашифрованную строку.

```
<input type="hidden" name="_token" value="{{csrf_token()}}"/> .
```

Проверка пути хранения и разрешения сеанса

Здесь я предполагаю, что URL-адрес приложения проекта - `APP_URL=http://project.dev/ts/toys-store`

1. Задайте для записи доступ к `storage_path('framework/sessions') .`
2. Проверьте путь к вашему проекту laravel `'path' => '/ts/toys-store',` корень вашего проекта laravel.
3. Измените имя вашего cookie `'cookie' => 'toys-store',`

```
return [
    'driver' => env('SESSION_DRIVER', 'file'),
    'lifetime' => 120,
    'expire_on_close' => false,
    'encrypt' => false,
    'files' => storage_path('framework/sessions'),
    'connection' => null,
    'table' => 'sessions',
    'lottery' => [2, 100],
    'cookie' => 'toys-store',
    'path' => '/ts/toys-store',
    'domain' => null,
    'secure' => false,
    'http_only' => true,
];
```

Использовать поле _token на Ajax

Есть много способов отправить `_token` на вызов AJAX

1. Получите все значение поля ввода в `<form>` используя `var formData = new FormData($("#cart-add")[0]);`
2. Используйте `$("form").serialize();` или `$("form").serializeArray();`
3. Добавьте `_token` вручную по `data` Ajax. используя `$('meta[name="csrf-token"]').attr('content')` или `$('input[name="_token"]').val() .`
4. Мы можем установить заголовок для конкретного вызова Ajax, например, кода ниже.

```
$.ajax({
    url: $("#category-add").attr("action"),
    type: "POST",
    data: formData,
    processData: false,
    contentType: false,
    dataType: "json",
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

Прочитайте Ошибка несоответствия токена в AJAX онлайн:
https://riptutorial.com/ru/laravel/topic/10656/ошибка-несоответствия-токена-в-ajax

глава 41: пагинация

Examples

Разбиение страницы в Ларавеле

В других рамках разбиение на страницы - головная боль. Laravel делает это бриз, он может генерировать разбиение на страницы, добавляя несколько строк кода в Controller и View.

Основное использование

Существует много способов разбиения на страницы элементов, но самый простой - использование метода paginate для построителя запросов или запроса Eloquent . Laravel из коробки заботится о настройке предела и смещения на основе текущей страницы, просматриваемой пользователем. По умолчанию текущая страница определяется значением аргумента строки запроса страницы страницы в запросе HTTP. И точно, это значение обнаруживается Laravel автоматически и вставляется в ссылки, созданные с помощью paginator.

Теперь предположим, что мы хотим вызвать метод `paginate` по запросу. В нашем примере переданный аргумент для paginate - это количество элементов, которые вы хотите отобразить «на странице». В нашем случае, допустим, мы хотим отобразить 10 элементов на странице.

```
<?php

namespace App\Http\Controllers;

use DB;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show all of the users for the application.
     *
     * @return Response
     */
    public function index()
    {
        $users = DB::table('users')->paginate(10);

        return view('user.index', ['users' => $users]);
    }
}
```

> **Примечание.** В настоящее время операции с `groupBy` по `groupBy` которые используют оператор `groupBy` не могут эффективно выполняться Laravel. Если

вам нужно использовать `groupBy` с `groupBy` на страницы набором результатов, рекомендуется запросить базу данных и создать файл-указатель вручную.

Простая разбивка на страницы

Предположим, вы просто хотите отображать следующие и предыдущие ссылки в вашем представлении с разбивкой по страницам. Laravel предоставляет вам этот параметр, используя метод `simplePaginate` .

```
$users = DB::table('users')->simplePaginate(10);
```

Отображение результатов в представлении

Теперь можно отобразить разбивку на страницы. Фактически, когда вы вызываете `paginate` или `simplePaginate` методы по запросу Eloquent, вы получаете экземпляр paginator. Когда вызывается метод `Illuminate\Pagination\LengthAwarePaginator` , вы получаете экземпляр `Illuminate\Pagination\LengthAwarePaginator` , а когда вы вызываете метод `simplePaginate` , вы получаете экземпляр `Illuminate\Pagination\Paginator` . Эти экземпляры / объекты имеют несколько методов, которые объясняют набор результатов. Кроме того, в дополнение к этим методам помощников, экземпляры paginator являются итераторами и могут быть закодированы как массив.

Получив результаты, вы можете легко отобразить ссылки на страницы с помощью клинка

```
<div class="container">
    @foreach ($users as $user)
        {{ $user->name }}
    @endforeach
</div>

{{ $users->links() }}
```

Метод `links` автоматически отобразит ссылки на другие страницы в наборе результатов. Каждая из этих ссылок будет содержать конкретную строку страницы, т. Е. Переменную строки запроса `?page` . HTML, сгенерированный методом ссылок, полностью совместим с базой CSS Bootstrap.

Изменение видов разбиения на страницы

При использовании разбиения на страницы laravel вы можете использовать свои собственные пользовательские представления. Поэтому при вызове метода ссылок в экземпляре paginator передайте имя представления в качестве первого аргумента методу, например:

```
{{ $paginator->links('view.name') }}
```

или же

Вы можете настроить виды разбиения на страницы, экспортируя их в каталог `resources/views/vendor` с помощью поставщика: publish:

```
php artisan vendor:publish --tag=laravel-pagination
```

Эта команда поместит представления в каталог `resources/views/vendor/pagination` . Файл `default.blade.php` в этом каталоге соответствует представлению с разбивкой по страницам по умолчанию. Отредактируйте этот файл, чтобы изменить HTML-код разбивки на страницы.

Прочитайте пагинация онлайн: https://riptutorial.com/ru/laravel/topic/2359/пагинация

глава 42: Пакеты Laravel

Examples

Laravel-ида-помощник

Этот пакет генерирует файл, который ваша IDE понимает, поэтому он может обеспечить точное автозаполнение. Генерация выполняется на основе файлов вашего проекта.

Подробнее об этом читайте здесь.

Laravel-DataTables

Этот пакет создан для обработки серверных работ плагина JQuery DataTables через параметр AJAX, используя Eloquent ORM, Fluent Query Builder или Collection.

Подробнее об этом здесь или здесь

Изображение интервенции

Intervention Image представляет собой библиотеку обработки изображений и обработки изображений с открытым исходным кодом. Он обеспечивает более простой и выразительный способ создания, редактирования и компоновки изображений и в настоящее время поддерживает две наиболее распространенные библиотеки обработки изображений GD Library и Imagick.

Подробнее об этом читайте здесь.

Генератор Laravel

Получите свои API-интерфейсы и панель администратора в течение нескольких минут. Генератор Laravel для генерации CRUD, API, тестовых случаев и документации Swagger

Подробнее об этом читайте здесь.

Laravel Socialite

Laravel Socialite обеспечивает выразительный, плавный интерфейс для аутентификации OAuth с помощью Facebook, Twitter, Google, LinkedIn, GitHub и Bitbucket. Он обрабатывает почти весь код социальной аутентификации, который вы боитесь писать.

Подробнее об этом читайте здесь.

Официальные пакеты

Касса

Laravel Cashier предоставляет выразительный, плавный интерфейс для подписки на услуги Stripe's и Braintree . Он обрабатывает почти весь код подписки на подписку, который вы боитесь писать. Помимо основного управления подписками, Cashier может обрабатывать купоны, подписывать подписку, подписные «количества», периоды отмены отмены и даже создавать счета-фактуры.

Подробнее об этом пакете можно найти здесь .

посланник

Laravel Envoy обеспечивает чистый минимальный синтаксис для определения общих задач, выполняемых на ваших удаленных серверах. Используя синтаксис стиля Blade, вы можете легко настроить задачи для развертывания, команд Artisan и т. Д. В настоящее время Envoy поддерживает только операционные системы Mac и Linux.

Этот пакет можно найти на Github .

Паспорт

Laravel уже упрощает аутентификацию через традиционные формы входа, но как насчет API? API обычно используют токены для аутентификации пользователей и не поддерживают состояние сеанса между запросами. Laravel упрощает проверку подлинности API через Laravel Passport, который обеспечивает полную реализацию сервера OAuth2 для вашего приложения Laravel в течение нескольких минут.

Подробнее об этом пакете можно найти здесь .

разведчик

Laravel Scout предоставляет простое, основанное на драйверах решение для добавления полнотекстового поиска в ваши модели Eloquent. Используя модельных наблюдателей, Scout автоматически сохранит ваши поисковые индексы в синхронизации с вашими отчетами Eloquent.

В настоящее время Scout отправляется с водителем Algolia; однако писать пользовательские драйверы просты, и вы можете расширить Scout с помощью собственных поисковых реализаций.

Подробнее об этом пакете можно найти здесь .

Светская

Laravel Socialite обеспечивает выразительный, плавный интерфейс для аутентификации OAuth с помощью Facebook, Twitter, Google, LinkedIn, GitHub и Bitbucket. Он обрабатывает почти весь код социальной аутентификации, который вы боитесь писать.

Этот пакет можно найти на Github .

Прочитайте Пакеты Laravel онлайн: https://riptutorial.com/ru/laravel/topic/8001/пакеты-laravel

глава 43: Планирование заданий

Examples

Создание задачи

Вы можете создать задачу (Командная консоль) в Laravel с помощью Artisan. Из командной строки:

```
php artisan make:console MyTaskName
```

Это создает файл в **app / Console / Commands / MyTaskName.php** . Это будет выглядеть так:

```
<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;

class MyTaskName extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'command:name';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Command description';

    /**
     * Create a new command instance.
     *
     * @return void
     */
    public function __construct()
    {
        parent::__construct();
    }

    /**
     * Execute the console command.
     *
     * @return mixed
     */
    public function handle()
    {
```

```
        //
    }
}
```

Некоторые важные части этого определения:

- Свойство `$signature` - это то, что идентифицирует вашу команду. Вы сможете выполнить эту команду позже через командную строку, используя Artisan, запустив `php artisan command:name` (where `command:name` соответствует `$signature` вашей команде)
- Свойство `$description` является показателем помощи / использования Artisan рядом с вашей командой, когда он становится доступным.
- Метод `handle()` - это то, где вы пишете код для своей команды.

В конце концов, ваша задача будет доступна для командной строки через Artisan. `protected $signature = 'command:name';` свойство этого класса - это то, что вы использовали бы для его запуска.

Создание задачи

Вы можете сделать задачу доступной для Artisan и вашего приложения в файле **app / Console / Kernel.php** .

Класс `Kernel` содержит массив с именем `$commands` который делает ваши команды доступными для вашего приложения.

Добавьте свою команду в этот массив, чтобы сделать его доступным для Artisan и вашего приложения.

```
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        Commands\Inspire::class,
        Commands\MyTaskName::class // This line makes MyTaskName available
    ];

    /**
     * Define the application's command schedule.
     *
     * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
     * @return void
```

```
     */
    protected function schedule(Schedule $schedule)
    {

    }
}
```

Как только это будет сделано, вы можете теперь получить доступ к своей команде через командную строку, используя Artisan. Предполагая, что ваша команда имеет свойство `$signature` установленное в `my:task` , вы можете запустить следующую команду для выполнения своей задачи:

```
php artisan my:task
```

Планирование вашей задачи

Когда ваша команда станет доступной для вашего приложения, вы можете использовать Laravel, чтобы запланировать ее запуск с заранее определенными интервалами, как и CRON.

В файле **app / Console / Kernel.php** вы найдете метод `schedule` который вы можете использовать для планирования своей задачи.

```
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        Commands\Inspire::class,
        Commands\MyTaskName::class
    ];

    /**
     * Define the application's command schedule.
     *
     * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
     * @return void
     */
    protected function schedule(Schedule $schedule)
    {
        $schedule->command('my:task')->everyMinute();
        // $schedule->command('my:task')->everyFiveMinutes();
        // $schedule->command('my:task')->daily();
        // $schedule->command('my:task')->monthly();
```

```
        // $schedule->command('my:task')->sundays();
    }
}
```

Предполагая, что ваша `$signature` `my:task` - `my:task` вы можете запланировать ее, как показано выше, используя объект `Schedule $schedule` . Laravel предоставляет множество различных способов запланировать вашу команду, как показано в прокомментированных строках выше.

Настройка запуска планировщика

Планировщик можно запустить с помощью команды:

```
php artisan schedule:run
```

Планировщик должен запускаться каждую минуту, чтобы работать правильно. Вы можете установить это, создав задание cron со следующей строкой, которая запускает планировщик каждую минуту в фоновом режиме.

```
* * * * * php /path/to/artisan schedule:run >> /dev/null 2>&1
```

Прочитайте Планирование заданий онлайн: https://riptutorial.com/ru/laravel/topic/4026/планирование-заданий

глава 44: Полезные ссылки

Вступление

В этом разделе вы можете найти полезные ссылки, чтобы улучшить свои навыки Laravel или расширить свои знания.

Examples

Laravel Ecosystem

- Laravel Scout - Laravel Scout предоставляет простое, основанное на драйверах решение для добавления полнотекстового поиска в ваши модели Eloquent.
- Laravel Passport - аутентификация API без головной боли. Паспорт - это сервер OAuth2, готовый за считанные минуты.
- Усадьба - официальная среда разработки Laravel. Работает на Vagrant, Homestead получает всю вашу команду на одной странице с последними PHP, MySQL, Postgres, Redis и т. Д.
- Laravel Cashier - Сделайте подписку без бисером со встроенной интеграцией Stripe и Braintree. Купоны, подкачки подписки, аннулирования и даже счета-фактуры PDF готовы из коробки.
- Forge - Предоставление и развертывание неограниченных PHP-приложений на DigitalOcean, Linode и AWS.
- Envoyer - Zero Downtime Развертывание PHP.
- Valet - среда разработки Laravel для минималистов Mac. Нет бродяг, нет Апача, нет суеты.
- Spark - мощные строительные леса SaaS. Прекратите писать шаблон и сосредоточьтесь на своем приложении.
- Lumen - Если вам нужен только API и молниеносная скорость, попробуйте Lumen. Это супер-свет Laravel.
- Statamic - настоящая CMS, призванная сделать агентства прибыльными, разработчики счастливы, а клиенты обнимают вас.

образование

- Laracasts - Изучите практическое, современное веб-развитие, используя экспертные возможности.
- Новости Laravel - Будьте в курсе событий с Laravel с Laravel News.
- Laravel.io - Форум с открытым исходным кодом.

Подкасты

- Подкасты новостей Laravel
- Подкасты Laravel

Прочитайте Полезные ссылки онлайн: https://riptutorial.com/ru/laravel/topic/9957/полезные-ссылки

глава 45: полисы

Examples

Создание политик

Поскольку определение всей логики авторизации в `AuthServiceProvider` может стать громоздким в больших приложениях, Laravel позволяет разделить вашу авторизационную логику на классы «Политика». Политики - это простые классы PHP, которые группируют логику авторизации на основе ресурса, который они разрешают.

Вы можете создать политику, используя команду `make:policy` artisan. Сгенерированная политика будет помещена в каталог `app/Policies` :

```
php artisan make:policy PostPolicy
```

Прочитайте полисы онлайн: https://riptutorial.com/ru/laravel/topic/7344/полисы

глава 46: Помощники

Вступление

Помощники Laravel - это глобально доступные функции, определенные структурой. Его можно напрямую и независимо использовать в любом месте приложения без необходимости создания экземпляра объекта или импорта класса.

Есть помощники для манипулирования *массивами* , *путями* , *строками* , *URL-адресами* и т. Д.

Examples

Методы массива

array_add ()

Этот метод используется для добавления новых пар значений ключа в массив.

```
$array = ['username' => 'testuser'];

$array = array_add($array, 'age', 18);
```

результат

```
['username' => 'testuser', 'age' => 18]
```

Строковые методы

camel_case ()

Этот метод изменяет строку на случай верблюда

```
camel_case('hello_world');
```

результат

```
HelloWorld
```

Путь mehods

Способы путей облегчают доступ к связанным с приложениям пути легко из любого места.

public_path ()

Этот метод возвращает полностью квалифицированный общедоступный путь приложения. который является общедоступным каталогом.

```
$path = public_path();
```

Urls

URL ()

Функция url генерирует полный URL-адрес для данного пути.

если ваш сайт `hello.com`

```
echo url('my/dashboard');
```

вернется

```
hello.com/my/dashboard
```

если ничто не передается методу url, оно возвращает экземпляр `Illuminate\Routing\UrlGenerator` и может быть использовано как это

вернет текущий URL-адрес

```
echo url()->current();
```

вернет полный URL-адрес

```
echo url()->full();
```

вернет предыдущий URL-адрес

```
echo url()->previous();
```

Прочитайте Помощники онлайн: https://riptutorial.com/ru/laravel/topic/8827/помощники

глава 47: Посев базы данных

Examples

Запуск сеялки

Вы можете добавить новый Seeder в класс DatabaseSeeder.

```
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UserTableSeeder::class);
    }
```

Чтобы запустить сеялку базы данных, используйте команду Artisan

```
php artisan db:seed
```

Это запустит класс DatabaseSeeder. Вы также можете использовать параметр `--class=` чтобы вручную указать, какую сеялку следует запустить.

* Примечание. Возможно, вам понадобится запустить компоновщик dumpautoload, если ваш класс Seeder не найден. Обычно это происходит, если вы вручную создаете класс сеялки вместо использования команды artisan.

Создание семян

Семена базы данных хранятся в каталоге / database / seed. Вы можете создать семя, используя команду Artisan.

```
php artisan make:seed UserTableSeeder
```

В качестве альтернативы вы можете создать новый класс, который расширяет `Illuminate\Database\Seeder` . Класс должен иметь общедоступную функцию с именем `run()` .

Вставка данных с использованием сеялки

Вы можете ссылаться на модели в сеялке.

```
use DB;
use App\Models\User;
```

```
class UserTableSeeder extends Illuminate\Database\Seeder{

    public function run(){
        # Remove all existing entrie
        DB::table('users')->delete() ;
        User::create([
            'name' => 'Admin',
            'email' => 'admin@example.com',
            'password' => Hash::make('password')
        ]);

    }
}
```

Вставка данных с помощью Factory Factory

Вы можете захотеть использовать фабрики моделей в своих семенах. Это создаст 3 новых пользователя.

```
use App\Models\User;

class UserTableSeeder extends Illuminate\Database\Seeder{

    public function run(){
        factory(User::class)->times(3)->create();
    }
}
```

Вы также можете, например, указать определенные поля в своем посеве, например, пароль. Это создаст 3 пользователя с одинаковым паролем.

```
factory(User::class)->times(3)->create(['password' => '123456']);
```

Посещение MySQL Dump

Следуйте предыдущему примеру создания семени. В этом примере используется MySQL Dump для разбивки таблицы в базе данных проекта. Перед посевом должна быть создана таблица.

```
<?php

use Illuminate\Database\Seeder;

class UserTableSeeder extends Seeder
{

    /**
     * Run the database seeds.
     *
     * @return void
     */
     public function run()
     {
```

```
        $sql = file_get_contents(database_path() . '/seeds/users.sql');

        DB::statement($sql);
    }
}
```

Наш $ sql будет содержимым нашего дампа users.sql. Свалка должна иметь инструкцию INSERT INTO. Это зависит от вас, где вы храните свои свалки. В приведенном выше примере он сохраняется в каталоге проекта `\database\seeds` . Используя вспомогательную функцию laravel `database_path()` и добавьте каталог и имя файла дампа.

```
INSERT INTO `users` (`id`, `name`, `email`, `password`, `remember_token`, `created_at`,
`updated_at`) VALUES
(1, 'Jane', 'janeDoe@fakemail.com', 'superSecret', NULL, '2016-07-21 00:00:00', '2016-07-21
00:00:00'),
(2, 'John', 'johnny@fakemail.com', 'sup3rS3cr3t', NULL, '2016-07-21 00:00:00', '2016-07-21
00:00:00');
```

`DB::statement($sql)` будет выполнять вставки после запуска Seeder. Как и в предыдущих примерах, вы можете поместить `UserTableSeeder` в класс `DatabaseSeeder` предоставляемый laravel:

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UserTableSeeder::class);
    }
}
```

и запустить из CLI в каталоге проекта `php artisan db:seed` . Или вы можете запустить Seeder для одного класса, используя `php artisan db:seed --class=UsersTableSeeder`

Использование faker и ModelFactories для генерации семян

1) ОСНОВНОЙ ПРОСТОЙ ПУТЬ

Приложениям, управляемым базами данных, часто требуются данные, предварительно загруженные в систему для тестирования и демонстрации.

Чтобы сделать такие данные, сначала создайте класс сеялки

ProductTableSeeder

```
use Faker\Factory as Faker;
use App\Product;

class ProductTableSeeder extends DatabaseSeeder {

public function run()
{
    $faker = $this->getFaker();

    for ($i = 0; $i < 10; $i++)
    {
        $name =          $faker->word;
        $image =         $faker->imageUrl;

        Modelname::create([
            'name' => $name,
            'image' => $image,
        ]);
      }
    }
 }
```

Чтобы вызвать возможность выполнения класса сеялки, вы вызываете его из класса DatabaseSeeder, просто передав имя сеялки, которую вы хотите запустить:

используйте Illuminate \ Database \ Seeder;

```
class DatabaseSeeder extends Seeder {

    protected $faker;

    public function getFaker() {
        if (empty($this->faker)) {
            $faker = Faker\Factory::create();
            $faker->addProvider(new Faker\Provider\Base($faker));
            $faker->addProvider(new Faker\Provider\Lorem($faker));
        }
        return $this->faker = $faker;
    }
    public function run() {
        $this->call(ProductTableSeeder::class);
    }
}
```

Не забудьте запустить `$ composer dump-autoload` после создания Seeder, поскольку они не автоматически автоматически загружаются композитором (если только вы не создали сеялку командой artisan `$ php artisan make:seeder Name`)

Теперь вы готовы к семени, запустив эту команду `php artisan db:seed`

2) ИСПОЛЬЗОВАНИЕ ЗАВОДОВ

Прежде всего, вы должны определить набор атрибутов по умолчанию для каждой модели

в `App/database/factories/ModelFactory.php`

Взятие модели пользователя в качестве примера. Как выглядит ModelFactory

```
$factory->define(App\User::class, function (Faker\Generator $faker) {
    return [
        'name' => $faker->name,
        'email' => $faker->email,
        'password' => bcrypt(str_random(10)),
        'remember_token' => str_random(10),
    ];
});
```

Теперь создайте `php artisan make:seeder UsersTableSeeder` для таблицы `php artisan make:seeder UsersTableSeeder`

И добавьте это

```
public function run()
{
    factory(App\User::class, 100)->create()
}
```

затем добавьте это в `DatabaseSeeder`

```
public function run()
{
    $this->call(UsersTableSeeder::class);
}
```

Это заселоет таблицу со 100 записями.

Прочитайте Посев базы данных онлайн: https://riptutorial.com/ru/laravel/topic/1118/посев-базы-данных

глава 48: почта

Examples

Основной пример

Вы можете настроить Mail, просто добавив / изменив эти строки в файле **.ENV** приложения с **данными** вашего входа в систему электронной почты, например, чтобы использовать его с помощью gmail, который вы можете использовать:

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=yourEmail@gmail.com
MAIL_PASSWORD=yourPassword
MAIL_ENCRYPTION=tls
```

Затем вы можете начать отправку писем с помощью Mail, например:

```
$variable = 'Hello world!'; // A variable which can be use inside email blade template.
Mail::send('your.blade.file', ['variable' => $variable], function ($message) {
            $message->from('john@doe.com');
            $message->sender('john@doe.com');
            $message->to(foo@bar.com);
            $message->subject('Hello World');
        });
```

Прочитайте почта онлайн: https://riptutorial.com/ru/laravel/topic/8014/почта

глава 49: Проверка

параметры

параметр	подробности
требуется	Поле обязательное
иногда	Выполнять проверки проверки по полю только в том случае, если это поле присутствует во входном массиве
Эл. адрес	Вход является действительным адресом электронной почты
макс: значение	Входное значение должно быть ниже максимального значения
Уникальный: db_table_name	Входное значение должно быть уникальным в названии таблицы базы данных
принято	Да / Вкл / 1 истинно, полезно для проверки TOS
active_url	Должен быть действительный URL-адрес в соответствии с checkdnsrr
после *: дата*	Поле под валидацией должно содержать значение после указанной даты
альфа	Поле под валидацией должно быть полностью буквенным символом.
alpha_dash	Поле под проверкой может иметь буквенно-цифровые символы, а также тире и символы подчеркивания.
alpha_num	Поле под проверкой должно быть полностью буквенно-цифровыми символами.
массив	Должен быть массив PHP
до *: дата*	Поле должно быть значением под данной датой
между: мин, макс	Входное значение должно находиться между минимальным (мин) и максимальным (максимальным) значением
логический	Поле под валидацией должно быть отличным как логическое. Принимаемые входные данные: `true` , `false` , `1` , `0` , `"1"` и `"0"` .

параметр	подробности
подтвердил	Поле под проверкой должно иметь соответствующее поле `foo_confirmation` . Например, если поле под проверкой является `password` , на входе должно присутствовать соответствующее поле `password_confirmation` .
Дата	Поле под проверкой должно быть допустимой датой согласно функции PHP strtotime .
целое число	Поле под проверкой должно быть целым числом
строка	Поле под проверкой должно быть строковым .

Examples

Основной пример

Вы можете проверить данные запроса, используя метод `validate` (доступный в базовом контроллере, предоставляемый признаком `ValidatesRequests`).

Если правила пройдут, ваш код будет работать нормально; однако, если проверка не удалась, ответ об ошибке, содержащий ошибки проверки, будет автоматически отправлен обратно:

- для типичных запросов формы HTML пользователь будет перенаправлен на предыдущую страницу, при этом форма сохраняет представленные значения
- для запросов, ожидающих ответа JSON, будет генерироваться HTTP-ответ с кодом 422

Например, в вашем `UserController` вы можете сохранить нового пользователя в методе `store` , который должен быть проверен перед сохранением.

```
/**
 * @param  Request  $request
 * @return Response
 */
public function store(Request $request) {
    $this->validate($request, [
        'name' => 'required',
        'email' => 'email|unique:users|max:255'
    ],
    // second array of validation messages can be passed here
    [
        'name.required' => 'Please provide a valid name!',
        'email.required' => 'Please provide a valid email!',
    ]);

    // The validation passed
}
```

В приведенном выше примере мы проверяем, что поле `name` существует с непустым значением. Во-вторых, мы проверяем, что поле `email` имеет допустимый формат электронной почты, уникально в таблице базы данных «пользователи» и имеет максимальную длину 255 символов.

`|` (pipe) объединяет различные правила валидации для одного поля.

Иногда вы можете прекратить выполнение правил проверки атрибута после первого отказа проверки. Для этого присвойте правилу `bail` атрибут:

```
$this->validate($request, [
    'name' => 'bail|required',
    'email' => 'email|unique:users|max:255'
]);
```

Полный список доступных правил проверки можно найти в разделе параметров ниже .

Проверка массива

Проверка полей ввода формы массива очень проста.

Предположим, вам нужно проверить каждое имя, адрес электронной почты и имя отца в данном массиве. Вы можете сделать следующее:

```
$validator = \Validator::make($request->all(), [
    'name.*'       => 'required',
    'email.*'      => 'email|unique:users',
    'fatherName.*' => 'required'
]);

if ($validator->fails()) {
    return back()->withInput()->withErrors($validator->errors());
}
```

Laravel отображает сообщения по умолчанию для проверки. Однако, если вы хотите, чтобы пользовательские сообщения для полей на основе массива, вы можете добавить следующий код:

```
[
    'name.*' => [
        'required' => 'Name field is required',
    ],
    'email.*' => [
        'unique'   => 'Unique Email is required',
    ],
    'fatherName.*' => [
        'required' => 'Father Name required',
    ]
]
```

Ваш последний код будет выглядеть так:

```
$validator = \Validator::make($request->all(), [
    'name.*'       => 'required',
    'email.*'      => 'email|unique:users',
    'fatherName.*' => 'required',
], [
    'name.*'       => 'Name Required',
    'email.*'      => 'Unique Email is required',
    'fatherName.*' => 'Father Name required',
]);

if ($validator->fails()) {
    return back()->withInput()->withErrors($validator->errors());
}
```

Другие подходы к валидации

1) Проверка запроса формы

Вы можете создать «запрос формы», который может содержать логику авторизации, правила проверки и сообщения об ошибках для конкретного запроса в вашем приложении.

Команда `make:request` Команда Artisan CLI генерирует класс и помещает его в каталог `app/Http/Requests` :

```
php artisan make:request StoreBlogPostRequest
```

Метод `authorize` может быть переопределен логикой авторизации для этого запроса:

```
public function authorize()
{
    return $this->user()->can('post');
}
```

Метод `rules` может быть переопределен конкретными правилами для этого запроса:

```
public function rules()
{
    return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
}
```

Метод `messages` может быть переопределен конкретными сообщениями для этого запроса:

```
public function messages()
{
    return [
        'title.required' => 'A title is required',
        'title.unique' => 'There is another post with the same title',
        'title.max' => 'The title may not exceed :max characters',
        'body.required' => 'A message is required',
    ];
```

```
}
```

Чтобы проверить запрос, просто введите hint конкретный класс запроса в соответствующий метод контроллера. Если проверка не удалась, ответ об ошибке будет отправлен обратно.

```
public function store(StoreBlogPostRequest $request)
{
    // validation passed
}
```

---

2) Ручное создание валидаторов

Для большей гибкости вы можете захотеть создать Validator вручную и обработать неудачную проверку напрямую:

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

use Validator;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class PostController extends Controller
{
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'title' => 'required|unique:posts|max:255',
            'body' => 'required',
        ]);

        if ($validator->fails()) {
            return redirect('post/create')
                    ->withErrors($validator)
                    ->withInput();
        }

        // Store the blog post...
    }
}
```

2) Свободное создание правил

Иногда вам может потребоваться создание уникальных правил «на лету», работа с методом `boot()` в Провайдере может быть сверху, а с Laravel 5.4 вы можете свободно создавать новые правила, используя класс `Rule` .

В качестве примера мы будем работать с `UserRequest` для того, когда вы хотите вставить или обновить пользователя. Пока мы хотим, чтобы имя было обязательным, и адрес электронной почты должен быть уникальным. Проблема с использованием `unique` правила заключается в том, что если вы редактируете пользователя, они могут хранить один и тот

же адрес электронной почты, поэтому вам нужно исключить текущего пользователя из правила. В следующем примере показано, как вы можете легко сделать это, используя новый класс `Rule` .

```
<?php
namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class UserRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules(Request $request)
    {
        $id = $request->route()->getParameter('user');

        return [
            'name'           =>  'required',

            // Notice the value is an array and not a string like usual
            'email'         =>  [
                'required',
                Rule::unique('users')->ignore($id)
            ]
        ];
    }
}
```

Один класс запроса формы для POST, PUT, PATCH

Следуя примеру «Проверка формы запроса» , для `POST` , `PUT` , `PATCH` можно использовать один и тот же класс запросов, поэтому вам не нужно создавать другой класс, используя те же / аналогичные проверки. Это пригодится, если у вас есть уникальные атрибуты в вашей таблице.

```
/**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */
public function rules() {
```

```
    switch($this->method()) {
        case 'GET':
        case 'DELETE':
            return [];
        case 'POST':
            return [
                'name'     => 'required|max:75|unique',
                'category' => 'required',
                'price'    => 'required|between:0,1000',
            ];
        case 'PUT':
        case 'PATCH':
            return [
                'name'     => 'required|max:75|unique:product,name,' . $this->product,
                'category' => 'required',
                'price'    => 'required|between:0,1000',
            ];
        default:break;
    }
}
```

Начиная с вершины, наш оператор switch будет искать тип метода запроса (`GET` , `DELETE` , `POST` , `PUT` , `PATCH`).

В зависимости от метода будет возвращен массив правил. Если у вас есть уникальное поле, например поле `name` в примере, вам нужно указать конкретный идентификатор для проверки, чтобы игнорировать.

```
'field_name' => 'unique:table_name,column_name,' . $idToIgnore`
```

Если у вас есть первичный ключ с надписью, отличным от `id` , вы укажете столбец первичного ключа в качестве четвертого параметра.

```
'field_name' => 'unique:table_name,column_name,' . $idToIgnore . ',primary_key_column'
```

В этом примере мы используем `PUT` и переходим к маршруту (`admin/products/{product}`) значение идентификатора продукта. Таким образом, `$this->product` будет равен `id` для игнорирования.

Теперь ваши правила проверки `PUT|PATCH` и `POST` не обязательно должны быть одинаковыми. Определите свою логику, которая соответствует вашим требованиям. Этот метод позволяет повторно использовать пользовательские сообщения, которые вы, возможно, определили в пользовательском классе запроса формы.

Сообщения об ошибках

Настройка сообщений об ошибках

Файлы `/resources/lang/[lang]/validation.php` содержат сообщения об ошибках, которые будут использоваться валидатором. Вы можете редактировать их по мере необходимости.

Большинство из них имеют заполнители, которые будут автоматически заменены при генерации сообщения об ошибке.

Например, в `'required' => 'The :attribute field is required.'` , То `:attribute` заполнитель будет заменен на имя поля (в качестве альтернативы, вы можете также настроить значение отображения каждого поля в `attributes` массива в том же файле).

пример

Конфигурация сообщения:

```
'required' => 'Please inform your :attribute.',
//...
'attributes => [
    'email' => 'E-Mail address'
]
```

правила:

```
`email' => `required`
```

сообщение об ошибке:

> «Пожалуйста, сообщите свой адрес электронной почты».

Настройка сообщений об ошибках в классе Request

Класс Request имеет доступ к методу `messages()` который должен возвращать массив, это может использоваться для переопределения сообщений без необходимости входить в файлы lang. Например, если у нас есть пользовательская проверка `file_exists` вы можете `file_exists` сообщения, как `file_exists` ниже.

```
class SampleRequest extends Request {

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'image' =>  'required|file_exists'
        ];
    }

    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
```

```
     */
    public function authorize()
    {
        return true;
    }

    public function messages()
    {
        return [
            'image.file_exists' =>  'That file no longer exists or is invalid'
        ];
    }

}
```

Отображение сообщений об ошибках

Ошибки проверки сверкают в сеанс и также доступны в переменной `$errors` , которая автоматически делится на все виды.

Пример отображения ошибок в представлении Blade:

```
@if (count($errors) > 0)
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

Пользовательские правила валидации

Если вы хотите создать собственное правило проверки, вы можете сделать это, например, в методе `boot` поставщика услуг через фасад Validator.

```
<?php
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Validator;

class AppServiceProvider extends ServiceProvider
{
    public function boot()
    {
        Validator::extend('starts_with', function($attribute, $value, $parameters, $validator)
{
            return \Illuminate\Support\Str::startsWith($value, $parameters[0]);
        });

        Validator::replacer('starts_with', function($message, $attribute, $rule, $parameters)
{
```

```
            return str_replace(':needle', $parameters[0], $message);
        });
    }
}
```

Метод `extend` принимает строку, которая будет именем правила и функции, которая, в свою очередь, будет передана имя атрибута, проверяемое значение, массив параметров правила и экземпляр проверки, и должен вернуть проверка проходит. В этом примере мы проверяем, начинается ли строка значений с заданной подстрокой.

Сообщение об ошибке для этого настраиваемого правила может быть установлено как обычно в файле `/resources/lang/[lang]/validation.php` и может содержать заполнители, например, для значений параметров:

```
'starts_with' => 'The :attribute must start with :needle.'
```

Метод `replacer` принимает строку, которая является именем правила и функции, которая, в свою очередь, будет передана исходному сообщению (перед заменой), имени атрибута, имени правила и массиву параметров правила, и должен возвращать сообщение после замены заполнителей по мере необходимости.

Используйте это правило как любое другое:

```
$this->validate($request, [
    'phone_number' => 'required|starts_with:+'
]);
```

Прочитайте Проверка онлайн: https://riptutorial.com/ru/laravel/topic/1310/проверка

глава 50: Промежуточное

Вступление

Middleware - это классы, которые могут быть назначены одному или нескольким маршрутам и используются для выполнения действий на ранней или заключительной этапах цикла запроса. Мы можем думать о них как о серии слоев, которые HTTP-запрос должен пройти, пока он выполняется

замечания

Перед программным кодом контроллера будет выполняться «до» промежуточное программное обеспечение; в то время как промежуточное ПО «После» выполняется после того, как запрос обрабатывается приложением

Examples

Определение промежуточного ПО

Чтобы определить новое промежуточное ПО, мы должны создать класс промежуточного ПО:

```
class AuthenticationMiddleware
{
    //this method will execute when the middleware will be triggered
    public function handle ( $request, Closure $next )
    {
        if ( ! Auth::user() )
        {
            return redirect('login');
        }

        return $next($request);
    }
}
```

Затем мы должны зарегистрировать промежуточное ПО: если промежуточное ПО должно быть привязано ко всем маршрутам приложения, мы должны добавить его в свойство промежуточного ПО `app/Http/Kernel.php` :

```
protected $middleware = [
        \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
        \App\Http\Middleware\AuthenticationMiddleware::class
];
```

в то время как если мы хотим связывать промежуточное программное обеспечение с

некоторыми маршрутами, мы можем добавить его в `$routeMiddleware`

```
//register the middleware as a 'route middleware' giving it the name of 'custom_auth'
protected $routeMiddleware = [
    'custom_auth' => \App\Http\Middleware\AuthenticationMiddleware::class
];
```

а затем привяжите его к одиночным маршрутам следующим образом:

```
//bind the middleware to the admin_page route, so that it will be executed for that route
Route::get('admin_page', 'AdminController@index')->middleware('custom_auth');
```

До и после промежуточного ПО

Примером «до» промежуточного программного обеспечения будет следующее:

```
<?php

namespace App\Http\Middleware;

use Closure;

class BeforeMiddleware
{
    public function handle($request, Closure $next)
    {
        // Perform action

        return $next($request);
    }
}
```

тогда как «после» промежуточное ПО будет выглядеть так:

```
<?php

namespace App\Http\Middleware;

use Closure;

class AfterMiddleware
{
    public function handle($request, Closure $next)
    {
        $response = $next($request);

        // Perform action

        return $response;
    }
}
```

Основное различие заключается в том, как обрабатывается параметр `$request` . Если действия выполняются до `$next($request)` которые будут выполняться до выполнения кода

контроллера при вызове `$next($request)` сначала будут выполняться действия, выполняемые после выполнения кода контроллера.

Маршрутное промежуточное ПО

Любое промежуточное программное обеспечение, зарегистрированное как `routeMiddleware` в `app/Http/Kernel.php` может быть назначено маршруту.

Существует несколько разных способов назначения промежуточного программного обеспечения, но все они делают то же самое.

```
Route::get('/admin', 'AdminController@index')->middleware('auth', 'admin');
Route::get('admin/profile', ['using' => 'AdminController@index', 'middleware' => 'auth']);
Route::get('admin/profile', ['using' => 'AdminController@index', 'middleware' => ['auth',
'admin']);
```

Во всех приведенных выше примерах вы также можете передавать полностью квалифицированные имена классов в качестве промежуточного программного обеспечения, независимо от того, было ли оно зарегистрировано в качестве промежуточного программного обеспечения маршрута.

```
use App\Http\Middleware\CheckAdmin;
Route::get('/admin', 'AdminController@index')->middleware(CheckAdmin::class);
```

Прочитайте Промежуточное онлайн: https://riptutorial.com/ru/laravel/topic/3419/промежуточное

глава 51: просвет

Examples

Начало работы с Lumen

Следующий пример демонстрирует использование `Lumen` *в* `WAMP / MAMP / LAMP` *.*

Для работы с `Lumen` вам нужно сначала установить пару вещей.

- Композитор
- PHPUnit
- git (не требуется, но настоятельно рекомендуется)

Предполагая, что у вас установлены все эти три компонента (по крайней мере, вам нужен композитор), сначала перейдите на корневой каталог вашего веб-сервера с помощью терминала. MacOSX и Linux оснащены отличным терминалом. Вы можете использовать `git bash` (который на самом деле `mingw32` или `mingw64`) в окнах.

```
$ cd path/to/your/document/root
```

Затем вам нужно использовать compose для установки и создания проекта `Lumen` . Выполните следующую команду.

```
$ composer create-project laravel/lumen=~5.2.0 --prefer-dist lumen-project
$ cd lumen-project
```

`lumen-app` в приведенном выше коде является именем папки. Вы можете изменить его по своему усмотрению. Теперь вам нужно настроить свой виртуальный хост, чтобы указать на `path/to/document/root/lumen-project/public` folder. Скажем, вы сопоставили `http://lumen-project.local` с этой папкой. Теперь, если вы перейдете к этому URL-адресу, вы должны увидеть сообщение вроде следующего (в зависимости от вашей установленной версии `Lumen` , в моем случае это было 5.4.4) -

```
Lumen (5.4.4) (Laravel Components 5.4.*)
```

Если вы `lumen-project/routers/web.php` файл `lumen-project/routers/web.php` вы увидите следующее:

```
$app->get('/', function () use($app) {
    return $app->version();
});
```

Поздравляем! Теперь у вас есть рабочая установка `Lumen` . Нет, вы можете расширить это

приложение, чтобы прослушивать свои пользовательские конечные точки.

Прочитайте просвет онлайн: https://riptutorial.com/ru/laravel/topic/9221/просвет

глава 52: Разверните приложение Laravel 5 на общем хостинге на сервере Linux

замечания

Чтобы получить дополнительную информацию о развертывании проекта Laravel на общем хостинге, посетите этот репозиторий Github.

Examples

Приложение Laravel 5 для совместного хостинга на Linux Server

По По умолчанию проекта Laravel в `public` каталог раскрывает содержание приложения , которые могут быть запрошены из любого места кто - либо, остальная часть коды приложения является невидимым или недоступным для тех , кто без соответствующих разрешений.

После разработки приложения на вашей машине разработки его нужно отправить на производственный сервер, чтобы он мог получить доступ через Интернет из любого места - правильно?

Для большинства приложений / веб-сайтов первым выбором является использование совместного пакета хостинга от хостинг-провайдеров, таких как GoDaddy, HostGator и т. Д. В основном из-за низкой стоимости.

> **Обратите внимание** : вы можете попросить своего провайдера вручную изменить **document_root** , так что вам нужно только загрузить ваше приложение Laravel на сервер (через FTP), запросить изменение root в **{app} / public,** и вы должны быть хорошими.

Такие пакеты общедоступного хостинга, однако, имеют ограничения с точки зрения доступа к терминалу и разрешений файлов. По умолчанию необходимо загрузить их приложение / код в папку `public_html` на своей учетной записи общего хостинга.

Поэтому, если вы хотите загрузить проект Laravel на общедоступную учетную запись хостинга, как вы это сделаете? Если вы загрузите все приложение (папку) в папку `public_html` на своей учетной записи общего хостинга? - **Конечно, НЕТ**

Потому что все в папке `public_html` доступно «публично, то есть кем угодно», что было бы большой угрозой безопасности.

Шаги по отправке проекта на общедоступную учетную запись - путь Laravel

Шаг 1
Создайте папку laravel (или что угодно) на том же уровне, что и папка `public_html` .

```
Eg:
/
|--var
    |---www
        |----laravel       //create this folder in your shared hosting account
        |----public_html
        |----log
```

Шаг 2
Скопируйте все, кроме `public` папки с вашего Laravel проекта (на машине развития) в `laravel` папке (на сервере хоста - общий хостинг аккаунт).
Ты можешь использовать:

- C-панель: это будет самый медленный вариант
- FTP-клиент: например, ***FileZilla*** для подключения к вашей учетной записи с общим хостингом и передачи файлов и папок через FTP-загрузку
- Map Network Drive: вы также можете создать подключенный сетевой диск на своей машине разработки, чтобы подключиться к корневой папке вашей учетной записи общего доступа, используя « ftp: // your-domain-name » в качестве сетевого адреса.

Шаг 3
Откройте `public` папку вашего проекта Laravel (на машине разработки), скопировать все и вставить в `public_html` папку (на сервере хоста - общий хостинг аккаунт).
Шаг 4
Теперь откройте файл `index.php` в папке `public_html` на учетной записи общего хостинга (в редакторе cpanel или любом другом подключенном редакторе) и:

Изменить:

```
require __DIR__.'/../bootstrap/autoload.php';
```

Для того, ***чтобы:***

```
require __DIR__.'/../laravel/bootstrap/autoload.php';
```

И Изменить:

```
$app = require_once __DIR__.'/../bootstrap/app.php';
```

Для того, ***чтобы:***

```
$app = require_once __DIR__.'/../laravel/bootstrap/app.php';
```

Сохрани и закрой.

Шаг 5
Теперь перейдите в папку `laravel` (на сервере общедоступного хостинга - сервер) и откройте файл `server.php`
+ Изменить

```
require_once __DIR__.'/public/index.php';
```

Для того, ***чтобы:***

```
require_once __DIR__.'../public_html/index.php';
```

Сохрани и закрой.

Шаг 6
Задайте разрешения для `laravel/storage` (рекурсивно) и всех файлов, подпапок и файлов внутри них на учетной записи общедоступного хостинга - от сервера до `777` .
Примечание. Будьте осторожны с правами доступа к файлам в Linux, они похожи на двуствольный меч, если они не используются правильно, они могут сделать ваше приложение уязвимым для атак. Для понимания прав доступа к файлам Linux вы можете прочитать https://www.linux.com/learn/tutorials/309527-understanding-linux-file-permissions

Шаг 7

Поскольку файл `.env` локального / сервера разработки игнорируется git, его следует игнорировать, поскольку он имеет все переменные среды, включая APP_KEY, и его нельзя публиковать, перетаскивая его в репозитории ». Вы также можете увидеть, что файл `.gitignore` имеет `.env` поэтому он не будет загружать его в репозитории.

После выполнения всех вышеописанных действий сделать `.env` файл в папке Laravel и добавить все переменные окружения , которые вы использовали от локального / сервер разработки в `.env` файл в `.env` файл производственного сервера.

Даже есть файлы конфигурации, такие как `app.php` , `database.php` в папке config приложения laravel, которая определяет эти переменные по умолчанию во втором параметре `env()` но не жестко кодирует значения в этих файлах, так как это повлияет на файлы конфигурации пользователей, которые вытаскивают ваш репозиторий. Поэтому рекомендуется создать файл `.env` вручную!

Также laravel дает `.env-example` который вы можете использовать в качестве ссылки.

Вот и все.

Теперь, когда вы посещаете URL-адрес, который вы настроили как домен с вашим сервером, ваше приложение laravel должно работать так же, как оно работало на вашей машине для создания localhost, в то же время код приложения безопасен и недоступен для всех без надлежащих прав доступа к файлам.

Прочитайте Разверните приложение Laravel 5 на общем хостинге на сервере Linux онлайн: https://riptutorial.com/ru/laravel/topic/2410/разверните-приложение-laravel-5-на-общем-хостинге-на-сервере-linux

глава 53: Разрешения для хранения

Вступление

Laravel требует, чтобы некоторые папки были доступны для записи для пользователя веб-сервера.

Examples

пример

Нам также необходимо установить правильные разрешения для файлов `storage` на `server` . Итак, мы должны предоставить разрешение на запись в каталоге хранилища следующим образом:

```
$ chmod -R 777 ./storage ./bootstrap
```

или вы можете использовать

```
$ sudo chmod -R 777 ./storage ./bootstrap
```

Для окон

Убедитесь, что вы являетесь пользователем admin на этом компьютере с возможностью записи

```
xampp\htdocs\laravel\app\storage needs to be writable
```

NORMAL способ установки разрешений состоит в том, чтобы ваши файлы принадлежали веб-серверу:

```
sudo chown -R www-data:www-data /path/to/your/root/directory
```

Прочитайте Разрешения для хранения онлайн: https://riptutorial.com/ru/laravel/topic/9797/разрешения-для-хранения

глава 54: ремесленник

Синтаксис

- php artisan [команда] [опции] [аргументы]

параметры

команда	Описание
ясно, скомпилированные	Удалить скомпилированный файл класса
вниз	Поместите приложение в режим обслуживания
окр	Отображение текущей среды среды
Помогите	Отображает справку для команды
список	Списки команд
мигрировать	Запуск миграции базы данных
оптимизировать	Оптимизация структуры для повышения производительности
обслуживать	Служить на сервере разработки PHP
паять	Взаимодействие с вашей заявкой
вверх	Вывести приложение из режима обслуживания
Имя приложения	Задайте пространство имен приложений
идентификации: Clear-переустанавливает	Флеш с истекшим сбросом пароля
Кэш: прозрачный	Очистить кеш приложения
Кэш: таблица	Создание переноса для таблицы базы данных кэша
конфигурации: кэш	Создайте файл кеша для более быстрой загрузки конфигурации
конфиг: прозрачный	Удалить файл кеша конфигурации

команда	Описание
дб: семена	Вставьте базу данных с записями
событие: генерировать	Генерировать недостающие события и слушателей на основе регистрации
Ключ: генерировать	Задайте ключ приложения
сделать: Auth	Основные виды регистрации и регистрации лесов
сделать: консоль	Создать новую команду Artisan
сделать: контроллер	Создайте новый класс контроллера
сделать: событие	Создать новый класс событий
сделать: работу	Создать новый класс работы
сделать: слушатель	Создать новый класс прослушивателя событий
сделать: промежуточное программное обеспечение	Создайте новый класс промежуточного программного обеспечения
сделать: миграция	Создать новый файл миграции
сделать модель	Создайте новый класс модели Eloquent
сделать: политику	Создание нового класса политики
сделать: поставщик	Создание нового класса поставщика услуг
Сделать запрос	Создать новый класс запроса формы
сделать: сеялка	Создайте новый класс сеялки
сделать: тест	Создать новый тестовый класс
мигрировать: установить	Создание репозитория миграции
мигрировать: обновить	Сбросить и повторно запустить все миграции
мигрируют: сброс	Откат всех миграций базы данных
мигрировать: Откат	Откат последней миграции базы данных
мигрировать: статус	Показывать статус каждой миграции

команда	Описание
Очередь: сбой	Список всех неудачных заданий очереди
очереди: не удалось стола	Создание миграции для таблицы базы данных заданий неудачных очередей
Очередь: флеш	Сбросить все неудачные задания очереди
Очередь: забыть	Удалить неудачное задание очереди
Очередь: слушать	Слушайте заданную очередь
Очередь: перезагрузка	Перезапустить демонов рабочих очередей после их текущей работы
Очередь: повторить	Повторить попытку неудачной очереди
Очередь: таблица	Создать миграцию для таблицы базы данных заданий очереди
Очередь: работа	Обработать следующее задание в очереди
Маршрут: кэш	Создайте файл кеша маршрута для более быстрой регистрации маршрута
Маршрут: прозрачный	Удалить файл кеша маршрута
Маршрут: список	Список всех зарегистрированных маршрутов
График работы: бежать	Запуск запланированных команд
сессия: таблица	Создание миграции для таблицы базы данных сеанса
Поставщик: публиковать	Публикация любых опубликованных активов из пакетов поставщиков
вид: прозрачный	Очистить все скомпилированные файлы просмотра

Examples

Вступление

Artisan - это утилита, которая может помочь вам выполнять определенные повторяющиеся задачи с помощью команд bash. Он охватывает множество задач, в том числе: работу с

миграциями и **посещением** базы данных, очистку **кеша** , создание необходимых файлов для настройки **аутентификации** , **создание** новых *контроллеров, моделей, классов событий* и многое другое.

> Artisan - это имя интерфейса командной строки, включенного в Laravel. Он предоставляет ряд полезных команд для вашего использования при разработке вашего приложения.

Чтобы просмотреть список всех доступных команд Artisan, вы можете использовать команду list:

```
php artisan list
```

Чтобы узнать больше о любой доступной команде, просто введите ее имя с **помощью** ключевого слова **help** :

```
php artisan help [command-name]
```

Список всех зарегистрированных маршрутов, отфильтрованных несколькими способами

```
php artisan route:list --method=GET --method=POST
```

Это будет включать все маршруты, которые одновременно принимают методы `GET` и `POST` .

Выполнение команд Laravel Artisan с использованием PHP-кода

Вы также можете использовать команды Laravel Artisan с ваших маршрутов или контроллеров.

Для запуска команды с использованием PHP-кода:

```
Artisan::call('command-name');
```

Например,

```
Artisan::call('db:seed');
```

Создание и регистрация новой команды мастера

Вы можете создавать новые команды через

```
php artisan make:command [commandName]
```

Таким образом, это создаст командный класс [commandName] внутри каталога `app/Console/Commands`

.

внутри этого класса вы найдете `protected $signature` и `protected $description` переменные `protected $description` , он представляет имя и описание вашей команды, которое будет использоваться для описания вашей команды.

после создания команды вы можете зарегистрировать свою команду в классе `app/Console/Kernel.php` где вы найдете свойство `commands` .

поэтому вы можете добавить свою команду в массив $ command, например:

```
protected $commands = [
    Commands\[commandName]::class
];
```

а затем я могу использовать свою команду через консоль.

так что в качестве примера я назвал свою команду как

```
protected $signature = 'test:command';
```

Поэтому всякий раз, когда я буду запускать

```
php artisan test:command
```

он вызовет метод `handle` внутри класса, имеющего `test:command` signature `test:command`

Прочитайте ремесленник онлайн: https://riptutorial.com/ru/laravel/topic/1140/ремесленник

глава 55: Светская

Examples

Монтаж

```
composer require laravel/socialite
```

Эта установка предполагает, что вы используете Composer для управления вашими зависимостями с Laravel, что является отличным способом борьбы с ним.

конфигурация

В вашем `config\services.php` вы можете добавить следующий код

```
'facebook' => [
    'client_id' => 'your-facebook-app-id',
    'client_secret' => 'your-facebook-app-secret',
    'redirect' => 'http://your-callback-url',
],
```

Вам также необходимо добавить поставщика в `config\app.php`

Найдите массив `'providers' => []` и, в конце этого, добавьте следующее

```
'providers' => [
      ...

      Laravel\Socialite\SocialiteServiceProvider::class,
]
```

Фасад также снабжен пакетом. Если вы хотите использовать его, убедитесь, что массив `aliases` (также в вашем `config\app.php`) имеет следующий код

```
'aliases' => [
    ....
    'Socialite' => Laravel\Socialite\Facades\Socialite::class,
]
```

Основное использование - фасад

```
return Socialite::driver('facebook')->redirect();
```

Это перенаправит входящий запрос на соответствующий URL-адрес для аутентификации. Основным примером может служить контроллер

```
<?php

namespace App\Http\Controllers\Auth;

use Socialite;

class AuthenticationController extends Controller {

    /**
     * Redirects the User to the Facebook page to get authorization.
     *
     * @return Response
     */
    public function facebook() {
        return Socialite::driver('facebook')->redirect();
    }

}
```

убедитесь, что ваш файл `app\Http\routes.php` имеет маршрут, чтобы разрешить входящий запрос здесь.

```
Route::get('facebook', 'App\Http\Controllers\Auth\AuthenticationController@facebook');
```

Основное использование - инъекция зависимостей

```
/**
 * LoginController constructor.
 * @param Socialite $socialite
 */
public function __construct(Socialite $socialite) {
    $this->socialite = $socialite;
}
```

Внутри конструктора вашего контроллера теперь вы можете ввести класс `Socialite` , который поможет вам обрабатывать логин в социальных сетях. Это заменит использование фасада.

```
/**
 * Redirects the User to the Facebook page to get authorization.
 *
 * @return Response
 */
public function facebook() {
    return $this->socialite->driver('facebook')->redirect();
}
```

Socialite для API - безгражданство

```
public function facebook() {
    return $this->socialite->driver('facebook')->stateless()->redirect()->getTargetUrl();
}
```

Это вернет URL-адрес, который потребитель API должен предоставить конечному пользователю, чтобы получить авторизацию от Facebook.

Прочитайте Светская онлайн: https://riptutorial.com/ru/laravel/topic/1312/светская

глава 56: Связывание модели маршрута

Examples

Неявное связывание

Laravel автоматически разрешает модели Eloquent, определенные в маршрутах или действиях контроллера, имена переменных которых соответствуют имени сегмента маршрута. Например:

```
Route::get('api/users/{user}', function (App\User $user) {
    return $user->email;
});
```

В этом примере, поскольку пользовательская переменная Eloquent $, определенная на маршруте, соответствует сегменту {user} в URI маршрута, Laravel автоматически вводит экземпляр модели, у которого есть идентификатор, соответствующий соответствующему значению из URI запроса. Если экземпляр сопоставимой модели не найден в базе данных, 404 HTTP-ответ будет автоматически сгенерирован.

Если имя таблицы модели составлено из нескольких слов, чтобы заставить неявную привязку модели работать, входная переменная должна быть все строчной;
Например, если пользователь может выполнить какое-то *действие* , и мы хотим получить доступ к этому действию, маршрут будет:

```
Route::get('api/useractions/{useraction}', function (App\UserAction $useraction) {
    return $useraction->description;
});
```

Явное связывание

Чтобы зарегистрировать явное связывание, используйте метод модели маршрутизатора, чтобы указать класс для данного параметра. Вы должны определить свои явные привязки модели в методе загрузки класса RouteServiceProvider

```
public function boot()
{
   parent::boot();

   Route::model('user', App\User::class);
}
```

Затем мы можем определить маршрут, содержащий параметр {user}.

```
$router->get('profile/{user}', function(App\User $user) {
```

```
});
```

Поскольку мы привязали все параметры `{user}` к модели `App\User` , пользовательский экземпляр будет введен в маршрут. Так, например, запрос к `profile/1` будет вводить экземпляр пользователя из базы данных с **идентификатором 1** .

Если экземпляр соответствующей модели не найден в базе данных, **404 HTTP-** ответ будет автоматически сгенерирован.

Прочитайте Связывание модели маршрута онлайн: https://riptutorial.com/ru/laravel/topic/7098/связывание-модели-маршрута

глава 57: Сервисы

Examples

Вступление

Laravel разрешает доступ к различным классам, называемым службами. Некоторые службы доступны из коробки, но вы можете создавать их самостоятельно.

Услуга может использоваться в нескольких файлах приложения, таких как контроллеры. Представим себе Service `OurService` реализующий метод `getNumber()` возвращающий случайное число:

```
class SomeController extends Controller {

    public function getRandomNumber()
    {
        return app(OurService::class)->getNumber();
    }
}
```

Чтобы создать Сервис, необходимо создать новый класс:

```
# app/Services/OurService/OurService.php

<?php
namespace App\Services\OurService;

class OurService
{
    public function getNumber()
    {
        return rand();
    }
}
```

В настоящее время вы уже можете использовать эту службу в контроллере, но вам нужно будет создавать новый объект каждый раз, когда вам это понадобится:

```
class SomeController extends Controller {

    public function getRandomNumber()
    {
        $service = new OurService();
        return $service->getNumber();
    }

    public function getOtherRandomNumber()
    {
        $service = new OurService();
        return $service->getNumber();
```

```
    }
}
```

Вот почему следующий шаг - зарегистрировать свой сервис в **контейнере сервисов** . Когда вы регистрируете Сервис в контейнер услуг, вам не нужно создавать новый объект каждый раз, когда вам это нужно.

Чтобы зарегистрировать услугу в контейнере услуг, вам необходимо создать **поставщика услуг** . Этот поставщик услуг может:

1. Зарегистрируйте свою службу в контейнере услуг с *помощью метода регистрации*)
2. Внедрение других служб в вашу службу (зависимости) с *помощью метода загрузки*

Поставщик услуг - это класс, расширяющий абстрактный класс `Illuminate\Support\ServiceProvider` . Ему необходимо реализовать метод `register()` для **регистрации** Сервиса в **контейнере сервисов** :

```
# app/Services/OurService/OurServiceServiceProvider.php

<?php
namespace App\Services\OurService;

use Illuminate\Support\ServiceProvider;

class OurServiceServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->singleton('OurService', function($app) {
            return new OurService();
        });
    }
}
```

Все **поставщики услуг** сохраняются в массиве в `config/app.php` . Поэтому нам нужно зарегистрировать нашего поставщика услуг в этом массиве:

```
return [

    ...

    'providers' => [

        ...

        App\Services\OurService\OurServiceServiceProvider::class,

        ...

    ],

    ...

];
```

Теперь мы можем получить доступ к нашей службе в контроллере. Три возможности:

1. Внедрение зависимости:

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

use App\Services\OurService\OurService;

class SomeController extends Controller
{
    public function __construct(OurService $our_service)
    {
        dd($our_service->getNumber());
    }
}
```

2. Доступ через помощник `app()` :

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

use App\Services\OurService\OurService;

class SomeController extends Controller
{
    public function getRandomNumber()
    {
        return app('OurService')->getNumber();
    }
}
```

Laravel предоставляет Фасады, воображаемые классы, которые вы можете использовать во всех своих проектах и отражать Сервис. Чтобы облегчить доступ к вашему сервису, вы можете создать фасад:

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

use Randomisator;

class SomeController extends Controller
{
    public function getRandomNumber()
    {
        return Randomisator::getNumber();
    }

}
```

Чтобы создать новый Facade, вам нужно создать новый класс, расширяющий `Illuminate\Support\Facades\Facade` . Этот класс должен реализовать метод `getFacadeAccessor()` и вернуть имя службы, зарегистрированной **поставщиком услуг** :

```
# app/Services/OurService/OurServiceFacade.php

<?php
namespace App\Services\OurService;

use Illuminate\Support\Facades\Facade;

class OurServiceFacade extends Facade
{
    protected static function getFacadeAccessor()
    {
        return 'OurService';
    }
}
```

Вам также необходимо зарегистрировать свой Фасад в `config/app.php` :

```
return [

    ...

    'aliases' => [

        ....

        'Randomisator' => App\Services\OurService\OurServiceFacade::class,
    ],

];
```

Теперь Facade доступен в любом месте вашего проекта.

Если вы хотите получить доступ к своей службе из своих представлений, вы можете создать вспомогательную функцию. Laravel поставляется с некоторыми помощниками, функционирует из коробки, как функция `auth()` или функция `view()` . Чтобы создать вспомогательную функцию, создайте новый файл:

```
# app/Services/OurService/helpers.php

if (! function_exists('randomisator')) {
    /**
     * Get the available OurService instance.
     *
     * @return \App\ElMatella\FacebookLaravelSdk
     */
    function randomisator()
    {
        return app('OurService');
    }
}
```

Вам также необходимо зарегистрировать этот файл, но в файле `composer.json` :

```
{
```

```
    ...

    "autoload": {
        "files": [
            "app/Services/OurService/helpers.php"
        ],

        ...
    }
}
```

Теперь вы можете использовать этот помощник в представлении:

```
<h1>Here is a random number: {{ randomisator()->getNumber() }}</h1>
```

Прочитайте Сервисы онлайн: https://riptutorial.com/ru/laravel/topic/1907/сервисы

глава 58: Сервисы

Examples

Связывание интерфейса с внедрением

В методе `register` Service Provider мы можем привязать интерфейс к реализации:

```
public function register()
{
    App::bind( UserRepositoryInterface::class, EloquentUserRepository::class );
}
```

С этого момента каждый раз, когда приложение понадобится экземпляр `UserRepositoryInterface` , Laravel автоматически `UserRepositoryInterface` новый экземпляр `EloquentUserRepository` :

```
//this will get back an instance of EloquentUserRepository
$repo = App::make( UserRepositoryInterface:class );
```

Связывание экземпляра

Мы можем использовать Service Container как реестр, привязывая экземпляр объекта в нем и возвращая его, когда нам это нужно:

```
// Create an instance.
$john = new User('John');

// Bind it to the service container.
App::instance('the-user', $john);

// ...somewhere and/or in another class...

// Get back the instance
$john = App::make('the-user');
```

Привязка синглтона к сервисному контейнеру

Мы можем связать класс как Singleton:

```
public function register()
{
    App::singleton('my-database', function()
    {
        return new Database();
    });
}
```

Таким образом, в первый раз, когда экземпляр `'my-database'` будет запрашиваться в контейнере службы, будет создан новый экземпляр. Все последующие запросы этого класса вернут первый созданный экземпляр:

```
//a new instance of Database is created
$db = App::make('my-database');

//the same instance created before is returned
$anotherDb = App::make('my-database');
```

Вступление

Контейнер сервисов является основным объектом приложения. Он может использоваться в качестве контейнера для инъекций зависимостей и реестра для приложения путем определения привязок в поставщиках услуг

Поставщики услуг - это классы, где мы определяем способ создания наших классов обслуживания через приложение, загрузку их конфигурации и привязку интерфейсов к реализациям

Службы - это классы, которые объединяют одну или несколько взаимосвязанных логических задач

Использование контейнер-контейнера в качестве контейнера для инъекций зависимостей

Мы можем использовать контейнер обслуживания как контейнер для инъекций зависимостей, связывая процесс создания объектов со своими зависимостями в одной точке приложения

Предположим, что для создания `PdfCreator` нужны два объекта в качестве зависимостей; каждый раз, когда нам нужно создать экземпляр `PdfCreator` , мы должны передать эти зависимости в конструктор che. Используя контейнер сервисов как DIC, мы определяем создание `PdfCreator` в определении привязки, принимая требуемую зависимость напрямую из Service Container:

```
App:bind('pdf-creator', function($app) {

    // Get the needed dependencies from the service container.
    $pdfRender = $app->make('pdf-render');
    $templateManager = $app->make('template-manager');

    // Create the instance passing the needed dependencies.
    return new PdfCreator( $pdfRender, $templateManager );
});
```

Затем, в каждой точке нашего приложения, чтобы получить новый `PdfCreator` , мы можем просто сделать:

```
$pdfCreator = App::make('pdf-creator');
```

И контейнер службы создаст новый экземпляр вместе с необходимыми зависимостями для нас.

Прочитайте Сервисы онлайн: https://riptutorial.com/ru/laravel/topic/1908/сервисы

глава 59: События и слушатели

Examples

Использование событий и прослушивателей для отправки электронной почты новому зарегистрированному пользователю

События Laravel позволяют реализовать шаблон Observer. Это можно использовать для отправки приветственного письма пользователю, когда они регистрируются в вашем приложении.

Новые события и слушатели могут быть сгенерированы с помощью утилиты командной строки artisan после регистрации события и их конкретного слушателя в классе `App\Providers\EventServiceProvider` .

```
protected $listen = [
    'App\Events\NewUserRegistered' => [
        'App\Listeners\SendWelcomeEmail',
    ],
];
```

Альтернативная нотация:

```
protected $listen = [
    \App\Events\NewUserRegistered::class => [
        \App\Listeners\SendWelcomeEmail::class,
    ],
];
```

Теперь выполните `php artisan generate:event` . Эта команда будет генерировать все соответствующие события и прослушиватели, упомянутые выше, в `App\Events` и `App\Listeners` соответственно.

Мы можем иметь несколько слушателей для одного события, например

```
protected $listen = [
    'Event' => [
        'Listner1', 'Listener2'
    ],
];
```

`NewUserRegistered` - это всего лишь класс-оболочка для недавно зарегистрированной модели пользователя:

```
class NewUserRegistered extends Event
{
    use SerializesModels;
```

```
    public $user;

    /**
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct(User $user)
    {
        $this->user = $user;
    }
}
```

Это `Event` будет обработано слушателем `SendWelcomeEmail` :

```
class SendWelcomeEmail
{
    /**
     * Handle the event.
     *
     * @param  NewUserRegistered  $event
     */
    public function handle(NewUserRegistered $event)
    {
        //send the welcome email to the user
        $user = $event->user;
        Mail::send('emails.welcome', ['user' => $user], function ($message) use ($user) {
                $message->from('hi@yourdomain.com', 'John Doe');
                $message->subject('Welcome aboard '.$user->name.'!');
                $message->to($user->email);
        });
    }
}
```

Последний шаг - вызывать / запускать событие каждый раз, когда регистрируется новый пользователь. Это можно сделать в контроллере, команде или службе, где бы вы не реализовали логику регистрации пользователя:

```
event(new NewUserRegistered($user));
```

Прочитайте События и слушатели онлайн: https://riptutorial.com/ru/laravel/topic/4687/события-и-слушатели

глава 60: Структура каталога

Examples

Изменить каталог приложений по умолчанию

Существуют случаи использования, когда вы можете переименовать каталог приложений в другое. В Laravel4 вы можете просто изменить запись в config, вот один из способов сделать это в Laravel5.

В этом примере мы переименуем каталог `app` в `src` .

Класс переопределения приложений

Каталоги имя `app` является зашито в класс основного приложения, поэтому он должен быть перегружен. Создайте новый файл `Application.php` . Я предпочитаю держать мой в каталоге `src` (тот, который мы будем заменять приложением), но вы можете разместить его в другом месте.

Вот как должен выглядеть переопределенный класс. Если вы хотите другое имя, просто измените строку `src` на другое.

```
namespace App;

class Application extends \Illuminate\Foundation\Application
{
    /**
     * @inheritdoc
     */
    public function path($path = '')
    {
        return $this->basePath . DIRECTORY_SEPARATOR . 'src' . ($path ? DIRECTORY_SEPARATOR .
$path : $path);
    }
}
```

Сохраните файл. Мы закончили с этим.

Вызов нового класса

Откройте `bootstrap/app.php` и найдите

```
$app = new Illuminate\Foundation\Application(
    realpath(__DIR__.'/../')
);
```

Мы заменим его этим

```
$app = new App\Application(
    realpath(__DIR__.'/../')
);
```

Композитор

Откройте файл `composer.json` и измените автозагрузку в соответствии с вашим новым местоположением

```
"psr-4": {
    "App\\": "src/"
}
```

И, наконец, в командной строке запустите `composer dump-autoload` и ваше приложение должно быть отправлено из каталога `src` .

Измените каталог контроллеров

если мы хотим изменить каталог `Controllers` нам нужно:

1. Переместите и / или переименуйте каталог `Controllers` по умолчанию, где мы хотим. Например, из `app/Http/Controllers` в `app/Controllers`
2. Обновите все пространства имен файлов внутри папки `Controllers` , чтобы они придерживались нового пути, учитывая специфику PSR-4.
3. Измените пространство имен, которое применяется к файлу `routes.php` , отредактировав `app\Providers\RouteServiceProvider.php` и изменив это:

```
protected $namespace = 'App\Http\Controllers';
```

к этому:

```
protected $namespace = 'App\Controllers';
```

Прочитайте Структура каталога онлайн: https://riptutorial.com/ru/laravel/topic/3153/структура-каталога

глава 61: тестирование

Examples

Вступление

Написание тестируемого кода является важной частью создания надежного, удобного и гибкого проекта. Поддержка самой широко используемой структуры PHP PHPUnit построена прямо в Laravel. PHPUnit настроен с использованием файла `phpunit.xml` , который находится в корневом каталоге каждого нового приложения Laravel.

Каталог `tests` , также в корневом каталоге, содержит отдельные файлы тестирования, которые содержат логику тестирования каждой части вашего приложения. Разумеется, именно ваша ответственность за разработку этих тестов заключается в том, что вы создаете свое приложение, но Laravel включает примерный файл `ExampleTest.php` , чтобы вы `ExampleTest.php` .

```
<?php

use Illuminate\Foundation\Testing\WithoutMiddleware;
use Illuminate\Foundation\Testing\DatabaseMigrations;
use Illuminate\Foundation\Testing\DatabaseTransactions;

class ExampleTest extends TestCase
{
    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testBasicExample()
    {
        $this->visit('/')
             ->see('Laravel 5');
    }
}
```

В `testBasicExample()` мы `testBasicExample()` на страницу индекса сайта и удостоверяемся, что мы видим текст `Laravel 5` где-то на этой странице. Если текст отсутствует, тест завершится с ошибкой и вызовет ошибку.

Тест без промежуточного программного обеспечения и новая база данных

Чтобы заставить ремесленника перенести новую базу данных перед запуском тестов, `use DatabaseMigrations` . Кроме того, если вы хотите избежать промежуточного программного обеспечения, такого как Auth, `use WithoutMiddleware` .

```
<?php

use Illuminate\Foundation\Testing\WithoutMiddleware;
use Illuminate\Foundation\Testing\DatabaseMigrations;

class ExampleTest extends TestCase
{
    use DatabaseMigrations, WithoutMiddleware;

    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testExampleIndex()
    {
        $this->visit('/protected-page')
         ->see('All good');
    }
}
```

Операции с базами данных для подключения к нескольким базам данных

`DatabaseTransactions` trait позволяет базам откатить все изменения во время тестов. Если вы хотите откатить несколько баз данных, вам нужно установить `$connectionsToTransact` properties

```
use Illuminate\Foundation\Testing\DatabaseMigrations;

class ExampleTest extends TestCase
{
     use DatabaseTransactions;

     $connectionsToTransact =["mysql","sqlite"] //tell Laravel which database need to rollBack

    public function testExampleIndex()
    {
        $this->visit('/action/parameter')
         ->see('items');
    }
}
```

Настройка тестирования, использование в базе данных памяти

После установки убедитесь, что среда тестирования (PHPUnit) использует `:memory:` database.

конфиг / database.php

```
'connections' => [

    'sqlite_testing' => [
        'driver'   => 'sqlite',
```

```
        'database' => ':memory:',
        'prefix'   => '',
    ],
    .
    .
    .
```

./phpunit.xml

```
    .
    .
    .
    </filter>
    <php>
        <env name="APP_ENV" value="testing"/>
        <env name="APP_URL" value="http://example.dev"/>
        <env name="CACHE_DRIVER" value="array"/>
        <env name="SESSION_DRIVER" value="array"/>
        <env name="QUEUE_DRIVER" value="sync"/>
        <env name="DB_CONNECTION" value="sqlite_testing"/>
    </php>
</phpunit>
```

конфигурация

Файл phpunit.xml является конфигурационным файлом по умолчанию для тестов и уже настроен для тестирования с помощью PHPUnit.

Стандартная среда тестирования `APP_ENV` определяется как `testing` с `array` являющимся драйвером кэша `CACHE_DRIVER` . При этой настройке никакие данные (сеанс / кеш) не будут сохранены во время тестирования.

Для запуска тестов в отношении определенной среды, такой как усадьба, значения по умолчанию могут быть изменены на:

```
<env name="DB_HOST" value="192.168.10.10"/>
<env name="DB_DATABASE" value="homestead"/>
<env name="DB_USERNAME" value="homestead"/>
<env name="DB_PASSWORD" value="secret"/>
```

Или использовать временную базу данных *памяти* :

```
<env name="DB_CONNECTION" value="sqlite"/>
<env name="DB_DATABASE" value=":memory:"/>
```

Последнее примечание, которое следует учитывать в документации Laravel :

> Обязательно очистите кеш конфигурации, используя команду `config:clear` Artisan перед запуском тестов!

Прочитайте тестирование онлайн: https://riptutorial.com/ru/laravel/topic/1249/тестирование

глава 62: Удалить публикацию из URL-адреса в laravel

Вступление

Как удалить `public` из URL-адреса в Laravel, есть много ответов в Интернете, но самый простой способ описан ниже

Examples

Как это сделать?

Выполните следующие действия, чтобы удалить `public` из URL-адреса

1. Скопируйте файл `.htaccess` из `/public` каталога в корневую папку `Laravel/project` .
2. Переименуйте `server.php` в корневую папку `Laravel/project` в `index.php` .

Приветствую вас сейчас.

Обратите внимание: оно проверено на *Laravel 4.2* , *Laravel 5.1* , *Laravel 5.2* , *Laravel 5.3* .

Я думаю, что это самый простой способ удалить `public` из URL-адреса.

Удалить публикацию из URL-адреса

1. Переименование `server.php` на `index.php`
2. Скопируйте `.htaccess` из `public` папки в `root` папке
3. Изменение `.htaccess` для статистики:

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)/$ /$1 [L,R=301]

RewriteCond %{REQUEST_URI} !(\.css|\.js|\.png|\.jpg|\.gif|robots\.txt)$ [NC]
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_URI} !^/public/
RewriteRule ^(css|js|images)/(.*)$ public/$1/$2 [L,NC]
```

Иногда я использую этот метод для удаления URL `public` формы.

Прочитайте Удалить публикацию из URL-адреса в laravel онлайн:
https://riptutorial.com/ru/laravel/topic/9786/удалить-публикацию-из-url-адреса-в-laravel

глава 63: Файловая система / облачное хранилище

Examples

конфигурация

Файл конфигурации файловой системы находится в файле `config/filesystems.php` . Внутри этого файла вы можете настроить все ваши «диски». Каждый диск представляет собой определенный драйвер хранилища и место хранения. Примеры конфигураций для каждого поддерживаемого драйвера включены в файл конфигурации. Таким образом, просто измените конфигурацию, чтобы отразить ваши предпочтения и учетные данные для хранения!

Перед использованием драйверов S3 или Rackspace вам необходимо установить соответствующий пакет через Composer:

- Amazon S3: `league/flysystem-aws-s3-v2 ~1.0`
- Rackspace: `league/flysystem-rackspace ~1.0`

Конечно, вы можете настроить столько дисков, сколько хотите, и даже можете иметь несколько дисков, которые используют один и тот же драйвер.

При использовании локального драйвера обратите внимание, что все операции с файлами относятся к корневому каталогу, определенному в вашем файле конфигурации. По умолчанию это значение устанавливается в `storage/app directory` . Поэтому следующий метод сохранит файл в файле `storage/app/file.txt` :

```
Storage::disk('local')->put('file.txt', 'Contents');
```

Основное использование

Фасад `Storage` может использоваться для взаимодействия с любым из ваших сконфигурированных дисков. В качестве альтернативы вы можете набрать команду « `Illuminate\Contracts\Filesystem\Factory` на любом классе, который разрешен через контейнер службы Laravel.

Получение отдельного диска

```
$disk = Storage::disk('s3');

$disk = Storage::disk('local');
```

Определение наличия файла

```
$exists = Storage::disk('s3')->exists('file.jpg');
```

Способы вызова на диске по умолчанию

```
if (Storage::exists('file.jpg'))
{
    //
}
```

Получение содержимого файла

```
$contents = Storage::get('file.jpg');
```

Настройка содержимого файла

```
Storage::put('file.jpg', $contents);
```

Подготовка к файлу

```
Storage::prepend('file.log', 'Prepended Text');
```

Добавить в файл

```
Storage::append('file.log', 'Appended Text');
```

Удалить файл

```
Storage::delete('file.jpg');

Storage::delete(['file1.jpg', 'file2.jpg']);
```

Скопировать файл в новое место

```
Storage::copy('old/file1.jpg', 'new/file1.jpg');
```

Переместить файл в новое место

```
Storage::move('old/file1.jpg', 'new/file1.jpg');
```

Получить размер файла

```
$size = Storage::size('file1.jpg');
```

Получить последнее время модификации (UNIX)

```
$time = Storage::lastModified('file1.jpg');
```

Получить все файлы в каталоге

```
$files = Storage::files($directory);

// Recursive...
$files = Storage::allFiles($directory);
```

Получить все каталоги в каталоге

```
$directories = Storage::directories($directory);

// Recursive...
$directories = Storage::allDirectories($directory);
```

Создать каталог

```
Storage::makeDirectory($directory);
```

Удалить каталог

```
Storage::deleteDirectory($directory);
```

Пользовательские файловые системы

Интеграция FlySystem от Laravel предоставляет драйверы для нескольких «драйверов» из коробки; однако Flysystem не ограничивается ими и имеет адаптеры для многих других систем хранения. Вы можете создать собственный драйвер, если хотите использовать один из этих дополнительных адаптеров в своем приложении Laravel. Не волнуйся, это не слишком сложно!

Чтобы настроить пользовательскую файловую систему, вам необходимо создать поставщика услуг, такого как `DropboxFilesystemServiceProvider` . В способе `boot` провайдера вы можете ввести экземпляр договора `Illuminate\Contracts\Filesystem\Factory` и вызвать метод `extend` инъецируемого экземпляра. Кроме того, вы можете использовать метод `extend` фасада `Disk` .

Первым аргументом метода `extend` является имя драйвера, а второе - закрытие, которое получает переменные `$app` и `$config` . Завершение решения должно возвращать экземпляр `League\Flysystem\Filesystem` .

> Примечание: переменная $ config уже содержит значения, определенные в `config/filesystems.php` для указанного диска. Пример Dropbox

```
<?php namespace App\Providers;
```

```
use Storage;
use League\Flysystem\Filesystem;
use Dropbox\Client as DropboxClient;
use League\Flysystem\Dropbox\DropboxAdapter;
use Illuminate\Support\ServiceProvider;

class DropboxFilesystemServiceProvider extends ServiceProvider {

    public function boot()
    {
        Storage::extend('dropbox', function($app, $config)
        {
            $client = new DropboxClient($config['accessToken'], $config['clientIdentifier']);

            return new Filesystem(new DropboxAdapter($client));
        });
    }

    public function register()
    {
        //
    }

}
```

Создание символической ссылки на веб-сервере с использованием SSH

В документации Laravel должна быть создана символическая ссылка (символическая ссылка или софт-ссылка) из общедоступного / хранилища в хранилище / приложение / публикация, чтобы сделать файлы доступными из Интернета.

(ЭТА ПРОЦЕДУРА СОЗДАВАЕТ СИМВОЛИЧЕСКУЮ ЛИНИЮ В РАЙОНЕ ПРОЕКТА LARAVEL)

Ниже приведены шаги по созданию символической ссылки на вашем веб-сервере Linux с использованием SSH-клиента:

1. Подключите и войдите в систему на свой веб-сервер, используя SSH-клиент (например, PUTTY).
2. Свяжите **хранилище** / **приложение** / **общедоступное** с **общедоступным** / **хранилищем** с помощью синтаксиса

   ```
   ln -s target_path link_path
   ```

   Пример (в каталоге файлов CPanel)

   ```
   ln -s /home/cpanel_username/project_name/storage/app/public
   /home/cpanel_sername/project_name/public/storage
   ```

*(Будет создана папка с именем **storage** , чтобы связать путь с индикатором **>>>** на значке папки.)*

Прочитайте Файловая система / облачное хранилище онлайн:

https://riptutorial.com/ru/laravel/topic/3040/файловая-система---облачное-хранилище

глава 64: Функция пользовательского помощника

Вступление

Добавление пользовательских помощников может помочь вам с вашей скоростью разработки. При написании таких вспомогательных функций нужно учитывать несколько вещей, поэтому этот учебник.

замечания

Всего несколько указателей:

- Мы помещаем определения функций в проверку (`function_exists`), чтобы исключить исключения, когда поставщик услуг вызывается дважды.
- Альтернативный способ - зарегистрировать файл помощников из файла `composer.json` . Вы можете скопировать логику из самой структуры laravel .

Examples

document.php

```
<?php

if (!function_exists('document')) {
    function document($text = '') {
        return $text;
    }
}
```

Создайте файл helpers.php, предположим, пока он живет в `app/Helpers/document.php` . Вы можете поместить много помощников в один файл (так это делает Laravel), или вы можете разделить их по имени.

HelpersServiceProvider.php

Теперь давайте создадим поставщика услуг. Положим это на `app/Providers` :

```
<?php

namespace App\Providers;

class HelpersServiceProvider extends ServiceProvider
{
```

```
    public function register()
    {
        require_once __DIR__ . '/../Helpers/document.php';
    }
}
```

Вышеупомянутый поставщик услуг загружает файл помощников и автоматически регистрирует вашу настраиваемую функцию. Пожалуйста, убедитесь, что вы зарегистрировали этот HelpersServiceProvider в своем `config/app.php` у `providers` :

```
'providers' => [
     // [..] other providers
     App\Providers\HelpersServiceProvider::class,
]
```

использование

Теперь вы можете использовать `document()` функции `document()` всюду в своем коде, например, в шаблонах клинков. Этот пример возвращает только ту же строку, которую он получает в качестве аргумента

```
<?php
Route::get('document/{text}', function($text) {
    return document($text);
});
```

Теперь перейдите в `/document/foo` в своем браузере (используйте `php artisan serve` или `valet`), который вернет `foo` .

Прочитайте Функция пользовательского помощника онлайн:
https://riptutorial.com/ru/laravel/topic/8347/функция-пользовательского-помощника

глава 65: Шаблоны кликов

Вступление

Laravel поддерживает Blade templating engine из коробки. Механизм шаблонов Blade позволяет создавать мастер-шаблоны и шаблоны для детей, загружая контент из мастер-шаблонов, мы можем иметь переменные, циклы и условные операторы внутри файла клика.

Examples

Просмотры:

Виды, в шаблон MVC, содержит логику о *том* , *как* представить данные пользователю. В веб-приложении обычно они используются для генерации вывода HTML, который отправляется обратно пользователям с каждым ответом. По умолчанию представления в Laravel хранятся в каталоге `resources/views` .

Вид можно вызвать с помощью вспомогательной функции `view` :

```
view(string $path, array $data = [])
```

Первый параметр помощника - это путь к файлу представления, а второй параметр - необязательный массив данных для передачи в представление.

Поэтому, чтобы вызвать `resources/views/example.php` , вы должны использовать:

```
view('example');
```

Просмотр файлов в подпапках в каталоге `resources/views` , таких как `resources/views/parts/header/navigation.php` , можно вызвать с помощью точечной нотации: `view('parts.header.navigation');`

В файле вида, например `resources/views/example.php` , вы можете включать как HTML, так и PHP вместе:

```
<html>
    <head>
        <title>Hello world!</title>
    </head>
    <body>
        <h1>Welcome!</h1>
        <p>Your name is: <?php echo $name; ?></p>
    </body>
</html>
```

В предыдущем примере (который не использует специальный синтаксис Blade) мы выводим переменную `$name` . Чтобы передать это значение нашему представлению, мы передадим массив значений при вызове помощника вида:

```
view('example', ['name' => $name]);
```

или, альтернативно, используйте помощник `compact()` . В этом случае строка, переданная в compact (), соответствует имени переменной, которую мы хотим передать в представление.

```
view('example', compact('name'));
```

НАИМЕНОВАЯ КОНВЕНЦИЯ ДЛЯ ПЕРЕМЕННЫХ ЛЕКАРСТВ

При отправке данных назад для просмотра. Вы можете использовать `underscore` для многословной `variable` но с `-` laravel дает ошибку.

Как и этот, вы получите сообщение об ошибке (отметьте `hyphen ( - )` в `user-address`

```
view('example',['user-address' => 'Some Address']);
```

Правильный способ сделать это будет

```
view('example', ['user_address' => 'Some Address']);
```

Контрольные структуры

Blade обеспечивает удобный синтаксис для общих структур управления PHP.

Каждая из структур управления начинается с `@[structure]` и заканчивается `@[endstructure]` . Обратите внимание, что в тегах мы просто вводим обычный HTML и включаем переменные с синтаксисом Blade.

Conditionals

Заявления «Если»

```
@if ($i > 10)
    <p>{{ $i }} is large.</p>
@elseif ($i == 10)
    <p>{{ $i }} is ten.</p>
@else
    <p>{{ $i }} is small.</p>
@endif
```

«За исключением»

(Короткий синтаксис для 'if not'.)

```
@unless ($user->hasName())
    <p>A user has no name.</p>
@endunless
```

Loops

Цикл «Пока»

```
@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

Цикл «Foreach»

```
@foreach ($users as $id => $name)
    <p>User {{ $name }} has ID {{ $id }}.</p>
@endforeach
```

Петля Forelse

(То же, что и цикл foreach, но добавляет специальную директиву `@empty` , которая выполняется, когда выражение массива, `@empty` которого пуста, является способом отображения содержимого по умолчанию).

```
@forelse($posts as $post)
    <p>{{ $post }} is the post content.</p>
@empty
    <p>There are no posts.</p>
@endforelse
```

Внутри циклов будет доступна специальная переменная `$loop` , содержащая информацию о состоянии цикла:

Имущество	Описание
`$loop->index`	Индекс текущей итерации цикла (начинается с 0).
`$loop->iteration`	Текущая итерация цикла (начинается с 1).

Имущество	Описание
`$loop->remaining`	Оставшиеся итерации цикла.
`$loop->count`	Общее количество элементов в массиве, которое повторяется.
`$loop->first`	Является ли это первой итерацией по циклу.
`$loop->last`	Является ли это последней итерацией цикла.
`$loop->depth`	Уровень вложенности текущего цикла.
`$loop->parent`	Когда во вложенном цикле переменная цикла родителя.

Пример:

```
@foreach ($users as $user)
  @foreach ($user->posts as $post)
        @if ($loop->parent->first)
            This is first iteration of the parent loop.
        @endif
    @endforeach
@endforeach
```

Начиная с Laravel 5.2.22, мы также можем использовать директивы `@continue` и `@break`

Имущество	Описание
`@continue`	Остановите текущую итерацию и запустите следующую.
`@break`	Остановите текущий цикл.

Пример :

```
@foreach ($users as $user)
    @continue ($user->id == 2)
        <p>{{ $user->id }} {{ $user->name }}</p>
    @break ($user->id == 4)
@endforeach
```

Затем (*при условии, что 5+ пользователей сортируются по ID, и идентификатор отсутствует*), страница будет отображать

```
1 Dave
3 John
4 William
```

Повторение выражений PHP

Любое выражение PHP в двойных фигурных скобках `{{ $variable }}` будет `echo` после выполнения функции `e` helper . (Таким образом, специальные символы html (`<` , `>` , `"` , `'` , `&`) безопасно заменены для соответствующих html-объектов.) (Выражение PHP должно оцениваться в строке, иначе будет выбрано исключение).

Повторение переменной

```
{{ $variable }}
```

Повторение элемента в массиве

```
{{ $array["key"] }}
```

Повторение свойства объекта

```
{{ $object->property }}
```

Повторение результата вызова функции

```
{{ strtolower($variable) }}
```

Проверка наличия

Обычно, в PHP, чтобы проверить, установлена ли переменная и распечатывать ее, вы должны делать

- Перед PHP 7

```
<?php echo isset($variable) ? $variable : 'Default'; ?>
```

- После PHP 7 (с использованием оператора «Null coalescing»)

```
<?php echo $variable ?? 'Default'; ?>
```

Оператор Blade `or` делает это проще:

```
{{ $variable or 'Default' }}
```

Сырые эхо

Как уже упоминалось, регулярный синтаксис двойных фигурных скобок `{{ }}` фильтруется через функцию `htmlspecialchars` PHP для обеспечения безопасности (предотвращая вредоносную инъекцию HTML в представление). Если вы хотите обойти это поведение, например, если вы пытаетесь вывести блок содержимого HTML в результате выражения PHP, используйте следующий синтаксис:

```
{!! $myHtmlString !!}
```

Обратите внимание, что рекомендуется использовать стандартный синтаксис `{{ }}` для исключения ваших данных, если это абсолютно необходимо. Кроме того, при отражении ненадежного контента (то есть контента, предоставляемого пользователями вашего сайта) вам следует избегать использования `{!! !!}` синтаксис.

Включая частичные виды

С Blade вы также можете включить частичные представления (называемые «частичные») прямо на страницу следующим образом:

```
@include('includes.info', ['title' => 'Information Station'])
```

В приведенном выше коде будет отображаться представление «views / includes / info.blade.php». Он также передается переменной `$title` со значением «Information Station».

В общем, включенная страница будет иметь доступ к любой переменной, к которой имеет доступ вызывающая страница. Например, если у нас есть:

```
{{$user}} // Outputs 'abc123'
@include('includes.info')
```

И 'includes / info.blade.php' имеет следующее:

```
<p>{{$user}} is the current user.</p>
```

Затем страница отобразит:

```
abc123
abc123 is the current user.
```

Включить каждый

Иногда вам нужно объединить оператор `include` инструкцией `foreach` и получить доступ к переменным из цикла foreach в include. В этом случае используйте директиву `@each` Blade:

```
@each('includes.job', $jobs, 'job')
```

Первый параметр - страница для включения. Второй параметр - это массив для перебора. Третий параметр - это переменная, назначенная элементам массива. Вышеприведенное утверждение эквивалентно:

```
@foreach($jobs as $job)
    @include('includes.job', ['job' => $job])
@endforeach
```

Вы также можете передать необязательный четвертый аргумент директиве `@each` чтобы указать представление, показывающее, когда массив пуст.

```
@each('includes.job', $jobs, 'job', 'includes.jobsEmpty')
```

Наложение макета

Макет - это файл вида, который расширяется другими представлениями, которые вводят блоки кода в их родительский. Например:

parent.blade.php:

```
<html>
    <head>
        <style type='text/css'>
        @yield('styling')
        </style>
    </head>
    <body>
        <div class='main'>
        @yield('main-content')
        </div>
    </body>
</html>
```

child.blade.php:

```
@extends('parent')

@section('styling')
.main {
    color: red;
}
@stop

@section('main-content')
This is child page!
@stop
```

otherpage.blade.php:

```
@extends('parent')

@section('styling')
```

```
.main {
    color: blue;
}
@stop

@section('main-content')
This is another page!
@stop
```

Здесь вы видите две примерные дочерние страницы, каждая из которых расширяет родительский элемент. На дочерних страницах определяется `@section` , которое вставлено в родительский `@yield` в соответствующий оператор `@yield` .

Таким образом, представление, представленное `View::make('child')` будет `View::make('otherpage')` **страница!** » В красном цвете, а `View::make('otherpage')` создаст тот же html, кроме как с текстом « **Это другое страница!** "вместо синего.

Обычно отдельные файлы просмотра, например, имеют папку макетов специально для файлов макета, и отдельную папку для различных конкретных индивидуальных представлений.

Макеты предназначены для применения кода, который должен отображаться на каждой странице, например, добавить боковую панель или заголовок, без необходимости записывать все html-шаблоны в каждом отдельном представлении.

Представления могут быть продлен неоднократно - т.е. **page3** может `@extend('page2')` , и **страница2** может `@extend('page1')` .

Команда расширения использует тот же синтаксис, что и для `View::make` и `@include` , поэтому к файлам `layouts/main/page.blade.php` обращается как `layouts.main.page` .

Совместное использование данных во всех представлениях

Иногда вам нужно установить одни и те же данные во многих своих представлениях.

Использование View :: share

```
// "View" is the View Facade
View::share('shareddata', $data);
```

После этого содержимое `$data` будет доступно во всех представлениях под именем `$shareddata` .

`View::share` обычно вызывается в поставщике услуг или, возможно, в конструкторе контроллера, поэтому данные будут совместно использоваться в представлениях, возвращаемых только этим контроллером.

Использование View :: composer

Просмотр композиторов - это обратные вызовы или методы класса, вызываемые при визуализации представления. Если у вас есть данные, которые вы хотите привязать к представлению каждый раз при визуализации представления, композитор представления может помочь вам организовать эту логику в одном месте. Вы можете напрямую привязать переменную к определенному виду или ко всем представлениям.

Композитор на основе Closure

```
use Illuminate\Support\Facades\View;

// ...

View::composer('*', function ($view) {
    $view->with('somedata', $data);
});
```

Композитор на основе классов

```
use Illuminate\Support\Facades\View;

// ...

View::composer('*', 'App\Http\ViewComposers\SomeComposer');
```

Как и в случае с `View::share` , лучше всего зарегистрировать композиторов в поставщике услуг.

Если вы собираетесь использовать подход класса композитора, у вас будет `App/Http/ViewComposers/SomeComposer.php` с:

```
use Illuminate\Contracts\View\View;

class SomeComposer
{
    public function compose(View $view)
    {
        $view->with('somedata', $data);
    }
}
```

Эти примеры используют `'*'` в регистрации композитора. Этот параметр представляет собой строку, которая соответствует именам представлений, для которых регистрируется композитор (`*` является подстановочным знаком). Вы также можете выбрать один вид (например, `'home'`) группы маршрутов под подпапкой (например, `'users.*'`).

Выполнить произвольный PHP-код

Хотя, возможно, было бы неправильно делать такую вещь в представлении, если вы намерены отдельно отнести проблемы, директива `php` Blade позволяет реализовать PHP-код, например, для установки переменной:

```
@php($varName = 'Enter content ')
```

(такой же как:)

```
@php
    $varName = 'Enter content ';
@endphp
```

потом:

```
{{ $varName }}
```

Результат:

> Введите содержание

Прочитайте Шаблоны кликов онлайн: https://riptutorial.com/ru/laravel/topic/1407/шаблоны-кликов

кредиты

S. No	Главы	Contributors
1	Начало работы с Laravel	alepeino, Alphonsus, boroboris, Colin Herzog, Community, Ed Rands, Evgeniy Maynagashev, Gaurav, Imam Assidiqqi, James, Ketan Akbari, Kovah, Lance Pioch, Marek Skiba, Martin Bean, Misa Lazovic, nyedidikeke, Oliver Adria, Prakash, rap-2-h, Ru Chern Chong, SeinopSys, Tatranskymedved, Tim
2	HTML и Form Builder	alepeino, Casper Spruit, Himanshu Raval, Prakash
3	Laravel Docker	Dov Benyomin Sohacheski
4	авторизация	Daniel Verem
5	Аутентификация	Aykut CAN, Imam Assidiqqi
6	База данных	A. Raza, adam, caoglish, Ian, Iftikhar uddin, Imam Assidiqqi, liamja, Panagiotis Koursaris, RamenChef, Rubens Mariuzzo, Sanzeeb Aryal, Vucko
7	Введение в laravel-5.2	A. Raza, ashish bansal, Community, Edward Palen, Ivanka Todorova, Shubhamoy
8	Введение в laravel-5.3	Ian
9	Запрос на перекрестный домен	Imam Assidiqqi, Suraj
10	Запрос формы	Bookeater, Ian, John Roca, Kyslik, RamenChef
11	Запросы	Ian, Jerodev, RamenChef, Rubens Mariuzzo
12	засеивание	A. Raza, Alphonsus, Ian, Imam Assidiqqi, Kyslik, SupFrost, whoan
13	Изменение поведения маршрутизации по умолчанию в Laravel 5.2.31 +	Frank Provost

14	Именование файлов при загрузке с помощью Laravel на Windows	Donkarnash, RamenChef
15	Инструкция по установке	Advaith, Amarnasan, aynber, Community, davejal, Dov Benyomin Sohacheski, Imam Assidiqqi, PaladiN, rap-2-h, Ru Chern Chong
16	Интеграция Sparkpost с Laravel 5.4	Alvin Chettiar
17	использовать псевдонимы полей в Eloquent	MM2
18	камердинер	David Lartey, Dov Benyomin Sohacheski, Imam Assidiqqi, Misa Lazovic, Ru Chern Chong, Shog9
19	Касса	littleswany, RamenChef
20	Класс CustomException в Laravel	ashish bansal
21	Коллекции	A. Raza, Alessandro Bassi, Alex Harris, bhill77, caoglish, Dummy Code, Gras Double, Ian, Imam Assidiqqi, Josh Rumbut, Karim Geiger, matiaslauriti, Nicklas Kevin Frank, Ozzy, rap-2-h, simonhamp, Vucko
22	Константы	Mubashar Iqbal, Oscar David, Zakaria Acharki
23	Контроллеры	Ru Chern Chong
24	красноречивый	aimme, alepeino, Alessandro Bassi, Alex Harris, Alfa, Alphonsus, andretzermias, andrewtweber, Andrey Lutskevich, aynber, Buckwheat, Casper Spruit, Dancia, Dipesh Poudel, Ian, Imam Assidiqqi, James, James, jedrzej.kurylo, John Slegers, Josh Rumbut, Kaspars, Ketan Akbari, KuKeC, littleswany, Lykegenes, Maantje, Mahmood, Marco Aurélio Deleu, marcus.ramsden, Marek Skiba, Martin Bean, matiaslauriti, MM2, Nicklas Kevin Frank, Niklas Modess, Nyan Lynn Htut, patricus, Pete Houston, Phroggyy, Prisoner Raju, RamenChef, rap-2-h, Rubens Mariuzzo , Sagar Naliyapara, Samsquanch, Sergio Guillen Mantilla, Tim, tkausl, whoan, Yasin Patel

25	Красноречивый: Аксессоры и мутаторы	Diego Souza, Kyslik
26	Красноречивый: Модель	Aeolingamenfel, alepeino, Alex Harris, Imam Assidiqqi, John Slegers, Kaspars, littleswany, Marco Aurélio Deleu, marcus.ramsden, Marek Skiba, matiaslauriti, Nicklas Kevin Frank , Samsquanch, Tim
27	Красноречивый: Отношения	Advaith, aimme, Alex Harris, Alphonsus, bhill77, Imam Assidiqqi, Ketan Akbari, Phroggyy, rap-2-h, Ru Chern Chong, Zulfiqar Tariq
28	Макросы в красноречивых отношениях	Alex Casajuana, Vikash
29	маршрутизация	A. Raza, alepeino, Alessandro Bassi, Alex Juchem, beznez, Dwight, Ilker Mutlu, Imam Assidiqqi, jedrzej.kurylo, Kyslik, Milan Maharjan, Rubens Mariuzzo, SeinopSys, Vucko
30	Миграции баз данных	Chris, Chris White, Hovsep, hschin, Iftikhar uddin, Imam Assidiqqi, Kaspars, liamja, littleswany, mnoronha, Nauman Zafar, Panagiotis Koursaris, Paulo Freitas, Vucko
31	Монтаж	A. Raza, alepeino, Alphonsus, Black, boroboris, Gaurav, Imam Assidiqqi, James, Ketan Akbari, Lance Pioch, Marek Skiba, Martin Bean, nyedidikeke, PaladiN, Prakash, rap-2-h, Ru Chern Chong, Sagar Naliyapara, SeinopSys, Tim
32	наблюдатель	matiaslauriti, Szenis
33	Начало работы с laravel-5.3	A. Raza, Advaith, Community, davejal, Deathstorm, Manish, Matthew Beckman, Robin Dirksen, Shital Jachak
34	Несколько соединений DB в Laravel	4444, A. Raza, Rana Ghosh
35	Обработка ошибок	Isma, Kyslik, RamenChef, Rubens Mariuzzo
36	Общие проблемы и быстрые исправления	Nauman Zafar
37	Основы Cron	A. Raza
38	Очереди	Alessandro Bassi, Kyslik

39	Ошибка несоответствия токена в AJAX	Pankaj Makwana
40	пагинация	Himanshu Raval, Iftikhar uddin
41	Пакеты Laravel	Casper Spruit, Imam Assidiqqi, Ketan Akbari, rap-2-h, Ru Chern Chong, Tosho Trajanov
42	Планирование заданий	Jonathon
43	Полезные ссылки	Jakub Kratina
44	полисы	Tosho Trajanov
45	Помощники	aimme
46	Посев базы данных	Achraf Khouadja, Andrew Nolan, Dan Johnson, Isma, Kyslik, Marco Aurélio Deleu
47	почта	Yohanan Baruchel
48	Проверка	A. Raza, alepeino, Alessandro Bassi, Alex Harris, Andrew Nolan, happyhardik, Himanshu Raval, Ian, Iftikhar uddin, John Slegers, Marco Aurélio Deleu, matiaslauriti, rap-2-h, Rubens Mariuzzo, Safoor Safdar, Sagar Naliyapara, Stephen Leppik, sun, Vucko
49	Промежуточное	Alex Harris, Kaspars, Kyslik, Moppo, Pistachio
50	просвет	maksbd19
51	Разверните приложение Laravel 5 на общем хостинге на сервере Linux	Donkarnash, Gayan, Imam Assidiqqi, Kyslik, PassionInfinite, Pete Houston, rap-2-h, Ru Chern Chong, Stojan Kukrika, ultrasamad
52	Разрешения для хранения	A. Raza
53	ремесленник	Alessandro Bassi, Gaurav, Harshal Limaye, Himanshu Raval, Imam Assidiqqi, Kaspars, Laurel, Rubens Mariuzzo, Safoor Safdar, Sagar Naliyapara, SeinopSys
54	Светская	Jonathon, Marco Aurélio Deleu
55	Связывание	A. Raza, GiuServ, Vikash

	модели маршрута	
56	Сервисы	A. Raza, El_Matella
57	События и слушатели	Bharat Geleda, matiaslauriti, Nauman Zafar
58	Структура каталога	Kaspars, Moppo, RamenChef
59	тестирование	Alessandro Bassi, Brayniverse, caoglish, Julian Minde, Kyslik, rap-2-h, Sven
60	Удалить публикацию из URL-адреса в laravel	A. Raza, Rana Ghosh, ultrasamad
61	Файловая система / облачное хранилище	Imam Assidiqqi, Nitish Kumar, Paulo Laxamana
62	Функция пользовательского помощника	Ian, Luceos, rap-2-h, Raunak Gupta
63	Шаблоны кликов	A. Raza, agleis, Akshay Khale, alepeino, Alessandro Bassi, Benubird, cbaconnier, Christophvh, Imam Assidiqqi, matiaslauriti, Nauman Zafar, rap-2-h, Safoor Safdar, Tosho Trajanov, yogesh