آموزش فریم ورک لاراول

لاراول (به انگلیسی Laravel) یک چارچوب کاری(framework)متن باز پی اچ پی (PHP) است که توسط تیلور اوتول، برای توسعه نرم‌افزارهای پی اچ پی وب بر پایهٔ معماری MVC طراحی شده است. لاراول تحت مجوز MIT در یک مخزن روی گیتهاب (گیت هاب) توسعه و پشتیبانی می‌شود. به گفتهٔ توسعه دهنگان در دسامبر سال ۲۰۱۴ و ۲۰۱۳ به عنوان محبوب ترین چارچوب پی اچ پی یاد شده است. لاراول به بیانی ساده یک ابزار کاربردی برای توسعه دهندگان پی اچ پی است تا کدهای خود را سریعتر و بهینه تر ایجاد نمایند، هم اکنون کاربران زیادی در یاد گرفتن این فریم ورک چه در ایران و چه در جهان هستند.

این کتاب شامل آموزش لاراول و پروژه عملی ساخت وبلاگ ساده و ساخت فرم تماس با ما است

فهرست

نصب فریم ورک لاراول

قبل از اینکه بخواهید فریمورک لاراول ۵ رو نصب کنید باید مطمئن باشید که extension های زیر روی سرورتان نصب باشد و ورژن PHP سرور هم باید ۵,۴ یا بیشتر باشد

Mcrypt

OpenSSL

Mbstring

Tokenizer

برای اطلاع از فعال بودن این extension ها و همچنین نسخه php روی سیستم می تونید با استفاده از دستور phpinfo() به این اطلاعات دست پیدا کنید و در صورت عدم نصب هرکدام با توجه به سیستم عاملتون اقدام به نصب و فعال کردن آنها بکنید.

بهترین راه نصب لاراول ۵ استفاده از composer است که در صورت نصب نبودن روی سیستم تان می توانید از اینجا آن را دریافت و نصب کنید.

ترمینال رو توی لینوکس یا cmd رو توی ویندوز باز کنید و ابتدا به دایرکتوری که میخواهید فریمورک رو داخلش نصب کنید (پوشه root نرم افزار شبیه ساز سروتان مثل xampp یا lamp و یا wamp) بروید مثلا با یکی از دستورات زیر که البته ممکن است مکان پوشه root در سیستم شما متفاوت باشد:

```
// for linux ubuntu
cd /var/www/html
 //for windows and xampp
cd c:\xampp\htdocs
//for windows and wamp
cd c:\wamp\www
```

حالا می تونید با تایپ دستور زیر توی ترمینال آخرین نسخه لاراول رو دانلود و نصب کنید که یک پوشه به نام laravel ساخته میشود:

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

نکته : افرادی که از لینوکس استفاده می کنند باید به پوشه های storage و vendor مجوز نوشتن فایل رو بهش بدهید

در صورتی که composer در سیستم شما نصب نمی شود یا مشکلی دارد میتوانید فایل های فریمورک لاراول را از آدرس زیر دریافت و در مسیر پوشه root سرورتان extract کنید:
http://fian.my.id/larapack/

پیکربندی لاراول

توی پوشه اصلی لاراول یک فایل به نام env. وجود دارد که می توانید تنظیمات برنامه تان و دیتابیس پروژه را در اینجا تعیین کنید :

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=zGKCjTPbzET3WiHhKCxSpTBNCuUVWWLc

DB_HOST=localhost
DB_DATABASE=learninglaravel
DB_USERNAME=root
DB_PASSWORD=secret
```

به طور مثال اگر APP_DEBUG را روی true ست کنید خطاهای برنامه نویسی در هنگام کدنویسی برایتان قابل مشاهده خواهد بود و مناسب برای حالت development هست و در هنگام آپلود سایت روی هاست آن را fasle قرار دهید.
بهتره مقدار APP_KEY را هم با تایپ دستور زیر در ترمینال تغییر دهیم:

```
php artisan key:generate
```

سایر تنظیمات رو هم میتونید در پوشه config در فایل مورد نظرش اعمال کنید. به طور مثال می توانید در فایل app.php مقدار timezone رو به Asia/Tehran تغییر دهید.

شما می توانید داخل فایل app.php در پوشه config تنظیمات برنامه را اعمال کنید. تنظیمات به صورت یک جفت کلید/مقدار هستند. بعضی از آیتم ها مقدار خودشان را ا توسط تابع کمکی env از فایل .env واقع در دابرکتوری root پروژه که در پست قبلی توضیح دادم می گیرند به طور مثال:

```
'debug' => env('APP_DEBUG'),

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

debug و key مقدار خودش رو از فایل .env می گیرند در صورتی که در فایل .env برایشان مقداری ست نکرده باشیم می توانیم به تابع env() پارامتر دومی بدهیم که نشانگر مقدار آن هست. در مثال بالا keyبه این صورت است و اگر در فایل .env آن را حذف کنیم از این مقدار پیش فرض استفاده خواهد کرد.

در زیر توضیح مختصری برای هر آیتم آن میدهم:

debug: اگر مقدار آن را true ست کنید برنامه در مد development خواهد بود و خطاهای برنامه نشان داده می شود و اگر false باشد در مد production می باشد و مناسب برای publish و استفاده نهایی برنامه هست.

url: آدرس url پروژه را در اینجا ست میکنیم مثلاhttp://localhost/laravel/public

timezone: موقعیت زمانی را مشخص می کنیم که برای مثال در کشور ایران Asia/Tehran ست می کنیم.

locale: در مسیر resources/lang می توانیم یک پوشه دیگر به نام fa ایجاد کرده تا در آن پیغام ها و متون فارسی را تایپ کنیم تا در برنامه از آنها استفاده کنیم. به طور مثال یک کاربرد آن در فارسی سازی پیغام های اعتبارسنجی فرم ها می باشد. مقدار این آیتم را fa که همنام آن پوشه که ایجاد کردیم ست میکنیم.

fallback_locale: در صورتی که locale موردنظر برای آن رشته موجود نبود از این locale استفاده شود.

key: کلید برنامه که یک رشته تصادفی هست و در رمزنگاری های برنامه توسط لاراول مورد استفاده قرار می گیرد. نحوه ست کردن آن را در پست قبلی توضیح دادم.

...

سایر موارد را در جای مناسب خودش توضیح خواهم داد.

لاراول ۵ به طور پیش فرض از دایرکتوری app تحت namespace ای به نام App استفاده میکند که هنگام ایجاد کلاس هایتان از آن استفاده میکنید که شما می توانید با استفاده از دستور زیر و تایپ در ترمینال آن فضای نام را به نام دلخواهتان تغییر دهید مثلا در مثال زیر من آن را به Hamo تغییر دادم:

```
php artisan app:name Hamo
```

بعد از اجرای این دستور لاراول به طور خودکار تمام namespace های استفاده شده در کلاس هایتان را به نام جدید تغییر خواهد داد.

دسترسی به مقادیر پیکر بندی:

با استفاده از کلاس Config هم می توانید مقادیر config رو با استفاده از متد get بدست بیارید یا مقدار جدیدی را با استفاده از متد set ست کنید به مثال های زیر توجه کنید:

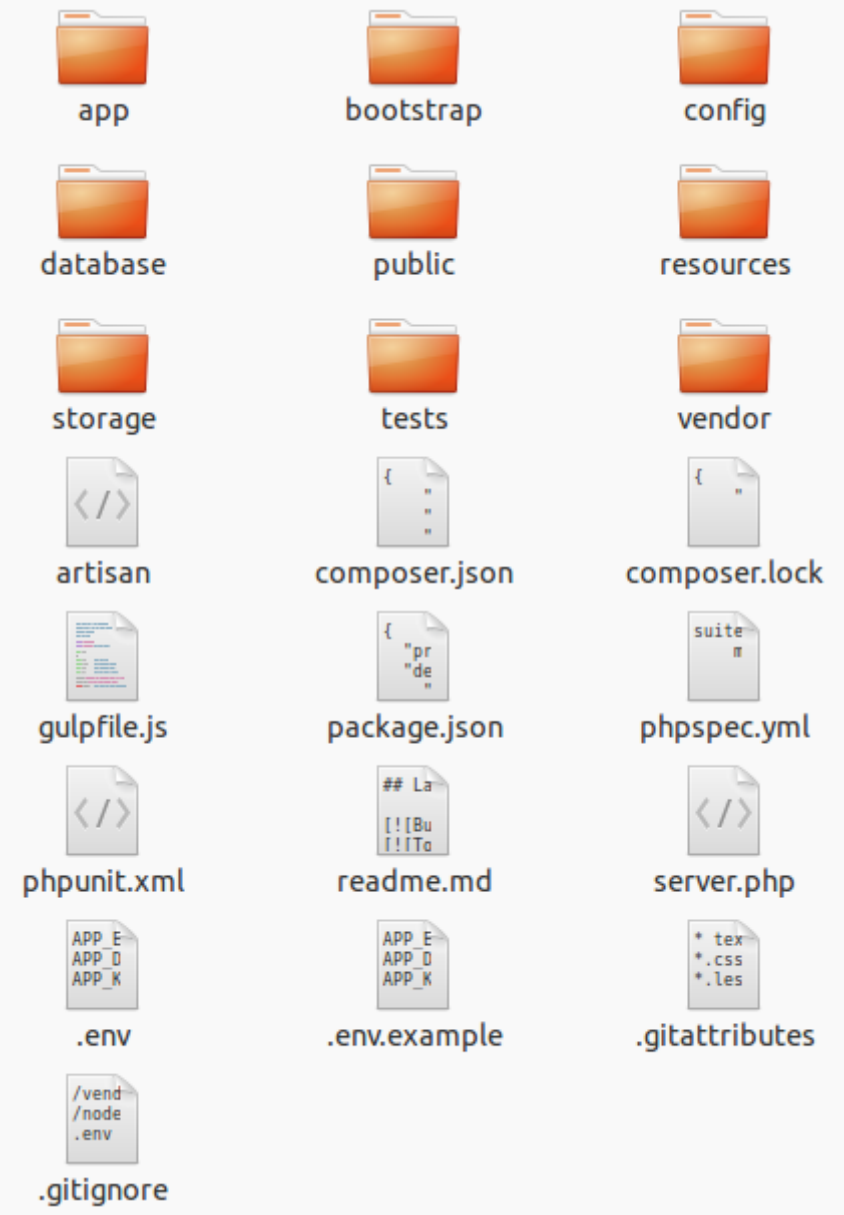
```
$value = Config::get('app.timezone');

Config::set('app.timezone', 'Asia/Tehran');
```

همچنین می توانید از تابع کمکی config هم استفاده کنید :

```
$value = config('app.timezone');
```

ساختار برنامه در لاراول

در این قسمت قصد دارم در مورد ساختار دایرکتوری ها و فایل های موجود در فریمورک لاراول ۵ توضیحات مختصری رو ارائه کنم.



در بالا تصویری از دایرکتوری root لاراول قرار دادم. در زیر درمورد آنها توضیحاتی می دهم:

app: این دایرکتوری حاوی تمام کدهای برنامه تان از جمله کنترلرها و مدل های برنامه تان هست. با این دایرکتوری زیاد سروکار خواهیم داشت.

Bootstrap: این دایرکتوری حاوی یک سری فایل برای autoloading و راه اندازی فریمورک هست.

Config: حاوی تمام فایل های پیکربندی برنامه تان است.

database: حاوی فایل های migration و seed است.

public: فایل های استاتیک و front-end برنامه تان از قبیل javascript , css, images در اینجا قرار میگرند.

Resources: در این دایرکتوری فایل های view برنامه و فایل های loacale و زبان در آن قرار می گیرند.

storage: در این دایرکتوری فایل هایی که توسط موتور پوسته blade کامپایل می شوند و همچنین مکان ذخیره سازی فایل های سشن و کش و سایر فایل هایی که توسط فریمورک ایجاد می شوند می باشد.

Test: حاوی فایل های تست خودکار برنامه است.

vendor: حاوی تمام third-party ها و وابستگی هایی که توسط composer به برنامه اضافه می شوند هست.

داخل دایرکتوری app می توانید مدل ها را ایجاد کنید و همچنین در مسیر app/Http/controllers می توانید کنترلرهای برنامه را ایجاد کنیم و همچنین فایل routes.php که در مسیر app/Http قرار دارد که مدیریت مسیرها از آن استفاده میکنیم ازجمله فایل ها و دایرکتوری های پرکاربرد ما در این فریمورک هستند.

فایل های view برنامه را هم در مسیر resources/views قرار می دهیم. در قسمت های بعدی نحوه مسیردهی و ایجاد کنترلر و ویو ها را خواهیم آموخت.

برای اطلاعات بیشتر می توانید به اینجا مراجعه کنید

Routing در لاراول

از مزیت های فریمورک لاراول نسبت به سایر فریمورک های PHP مبحث Routing آن است که می توان مدیریت خوبی روی مسیرها داشت. در مسیر app/Http و فایل routes.php می توانیم تمامی مسیرهای برنامه را در آنجا تعریف و مدیریت کنیم.این فایل توسط کلاس App\Providers\RouteServiceProvider بارگزاری میشود.

یک مثال ساده:

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

کلاس Route چند متد دارد که نوع درخواست http را مشخص میکند. در مثال بالا متد get فقط در خواست های GET به این مسیر را قبول میکند. سایر متدها که نوع درخواست http را مشخص میکنند post , put, patch, delete می باشند. این متد دوتا پارامتر می گیرد که اولی مسیری است که بعد از نام دامنه سایت می آید مثلا در آدرس http://www.example.com/about مسیری که وارد میکنیم about است .

در پارامتر دومی هم می توانیم بدون استفاده از کنترلر و اکشن و با دادن یک تابع بی نام در همین روتر آن را مدیریت کنیم.

کلاس Route دارای متد دیگری به نام match هست که می توانیم چند نوع درخواست http را به یک مسیر مجاز کنیم در مثال زیر مسیر هردونوع درخواست GET و POST را قبول می کند :

```
Route::match(['get', 'post'], '/', function()
{
    return 'Hello World';
});
```

در صورتی که بخواهیم مسیر همه در خواست ها را قبول کنید از متد any استفاده میکنیم مثلا آدرس http://www.example.com/foo هر درخواستی را قبول میکند :

```
Route::any('foo', function()
{
    return 'Hello World';
});
```

درخواست های HTML , DELETE یا PATCH را پشتیبانی نمی کند برای اینکه یک فرم HTML را با این متدها تعریف کنیم کافیه یک تگ input از نوع hidden و با نام _method تعریف میکنیم و به value آن یکی از مقادیر PUT, DELETE, PATCH را بدهید مثلا:

```
<form action="/foo/bar" method="POST">
    <input type="hidden" name="_method" value="PUT">
    <input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
</form>
```

در مثال بالا آدرس http://www.example.com/foo/bar در روتر با متد put قابل دریافت است که می توانیم برای DELETE , PATCH هم به همین صورت عمل کنیم. کاربرد این متدها را در بخش کنترلر ها تشریح خواهم کرد. همچنین یک تگ از نوع مخفی به نام _token هم در فرم وجود دارد که در یک پست جداگانه در مورد فرم ها و کار با آنها توضیح خواهم داد.

مسیر با پارامتر

به همراه مسیر می توانیم هر تعداد پارامتر را هم ارسال کنیم فقط کافی است نام پارامترها را داخل آکولاد قرار دهیم. به مثال های زیر توجه کنید:

```
Route::get('user/{id}', function($id)
{
    return 'User '.$id;
});

Route::get('user/{name?}', function($name = null)
{
    return $name;
});

Route::get('user/{name?}', function($name = 'John')
{
    return $name;
});
```

در مثال های بالا همانطور که مشاهده کردید می توانیم برای پارامترها یک مقدار پیش فرض یا null هم در نظر گرفت تا درصورت وارد نکردن مقداری برای پارامتر در url خطایی ایجاد نشود. همچنین باید جلوی نام پارامتر های اختیاری یک علامت ? قرار دهیم.

افزودن عبارت منظم به پارامترها

می توانیم با افزودن متد where به انتهای متد get برای هر پارامتر یک عبارت منظم هم تعریف کرد تا مثلا id فقط مقدار عدد مورد قبول باشد. در صورتی که چند پارامتر را بخواهیم برایش عبارت منظم تعریف کنیم آنها را داخل آرایه قرار می دهیم.

```
Route::get('user/{id}/{name}', function($id, $name)
{
    //
})
->where(['id' => '[0-9]+', 'name' => '[a-z]+'])
```

همچنین می توانیم برای یک پارامتر خاص در کل برنامه یک عبارت منظم عمومی تعریف کنیم به این صورت که در کلاس RouteServiceProvider در دایرکتوری app/Providers در متد boot این عبارت را قرار دهیم مثلا در مثال زیر کاربر در routing هرجایی از پارامتر id استفاده کرد فقط مجاز به دادن مقدار عددی به آن است و دیگر مانند بالا نیاز به تعریف متد where نیست :

```
$router->pattern('id', '[0-9]+');
```

مسیردهی به یک کنترلر و اکشن

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

در پارامتر دوم فقط کافی است بین نام کلاس کنترلر و اکشن یک علامت @ قرار دهیم.

نامگزاری مسیر

با استفاده از کلمه as می توانیم برای مسیر یک نام هم تعریف کنیم و همچنین با استفاده از uses می توانیم آن را به اکشن و کنترلر خاصی هدایت کنیم.

```
Route::get('user/profile', [
    'as' => 'profile', 'uses' => 'UserController@showProfile'
]);
```

از کاربردهای نامگزاری مسیر برای ایجاد و ساختن url است که می توانیم با استفاده از تابع کمکی route نام مسیر را به آن بدهیم مثلا در مثال بالا با دادن نام profile آدرس http://www.example.com/user/profile ایجاد خواهد شد و همچنین برای ریدایرکت به یک مسیر هم کاربرد دارد.

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

$redirect = redirect()->route('profile');
```

مسیردهی گروهی

در لاراول می توانیم یک دسته از مسیرها را که مثلا در یک قسمت از url خود مشترک هستند یا middleware مشترکی دارند و یا دارای یک namespace مشترک هستند را در یک گروه قرار دهیم.
همجنین می توانیم sub-domain ها را از این طریق مدیریت کنیم.

```
Route::group(['prefix' => 'admin'], function()
{
    Route::get('users', function()
    {
        // Matches The "/admin/users" URL
    });
})
```

در مثال بالا تمامی مسیرهایی که با admin شروع می شوند را داخل این گروه قرار می دهیم.

برای اطلاعات و مثال های بیشتر در این مورد می توانید به اینجا مراجعه کنید.

Middleware ها در لاراول

middlewareها یک مکانیسم ساده ای را برای فیلتر کردن درخواست های http ورودی به برنامه تان تدارک می بیند. به طور مثال لاراول یک (iddlewareترجمه فارسیش میشه میان افزار) برای احراز هویت کاربران دارد و در صورتی که کاربری Login نکرده باشد و احراز هویت نشده باشد میان افزار آن را به صفحه لاگین هدایت میکند وگرنه میان افزار به درخواست اجازه ادامه کارش را میدهد.

middleware ها در دایرکتوری app/Http/Middleware قرار میگیرند.

تعریف یک middleware

با تایپ دستور make:middleware در ترمینال می توانیم یک میان افزار جدید ایجاد کنیم. در مثال زیر میان افزار OldMiddleware را ایجاد کردیم.

```
php artisan make:middleware OldMiddleware
```

فایل ایجاد شده را باز میکنیم و در متد handle شرط زیر را قرار میدهیم به این صورت که درخواست ورودی به نام age اگر کوچکتر از ۲۰۰ بود به صفحه home ریدایرکت شود وگرنه به درخواست اجازه ادامه کار بدهد.

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

class OldMiddleware {

    /**
     * Run the request filter.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if ($request->input('age') < 200)
        {
            return redirect('home');
        }

        return $next($request);
    }

}
```

اکنون برای اینکه بخواهیم از این میان افزار استفاده کنیم ابتدا باید آن را در فایل app/Http/Kernel.php ثبت کنیم. اگر می خواهید این میان افزار برای هر درخواست http برنامه تان اجرا شود آن را به آرایه $middleware اضافه کنید که بعد از این هردرخواستی با این نام را فیلتر خواهد کرد.

اگر می خواهید میان افزار فقط به یک مسیر خاص اعمال شود ابتدا باید آن را به آرایه $routeMiddleware اضافه کنید به این صورت که کلید آن در آرایه نام خلاصه آن برای استفاده در برنامه به کار می رود :

```
protected $routeMiddleware = [
        'auth' => 'App\Http\Middleware\Authenticate',
        'auth.basic' =>
'Illuminate\Auth\Middleware\AuthenticateWithBasicA  uth',
        'guest' => 'App\Http\Middleware\RedirectIfAuthenticated',
        'old' => 'App\Http\Middleware\OldMiddleware',
    ];
```

حالا می تونید میان افزار را به هر مسیری در فایل routing.php مانند مثال های زیر اضافه کنید که دوتا میان افزار old و auth را به مسیرهای موردنظرمان افزودیم :

```
Route::post('url/create', ['middleware' => 'old',
'uses'=>'UrlController@create']);

Route::get('admin/profile', ['middleware' => 'auth', function()
{
    //
}]);
```

Before / _After_ Middleware

همچنین می توانیم میان افزارهای خاصی را ایجاد کنیم که قبل یا بعد از مدیریت درخواست توسط برنامه عملی را اجرا کنند.

برای اطلاعات بیشتر به اینجا مراجعه کنید

کار با کنترلرها

یکی از سه عنصر اصلی الگوی طراحی MVC کنترلرها هستند. در فایل routing.php می توانیم درخواست ها را به یک کنترلر و اکشن خاصی ارسال کنیم به طور مثال آدرس
http://www.example.com/user/5

را در مثال زیر به کنترلر UserController و اکشن showProfile هدایت می کند.

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

تعریف کنترلر : کترلر ها در مسیر دایرکتوری app/Http/Controllers قرار می گیرند.

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

use App\Http\Controllers\Controller;

class UserController extends Controller {

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

}
```

کنترلرها و فضای نام(namespace)

برای هرکلاس باید namespace آن را تعریف کنیم که این فضای نام در واقع مسیر قرارگیری کلاس از پوشه app می باشد و برای کنترلرها App\Http\Controllers تعریف می کنیم. در صورتی که داخل دایرکتوری Controllers یک دایرکتوری دیگر مثلا به نام Auth ایجاد کرده باشیم و کنترلری در آن تعریف کنیم فضای نام به صورت namespace App\Http\Controllers\Auth می باشد.

نکته : همیشه نام کلاس های کنترلر را به صورت PascalCase و در انتهای آن کلمه Controller را بیاورید. بهتر است اکشن ها را هم به صورت camelCase نامگزاری کنید.

البته من خودم همیشه عادت دارم کلاس های کنترلر و مدل را با ترمینال ایجاد کنم که شما هم می توانید با این دستور یک کنترلر بدون هیچ متدی ایجاد کنید:

```
php artisan make:controller UserController -plain
```

استفاده از middleware در کنترلر

همانطور که در پست قبلی توضیح دادم می توانیم برای هر مسیر خاص یک کلاس میان افزار اضافه کنیم تا درخواست ها فیلتر شوند. مثلا در مثال زیر برای مسیر میان افزار auth را اضافه کردیم:

```
Route::get('profile', [
    'middleware' => 'auth',
    'uses' => 'UserController@showProfile'
]);
```

در مثال زیر همانطور که مشاهده می کنید سه مثال از استفاده از میان افزار در کنترلرها را آورده است که در متد سازنده کلاس هم قرار می گیرند :

```
class UserController extends Controller {

    /**
     * Instantiate a new UserController instance.
     */
    public function __construct()
    {
        $this->middleware('auth');

        $this->middleware('log', ['only' => ['fooAction', 'barAction']]);

        $this->middleware('subscribed', ['except' => ['fooAction',
'barAction']]);
    }

}
```

در مثال دوم میان افزار را با استفاده از کلمه only فقط به اکشن های fooAction و barAction محدود کردیم و فیلتر فقط به این اکشن ها اعمال شود و در مثال سوم با استفاده از کلمه except میان افزار به همه اکشن ها اعمال شود به جز اکشن های fooAction و barAction.

در لاراول همچنین می توانیم به مسیردهی به یک اکشن را به صورتی ساده تر هم انجام دهیم مثلا با تعریف مسیر به این صورت:

```
Route::controller('users', 'UserController');
```

با افزودن درخواست http به ابتدای نام اکشن با توجه به نوع درخواست به اکشن مورد نظر تحویل داده می شود:

```
class UserController extends BaseController {

    public function getIndex()
    {
        //
    }

    public function postProfile()
    {
        //
    }

    public function anyLogin()
    {
        //
    }

}
```

نکته : اگر می خواهید برخی از مسیرها را نامگزاری کنید کافیست پارامتر سومی هم به صورت آرایه در نظر بگیرید و کلید آرایه نام اکشن و مقدار آن نام مسیر باشد:

```
Route::controller('users', 'UserController', [
    'anyLogin' => 'user.login',
]);
```

کنترلرهای RESTful

در لاراول می توانیم با دستور زیر در ترمینال کنترلرهایی با اکشن های خاصی ایجاد کنیم که هر اکشن یک مسیر و درخواست http را تحویل میگیرند. به طور مثال کنترلر PhotoController را ایجاد می کنیم:

```
php artisan make:controller PhotoController
```

مسیر را هم به این صورت در فایل routes.php تعریف می کنیم:

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

حالا اگر url را به صورت http://www.example.com/photo بنویسیم اکشن index درخواست را دریافت میکند. در تصویر زیر می توانید اطلاعات کاملی را از تمام اکشن ها داشته باشید. verb نوع درخواست http و path مسیری که در url وارد میکنیم و action اکشنی که این درخواست را دریافت میکند و route name هم نام مسیر می باشد.

Verb	Path	Action	Route Name
GET	/photo	index	photo.index
GET	/photo/create	create	photo.create
POST	/photo	store	photo.store
GET	/photo/{photo}	show	photo.show
GET	/photo/{photo}/edit	edit	photo.edit
PUT/PATCH	/photo/{photo}	update	photo.update
DELETE	/photo/{photo}	destroy	photo.destroy

همچنین می توانیم فقط اکشن های خاصی را به صورت RESTful تعریف کنیم:

```
Route::resource('photo', 'PhotoController',
                ['only' => ['index', 'show']]);

Route::resource('photo', 'PhotoController',
                ['except' => ['create', 'store', 'update', 'destroy']]);
```

کار با view ها

viewها را در مسیر resources/views قرار می دهیم. شما می توانید آنها را با استفاده از موتور قالب Bladeو یا به صورت معمولی ایجاد کنید. در مثال زیر فایل greeting.php را در مسیر ذکر شده قرار می دهیم و در آن دستورات زیر را قرار میدهیم:

```
<!-- View stored in resources/views/greeting.php -->

<html>
  <body>
    <h1>Hello, <?php echo $name; ?></h1>
  </body>
</html>
```

با استفاده از تابع کمکی view هم می توانیم فایل ویو را render کنیم. این تابع دو پارامتر می گیرد که اولی نام فایل ویو موردنظر بدون قرار دادن فرمت آن و دومین پارامتر آرایه ای از داده هایی هست که به فایل ویو می فرستیم.کلید آرایه در فایل ویو به صورت نام متغیر قابل استفاده است. در مثال زیر کاربر با وارد کردن آدرس http://www.example.com به او Hello, James نمایش داده می شود.

```
Route::get('/', function()
{
  return view('greeting', ['name' => 'James']);
});
```

در صورتی که فایل ویو داخل یک دایرکتوری باشد کافی است نام دایرکتوری و فایل را با یک نقطه از هم جدا کنید:

```
return view('admin.profile', $data);
```

در مثال فوق فایل ویو در مسیر resources/views/admin/profile.php قرار دارد.

همچنین به روش های زیر هم می توانیم داده را به ویو ارسال کنیم:

```
// Using conventional approach
$view = view('greeting')->with('name', 'Victoria');


// Using Magic Methods
$view = view('greeting')->withName('Victoria');
```

متد with دو پارامتر میگیرد که اولی نام متغیر و دومی مقدار آن هست. همچنین می توانید به روش دوم که در انتهای متد with نام متغیر را اضافه و مقدارش را به عنوان پارامتر به آن می دهیم.

بررسی وجود فایل view

```
if (view()->exists('emails.customer'))
{
  //
}
```

رندر کردن view از طریق مسیر فایل

```
Route::get('/', function(){
  return view()->file('/var/www/html/laravel/public/greeting.php', ['name' => 'James']);
});
```

همانطور که می بنید کاربرد آن برای مواقعی است که شما فایل view که خارج از مسیر resourrces/viewsتعریف کرده اید را بتوانید رندر کنید. در مثال بالا من فایل ویو را در پوشه public ایجاد کردم.

برای اطلاعات تکمیلی کار با view ها به اینجا مراجعه کنید

درخواست های HTTP

در لاراول درخواست های http که با متدهای ... ,GET , POST ارسال می کنیم را می توانیم مقادیر آنها را با استفاده از کلاس Request دریافت کنیم:

```
$name = Request::input('name');
```

نکته : برای استفاده از هر کلاسی در کلاس های کنترلر ابتدا باید آن کلاس را با استفاده از دستور useایمپورت کنیم. در مثال بالا هم بایستی به این صورت قبل از تعریف کلاس کنترلر موردنظر کلاس Request را ایمپورت کنیم .

```
use Request;
```

همچنین می توانیم به روش دیگری هم مقادیر را به دست بیاوریم. به این صورت که ابتدا کلاس Illuminate\Http\Requestرا به کنترلر مورد نظر ایمپورت می کنیم سپس دستور Request $request را به عنوان پارامتر به اکشن مورد نظر می دهیم. در طول برنامه داخل اکشن می توانیم از متغیر $requestاستفاده کنیم.

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

use Illuminate\Http\Request;
use Illuminate\Routing\Controller;

class UserController extends Controller {

  /**
   * Store a new user.
   *
   * @param  Request  $request
   * @return Response
   */
  public function store(Request $request)
  {
```

```
        $name = $request->input('name');

        //
    }

}
```

می توانیم برای یک ورودی مقداری پیش فرض هم تعیین کنیم تا در صورتی که مقداری برای آن ست نشده بود این مقدار جایگزین آن شود:

```
$name = Request::input('name', 'Sally');
```

با استفاده از متد has می توانیم بررسی کنیم که آیا ورودی با این مقدار وجود دارد یا خیر:

```
if (Request::has('name'))
{
    //
}
```

با استفاده از متد all می توانیم تمامی ورودی ها را دریافت کنیم.

```
$input = Request::all();
```

همچنین می توانیم فقط برخی ورودی ها یا همه ورودی ها به جز برخی را دریافت کنیم.

```
$input = Request::only('username', 'password');

$input = Request::except('credit_card');
```

هنگامی که مقدار ورودی یک آرایه باشد می توانیم با استفاده از نقطه به مقدار آیتم مورد نظر دست پیدا کرد:

```
$input = Request::input('products.0.name');
```

همچنین می توانیم به مقادیر flash که توسط سشن ایجاد می شوند و به صورتی هستند که فقط برای درخواست بعدی معتبر هستند و از بین می روند هم به صورت های زیر دسترسی داشته باشیم:

```
Request::flash();

Request::flashOnly('username', 'email');


Request::flashExcept('password');
```

در مثال دوم و سوم هم مثل قبل که دیدیم فقط یا به جز برخی موارد دسترسی داریم.

می توانیم مقادیر ورودی ها را دوباره با استفاده flash به صفحه قبلی یا صفحه دیگری ارسال کنیم:

```
return redirect('form')->withInput();

return redirect('form')->withInput(Request::except('password'));
```

کاربرد آن در فرم ها می باشد که اگر بعد از اعتبارسنجی ورودی ها دارای خطایی باشد و بخواهیم دوباره به صفحه فرم بازگردیم ورودی های فرم که کاربر نوشته از بین نروند. در مثال دوم به اجازه password

حفظ شدن ندادیم.

برای چاپ مقادیر قبلی هم باید داخل تکست باکس های فرم مقدارش را به این صورت چاپ کنیم:

```
<input type="text" name="email" value="<?php echo old('name') ?>">
```

کوکی ها

می توانیم به مقدار یک کوکی هم به این صورت دسترسی داشته باشیم:

```
$value = Request::cookie('name');
```

فایل ها

فایلی که آپلود شده را می توانیم به این صورت اطلاعاتش دریافت کنیم. در مثال زیر نام فیلد فایل در فرم photo بوده است:

```
$file = Request::file('photo');
```

در مثال زیر بررسی می کند که آیا این فایل با این نام وجود دارد:

```
if (Request::hasFile('photo'))
{
  //
}
```

مقداری که متد file در کلاس Request به ما می دهد یک آبجکت از کلاس Symfony\Component\HttpFoundation\File\UploadedFile که می توانید با متدهای آن برای کار با فایل کار کنید.

```
if (Request::file('photo')->isValid())
{
  //
}
```

در مثال بالا بررسی می کند که آیا فایل آپلود شده صحیح و بدون خطا می باشد:

با استفاده از متد move می توانیم فایل را به مسیر مورد نظر که به عنوان پارامتر اول به آن می دهیم و همچنین نام فایل که اختیاری است ذخیره کنیم.

```
Request::file('photo')->move($destinationPath);
```

```
Request::file('photo')->move($destinationPath, $fileName);
```

سایر اطلاعات را می توانید در اینجا مشاهده کنید

پاسخ های HTTP

پاسخ ساده

بعد از دریافت درخواست و انجام عملیات مورد نظر باید پاسخی هم ایجاد کنیم. ساده ترین نوع پاسخ returnرشته هست که قبلا هم با آن آشنا شدیم:

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

ایجاد پاسخ دلخواه

با استفاده از کلاس Response یا تابع کمکی response می توانیم یک پاسخ دلخواه ایجاد کنیم مثلا مثال زیر را در نظر بگیرید:

```
return response($content, $status)
        ->header('Content-Type', $value);
```

محتویات را به عنوان پارامتر اول و status code را به عنوان پارامتر دوم به آن بدهیم و همچنین با استفاده از متد header نوع هدر را هم مشخص کنیم مثلا.application/pdf

همینطور که در مثال زیر می بینید می توانید یک فایل ویو و همچنین یک فایل کوکی را هم به عنوان پاسخ ارسال کنید و استفاده از متدها به صورت زنجیره ای امکان پذیر است.

```
return response()->view('hello')->header('Content-Type', $type)
         ->withCookie(cookie('name', 'value'));
```

Redirect

با استفاده از تابع کمکی redirect و افزودن مسیر به آن می توانیم به مسیر مورد نظر هدایت شویم.

```
return redirect('user/login');
```

```
return redirect('user/login')->with('message', 'Login Failed');
```

همچنین می توانیم به همراه ریدایرکت کردن یک داده flash هم ارسال کنیم.

با استفاده از متد back می توانیم به مسیر قبلی که بودیم دوباره هدایت شویم.

```
return redirect()->back();
```

```
return redirect()->back()->withInput();
```

در مثال دومی می توانیم درخواست هایی که به این مسیر آمده را هم دوباره به مسیر قبلی ارسال کنیم که در پست قبلی نحوه کار با آنها را مشاهده کردیم.

می توانیم با استفاده از نام مسیر که در فایل routes.php تعریف میکنیم هم ریدایرکت را با استفاده از متد route انجام دهیم.

```
return redirect()->route('login');
```

```
// For a route with the following URI: profile/{id}
```

```
return redirect()->route('profile', [1]);
```

همچنین می توانیم با استفاده از یک آرایه به عنوان پارامتر دوم متد route داده هم به آن ارسال کنیم.

می توانیم با استفاده از متد action به یک اکشن در کلاس کنترلر دیگری هدایت شویم که بایستی نام کلاس با فضای نام آن نوشته شود و همچنین در صورت وجود پارامتر به صورت آرایه به عنوان پارامتر دوم به آن اضافه میکنیم.

```
return redirect()->action('App\Http\Controllers\HomeController@index ');
```

```
return redirect()->action('App\Http\Controllers\UserController@profi le', ['user' => 1]);
```

ایجاد پاسخ به صورتJSON

با استفاده از متد json که یک آرایه را به عنوان پارامتر ورودی دریافت میکنید و خروجی آن به صورت JSON می باشد.

```
return response()->json(['name' => 'Abigail', 'state' => 'CA']);
```

ایجاد پاسخ به صورت دانلود فایل

با استفاده از متد download که مسیر فایل را به عنوان پارامتر می گیرد و دو پارامتر اختیاری دیگر که نام فایل و هدر های فایل هست را دریافت میکند.

```
return response()->download($pathToFile);
```

```
return response()->download($pathToFile, $name, $headers);
```

```
return response()->download($pathToFile)->deleteFileAfterSend(true);
```

در مثال سوم فایل بعد از دانلود حذف خواهد شد.

برای اطلاعات بیشتر به اینجا مراجعه کنید.

کار با موتور قالب Blade و ایجاد Layout

توی این پست نحوه ایجاد فایل های view و نحوه render کردن اون توسط کنترلر و ارسال دیتا به اون رو کار کردیم. در لاراول برای ایجاد ویو ها میتونید از موتور قالب Blade هم استفاده کنید که کارتون رو در ایجاد layout ها و کدنویسی خیلی آسون میکنه. شما می تونید بخش هایی از وبسایت از جمله هدر و فوتر و منو ها و ... که در تمام صفحات وبسایت یکی هستن را داخل یک فایل layout ایجاد کرده و در فایل های دیگر قابل ارث بردن هست. این فایل ها با فرمت .blade.php ایجاد می شوند.

تعریف یک Layout ساده

در مسیر resources/views یک پوشه به نام layouts ایجاد کرده و فایل master.blade.php را داخل آن ایجاد کرده و کدهای زیر را داخل آن می نویسیم:

```
<!-- Stored in resources/views/layouts/master.blade.php -->


<html>
  <head>
    <title>App Name - @yield('title')</title>
  </head>
  <body>
    @section('sidebar')
      This is the master sidebar.
    @show


    <div class="container">
      @yield('content')
    </div>
  </body>
</html>
```

اکثر دستورات blade با علامت @ شروع می شوند. با استفاده از دستور yield می توانیم یک بخش را ایجاد کنیم که بعدا در فایل هایی که از آن ارث برده می شوند بتوانید محتوایی که در هر فایل متفاوت است را در ان قرار دهیم. نحوه استفاده از layout بالا را در فایلی دیگر مشاهده کنید:

```
@extends('layouts.master')


@section('title', 'Page Title')


@section('sidebar')
   @parent


   <p>This is appended to the master sidebar.</p>
@stop


@section('content')
   <p>This is my body content.</p>
@stop
```

همانطور که مشاهده کردید با استفاده از دستور extends می توانید فایل layout را به صفحه اضافه کنید. نحوه آدرس دهی هم به این صورت است که بین دایرکتوری و نام فایل ویو نقطه قرار می دهیم.

با استفاده از دستور section که نام yield مورد نظر را به آن می دهیم می توانیم محتوای جدید را داخل آن قرار دهیم. در پایان هم باید stop را بنویسیم .yield ها در فایل layout هیچ محتوایی ندارند اما اگر بخواهیم بخشی را تعریف کنیم که در فایل layout هم محتو ادشته باشند باید از section استفاده با این تفاوت که در layout باید در انتها show قرار دهیم. بخش ها در فایل به ارث برده شده override می شوند برای اینکه بتوانیم محتوای فایل والد رو هم داشته باشیم کافیه در ابتدا یا انتهای محتوای جدید دستور parent را اضافه کنیم. در مثال بالا بخش sidebar به این صورت است.

برای بخش yield می توانیم یک محتوای پیش فرض هم تعیین کنیم مثلا

```
@yield('section', 'Default Content')
```

چاپ داده یا متغیر ها درblade

با استفاده از بلاک های دو آکولاده می توانیم یک متغیر یا عبارت قابل چاپ را در صفحه چاپ کنیم.

```
Hello, {{ $name }}.
```

```
The current UNIX timestamp is {{ time() }}.
```

همچنین اگر متغیری با نام مورد نظر ست نشده بود یک مقدار پیش فرض برای چاپ در نظر بگیریم تا باعث ایجاد خطا در صفحه نشود.

```
{{ $name or 'Default' }}
```

دوآکولاد در blade تمامی دستورات html را escape میکند مانند دستور htmlentities در php عمل میکند.
اگر نخواهیم داده ها escape شوند به این صورت انجام دهید:

```
Hello, {!! $name !!}.
```

دستورات شرطی و حلقه ها هم به صورت های زیر قابل نوشتن هستند:

```
@if (count($records) === 1)
   I have one record!
@elseif (count($records) > 1)
   I have multiple records!
@else
   I don't have any records!
@endif
@foreach ($users as $user)
   <p>This is user {{ $user->id }}</p>
@endforeach
```

برای دیدن مثال های بیشتر به اینجا مراجعه کنید

اینکلود کردن فایل view در view دیگر

مثلا در یک فایل ویو فرم لاگین را طراحی کرده ایم و می خواهیم آن را در چند صفحه استفاده کنیم کافیست آن را مانند مثال زیر در فایل های مورد نظر اینکلود کنیم:

```
@include('view.name')
```

```
@include('view.name', ['some' => 'data'])
```

در مثال بالا view نام پوشه و name نام فایل ویو مورد نظر است. همچنین می توانیم دیتا هم به آن فایل ارسال کنیم.

توضیحات در Blade

برای نوشتن کامنت یا توضیحات می توانید به صورت زیر عمل کنید:

```
{{-- This comment will not be in the rendered HTML --}}
```

برای اطلاعات بیشتر به اینجا مراجعه کنید

توابع کمکی در لاراول

توابع کمکی یا helper بسایر زیادی در لاراول وجود دارند که در حین توسعه برنامه به کارتون میان و توی پست های قبلی هم از چندتا ازونا استفاده کردیم مثل تابع . view برای کار با آرایه ها و مسیرها و ایجاد url و کار با رشته ها توابع بسیار خوبی دارد.توی این پست میخواستم چندتا از پرکاربردهاشو معرفی کنم.

افزودن به آرایه با تابعarray_add

```
$array = ['foo' => 'bar'];
```

```
$array = array_add($array, 'key', 'value');
```

تقسیم آرایه به دو آرایه از کلیدها و مقادیر با تابعarray_devide

```
$array = ['foo' => 'bar'];
```

```
list($keys, $values) = array_divide($array);
```

گرفتن مسیر فیزیکی دایرکتوری app و public با توابع app_path وpublic_path

```
$path = app_path();
```

```
$path = public_path();
```

اجرای دستور htmlentities روی رشته با پشتیبانی از UTF-8 با تابعe

```
$entities = e('<html>foo</html>');
```

ایجاد یک رشته تصادفی به طول دلخواه با تابع str_random که مثلا مناسب برای ایجاد کلمه عبور است

```
$string = str_random(40);
```

ایجاد مسیر کامل با تابع url - پارامتر اولش مسیر نسبی هست و پارامتر دوم هم پارامترهای مسیر در صورت وجود است و پارامتر سوم اگر true باشد مسیر با پروتکل https ایجاد می شود

```
echo url('foo/bar', $parameters = [], $secure = null);
```

ایجاد یک توکن در فرم ها برای جلوگیری از حملات csrf با تابعcsrf_token

```
$token = csrf_token();
```

تابع dd هم یک متغیر یا آبجکت یا آرایه را می گیرد و به صورتی شبیه var_dump نمایش می دهد و برای debug کردن خیلی کاربردی هست

```
dd($value);
```

این توابع خیلی زیاد هستند که برای آشنایی با همه آنها می توانید به اینجا مراجعه کنید

کار با Session ها

در لاراول ۵ می توانیم از طریق کلاس Session و هم با استفاده از تابع کمکی session به مقادیر آنها دسترسی داشته باشیم.

ذخیره مقدار در یک سشن

در مثال زیر با هردو روش مقداری را در سشن ذخیره کرده ایم .key نام سشن و value مقدار آن است.
برای تعریف چند سشن کلید و مقدار را داخل یک آرایه قرار دهید.

```
Session::put('key', 'value');
```

```
session(['key' => 'value']);
```

باید توجه داشته باشید که برای ست کردن یک سشن هم در تابع کمکی session باید آن را در آرایه قرار دهید.

افزودن مقدار به یک سشن آرایه ای

```
Session::push('user.teams', 'developers');
```

بازیابی مقدار سشن با متد get امکانپذیر است.

```
$value = Session::get('key');
```

```
$value = session('key');
```

در صورتی که سشن مقداری نداشت می توانیم برای آن یک مقدار پیش فرض تعریف کنیم

```
$value = Session::get('key', 'default');
```

```
$value = Session::get('key', function() { return 'default'; });
```

گرفتن مقدار یک سشن و بلافاصله حذف آن با متد pull امکانپذیر است:

```
$value = Session::pull('key', 'default');
```

با متد all می توانیم به تمام مقادیر سشن ها را در یک آرایه بازیابی کنیم.

```
$data = Session::all();
```

برای حذف یک سشن خاص از متد forget که نام سشن را به آن می دهیم استفاده می کنیم. برای حذف تمامی سشن ها از flush استفاده میکنیم.

```
Session::forget('key');
```

```
Session::flush();
```

برای امنیت بیشتر سشن ها می توانید از متد regenerate برای تولید دوباره session id استفاده کنید:

```
Session::regenerate();
```

داده های فلش

سشن ها بعد از تولید تا وقتی که مرورگر بسته نشود از بین نمی روند. در لاراول سشن هایی به نام فلش وجود دارند که فقط برای یک درخواست معتبر هستند و بلافاصله در درخواست بعدی از بین میروند که مناسب برای ایجاد پیغام های خطا می باشند. مانند مثال زیر آنها را تولید می کنیم و به مانند سشن های دیگر بازیابی میکنیم.

```
Session::flash('key', 'value');
```

ذخیره سشن ها در دیتابیس

سشن ها به طور پیش فرض در فایل ذخیره می شوند. شما می توانید آنها در چند جای مختلف از جمله دیتابیس ذخیره کنید که هرکدام در کاربردهای خاصی استفاده می شوند.

در صورتی که میخواهید سشن ها را در دیتابیس ذخیره کنید کافی است این سه دستور را به ترتیب در ترمنال تایپ و اجرا کنید:

```
php artisan session:table
```

```
composer dump-autoload
```

```
php artisan migrate
```

سپس در فایل .env مقدار SESSION_DRIVER را به database تغییر دهید .

برای اطلاعات بیشتر به اینجا مراجعه کنید

اعتبار سنجی فرم ها

توی این پست یک مثال کاربردی از اعتبار سنجی فرم ها رو خواهیم داشت. برای این منظور ابتدا یک فرم رو در فایل view مثلا به نام form.blade.php در پوشه resources/views ایجاد می کنم و کدهای فرم را به این صورت می نویسم:

```
<ul>
    @foreach($errors->all('<li>:message</li>') as $error)
        {!! $error !!}
    @endforeach
</ul>
<form action="{{ url('test') }}" method="post">
    <input type="hidden" name="_token" value="{{ csrf_token() }}">
    <label for="name">Name</label>
    <input type="text" name="name" id="name" value="{{ old('name') }}">

    <label for="email">Email</label>
    <input type="text" name="email" id="email" value="{{ old('email') }}">

    <label for="age">Age</label>
    <input type="text" name="age" id="age" value="{{ old('age') }}">

    <input type="submit" value="Submit">
</form>
```

همینطور که مشاهده میکنید اکشن فرم را به مسیر test تعیین کردم. برای فرم هایتان باید حتما یک توکن تعیین کنید که یک فیلد مخفی با نام token_ است و مقدار آن توسط تابع csrf_token ایجاد می شود و برای جلوگیری از حملات csrf به کار می رود. برای هر تکست باکس هم مقدار آن را با تابع کمکی old مقداردهی کردم تا در صورت ریدایرکت بک شدن درخواست مقادیر قبلی فرم حفظ شوند.

خب حالا باید توی فایل routes.php دوتا مسیر تعریف کنیم. مسیر get که فایل فرم را رندر میکند و در مرورگر نمایش می دهد و post هم که مقادیر بعد از سابمیت به آن ارسال می شوند .

```
Route::get('test', function(){
   return view('form');
});

Route::post('test' , function(){

});
```

من برای طولانی نشدن مثال در همین فایل routes اعتبارسنجی رو انجام میدم اما شما بهتره برای رعایت اصول mvc این اعمال را داخل کنترلرها انجام بدین.

حالا اعتبارسنجی رو به این صورت انجام میدم:

```
Route::post('test' , function(){
   $validator = Validator::make(
      Request::all(),
      [
         'name' => 'required',
         'email' => 'required|email|unique:users',
         'age' => 'numeric',
      ]
   );

   if($validator->fails()){
      return redirect()->back()->withErrors($validator->errors())->withInput();
   }
});
```

همانطور که می بینید از کلاس Validator و متد make استفاده کردم. این متد دوتا پارامتر آرایه ای می گیرد که اولی آرایه ای از مقادیر هست که از فرم ارسال کرده ایم و دومی هم آرایه ای هست که قوانین اعتبارسنجی را برای هر فیلد تعریف می کنیم. چیزی که اینجا جدیده نحوه نوشتن قوانین اعتبار سنجی هست که یک آرایه هست که باید کلید آن نام اون فیلد فرم و مقدار اون قوانین اون فیلد باشد و هر قانون را هم با کاراکتر | از هم جدا میکنیم required .یعنی الزامی بودن فرم و email یعنی یک آدرس ایمیل معتبر باشد یا numeric یعنی مقدار باید عددی باشد و ... . در اینجا از یک قانون به نام unique برای فیلد email قرار دادم که در جدول users بررسی می کند که مقدار ایمیل واردشده در جدول قبلا ثبت نشده باشد. البته باید نام ستون ایمیل در جدول با نام فیلد یکی باشد وگرنه باید نام ستون را هم جلوی قانون اضافه کنیم .

این قوانین خیلی زیاد هستند که برای اطلاع از آنها و نحوه کارشون به اینجا مراجعه کنید

در نهایت با متد fails بررسی میکنیم اگر اعتبارسنجی دارای خطا بود به صفحه قبل ریدایرکت شود. پیغام های خطا و مقادیر قبلی فرم هم ارسال شوند.

حالا یک روش خیلی ساده تر از قبلی رو بهتون میگم که به جای استفاده از کلاس Validator داخل کنترلر از متد validate خود کنترلر استفاده کنید:

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

   //
}
```

تو این روش اعتبارسنجی انجام می شود و اگر خطای اعتبارسنجی نداشت که به ادامه کار می پردازد وگرنه خودش اتوماتیک به صفحه قبلی ریدایرکت میکند و پیغام های خطا را هم به آنجا ارسال می کند.

همه برنامه نویسان حرفه ای بدنبال این هستند که همیشه حداقل کد رو بنویسن پس اگر توی کلاس کنترلر مورنظر چندین بار از اعتبارسنجی در اکشن های مختلف می خواهید استفاده کنید باز روش بهتری هست که قوانین رو در یک کلاس request ایجاد کنید. ابتدا با دستور زیر در ترمینال یک کلاس request با نام دلخواه ایجاد کنید:

```
php artisan make:request StoreBlogPostRequest
```

توجه داشته باشید این کلاس حتما باید از کلاس Request ارث برده شود. حالا توی متد rules اون کلاس قوانین رو تعریف کنیم:

```
public function rules()
{
   return [
      'title' => 'required|unique|max:255',
      'body' => 'required',
   ];
}
```

کافیه تو هر اکشن کنترلری که می خواهیم اعتبارسنجی انجام شود از این کلاس استفاده کنیم .

```
public function store(StoreBlogPostRequest $request)
{
   // The incoming request is valid...
}
```

درخواست ها ابتدا اعتبارسنجی می شوند در صورتی که بدون خطا باشند وارد اکشن می شوند وگرنه به طور اتوماتیک به صفحه قبلی ریدایرکت و پیغام های خطا هم قابل دسترسی هستند.

نمایش پیغام های خطا در view

```
echo $errors->first('email');


foreach ($errors->all() as $error)
{
  //
}
```

در صورتی که فقط خطای فیلد خاصی را بخواهیم نمایش دهیم مانند مثال اول و اگر همه پیغام ها را نمایش دهیم به مانند مثال دوم عمل میکنیم.

همچنین می توانیم پیغام های خطا را در قالب یک تگ HTML نمایش دهیم که من در مثالم به این صورت عمل کردم:

```
<ul>
  @foreach($errors->all('<li class="error">:message</li>') as $error)
    {!! $error !!}
  @endforeach
</ul>
```

ایجاد یک قانون اعتبار سنجی دلخواه

اگر قانون مورد نظر شما در قوانین موجود لاراول وجود نداشت می توانید با استفاده از متد extend این قانون را ایجاد کنید:

```
Validator::extend('alpha_spaces', function($attribute, $value)
    {
      return preg_match('/^[\pL\s]+$/u', $value);
    });
```

مثلا قانونی که من نیاز داشتم مجاز بودن حروف الفبا و فاصله در یک مقدار بود که در بالا تعریف کردم.

ایجاد پیغام خطای دلخواه برای قوانین اعتبارسنجی

پیغام ها خطا به طور پیش فرض در مسیر resources/lang/en و فایل validation.php تعریف شده اند و به زبان انگلیسی هستند. ما می توانیم یک آرایه تعریف کنیم که کلید آن نام قانون و مقدار آن پیغام خطای مورد نظر شما می باشد و این آرایه را به عنوان پارامتر سوم به متد make بدهیم.

```
$messages = [
   'same'    => 'The :attribute and :other must match.',
   'size'    => 'The :attribute must be exactly :size.',
   'between' => 'The :attribute must be between :min - :max.',
   'in'      => 'The :attribute must be one of the following types: :values',
];
```

```
$validator = Validator::make($input, $rules, $messages);
```

البته راه بهتری پیشنهاد میکنم به جای اینکه در هر اکشن بخواهید این پیغام هارا ست کنید بهتر است داخل مسیر resources/lang یک پوشه به نام fa ایجاد کنیم و همه محتویات پوشه en را داخل آن کپی کنیم و سپس داخل فایل validation.php پیغام های خطای هر قانون را به فارسی و دلخواه خودتان ست کنید. همچنین داخل آرایه attributes داخل همان فایل هم نام فیلدهای فرم که به طور پیش فرض از خاصیت name هرتکست باکس گرفته می شود را به دلخواه خودتان تغییر دهید.

به مثال زیر توجه کنید:

```
"required"        => "فیلد :attribute الزامی است پر کردن آن.",
```

```
   'attributes' => [
      'name' => 'نام' ,
      'email' => 'آدرس ایمیل',
      'age' => 'سن',
   ],
```

برای قانون required یک پیغام دلخواه و نام دلخواهی برای فیلدها در نظر گرفتم.

برای استفاده از این پیغام های دلخواه چون من این پوشه را fa نامگزاری کردم باید داخل فایل app.php در پوشه config آیتم locale را به fa تغییر دهید.

برای اطلاعات بیشتر به اینجا مراجعه کنید

مباحث پایه کار با دیتابیس

یکی از مزیت های فریمورک لاراول کار با دیتابیس آن است که بسیار ساده است و متدهای زیادی برای عملیات های مختلف دارد. برای اعمال تنظیمات دیتابیس خود باید داخل فایل .env و همچنین در پوشه config و فایل database.php تنظیمات مورد نظر خود را اعمال کنید. به طور پیش فرض لاراول از mysql استفاده می کند اما از دیتابیس های MySQL , Postgres, SQLite و SQL Server هم پشتیبانی می کند و می توانیم از هر یک از انها استفاده کنیم.

اجرای کوئری با کلاسDB

در لاراول به سادگی می توانیم با استفاده از کلاس DB و نوشتن کوئری به صورت prepared statementsعمل مورد نظرمان را انجام دهیم.

با استفاده از متد select می توانیم رکوردهای داخل یک جدول را بازیابی کنیم و خروجی آن یک آرایه است. پارامتر دوم متد select هم یک آرایه از مقادیر است که که در صورتی که کوئری نیاز به bindکردن مقداری داشته باشد از آن استفاده میکنیم.نحوه استفاده از آن را به دوشکل مختلف می بینید:

```
$results = DB::select('select * from users where id = ?', [1]);
```

```
$results = DB::select('select * from users where id = :id', ['id' => 1]);
```

insert , update , delete

برای درج در جدول از متد insert و برای به روز رسانی از update و حذف از جدول delete را استفاده میکنیم:

```
DB::insert('insert into users (id, name) values (?, ?)', [1, 'Dayle']);
```

```
DB::update('update users set votes = 100 where name = ?', ['John']);
```

```
DB::delete('delete from users where id =:id', ['id' => 1]);
```

نکته : متدهای update و delete تعداد رکوردهایی که با این کوئری تغییر یافتند یا حذف شدند را برمیگرداند.

اگر کوئری غیر از ۴ عمل اصلی دیتابیس بود می توانیم از متد ststement استفاده کنیم:

```
DB::statement('drop table users');
```

برای تراکنش هم می توانید از متد transaction استفاده کنید و عملیات مورنظرتان را داخل تابع که به آن می دهیم را انجام دهیم. در صورتی که هریک از کوئری ها با خطایی مواجه شوند و اجرا نشوند به ظور اتوماتیک تمام کوئری های اجرا شده به عقب بر میگردند که مناسب برای عملیات های مالی می باشد.

```
DB::transaction(function()

{

   DB::table('users')->update(['votes' => 1]);


   DB::table('posts')->delete();

});
```

در صورتی که در برنامه تان از چند اتصال به دیتابیس استفاده می کنید با استفاده از متد connection و دادن نام اتصال به آن به عنوان پارامتر از آن استفاده کنیم:

```
$users = DB::connection('foo')->select(...);
```

برای اطلاعات بیشتر به اینجا مراجعه کنید

کار با دیتابیس با Query Builder

روش بهتر و آسانتر برای کار با دیتابیس در لاراول به جای نوشتن کامل کوئری استفاده از Query Builderاست. شما می توانید اکثر عملیات های دیتابیس را در برنامه تان انجام بدهید و این کوئری ها در همه دیتابیس هایی که لاراول ساپورت می کند کار کند. در ضمن کوئری بیلدر لاراول از bind کردن پارامترها استفاده می کند که برنامه تان را در برابر حملات SQL Injection محافظت میکند.

SELECT

برای انتخاب تمامی رکوردهای یک جدول ابتدا نام جدول موردنظر را به متد table و سپس با متد get رکوردها را واکشی میکنیم.

```
$users = DB::table('users')->get();


foreach ($users as $user)

{

  var_dump($user->name);

}
```

برای استفاده از شرط در کوئری از متد where استفاده می کنیم و این متد سه پارامتر میگیرد که اولی نام ستون موردنظر و دومی operator شرط (= , < , > , <= , ...) و سومین پارامتر هم مقدار موردنظر است. در صورتی که پارامتر دوم را ننویسیم به صورت پیش فرض عملگر = در نظر گرفته می شود. متد first هم اولین رکورد که با شرط فوق همخوانی داشته باشد را برمیگرداند که برای بازیابی یک رکورد استفاده می شود. در صورتی که چند رکورد را بخواهیم بازیابی کنیم از متد get استفاده میکنیم.

```
$user = DB::table('users')->where('name', 'John')->first();


var_dump($user->name);$users = DB::table('users')->where('votes', '>', 100)->get();
```

در صورتی که بخواهیم مقدار یک ستون خاص را که در یک شرط صدق میکند را بازیابی کنیم از متد pluck استفاده و نام ستون را به آن میدهیم. اگر بخواهیم لیست مقادیر یک ستون را واکشی کنیم از متد lists استفاده و مقدار ستون را به عنوان پارامتر به آن میدهیم که خروجی آن یک آرایه است و می توانیم نام ستون دیگری را هم به عنوان پارامتر دوم به آن بدهیم تا کلید آرایه مقادیر ان ستون باشند.

```
$name = DB::table('users')->where('name', 'John')->pluck('name');
```

```
$roles = DB::table('roles')->lists('title');
```

```
$roles = DB::table('roles')->lists('title', 'name');
```

استفاده از OR یا AND برای جدا کردن شرط ها

برای این کار کافی است بعد از متد where که نوشتیم متد orWhere را استفاده کنیم:

```
$users = DB::table('users')
        ->where('votes', '>', 100)
        ->orWhere('name', 'John')
        ->get();
```

عبارت بالا معادل کوئری زیر است :

```
SELECT * FROM users WHERE votes > 100 OR name = 'john'
```

اگر دوباره از متد where استفاده کنیم معادل AND در نظر گرفته می شود.

متدهای بسیار زیادی وجود دارند که به علت طولانی شدن مبحث و وجود مثال ها به ظور واضح در داکیومنت برای اطلاعات بیشتر به اینجا مراجعه کنید

استفاده از متدهای جادویی شرط

روش بهتر و با کدنویسی کمتر استفاده از متدهای جادویی هست. در مثال های زیر کوئری های معادل آنها را هم نوشته ام:

```
//SELECT * FROM users WHERE id=1 LIMIT 1;
$admin = DB::table('users')->whereId(1)->first();


//SELECT * FROM users WHERE id=2 AND email = 'john@doe.com' LIMIT 1;
$john = DB::table('users')
        ->whereIdAndEmail(2, 'john@doe.com')
        ->first(); //


//SELECT * FROM users WHERE name='Jane' OR age = 22 LIMIT 1;
$jane = DB::table('users')
        ->whereNameOrAge('Jane', 22)
        ->first();
```

استفاده از Order By و Group By و Having با کوئری بیلدر

```
$users = DB::table('users')
        ->orderBy('name', 'desc')
        ->groupBy('count')
        ->having('count', '>', 100)
        ->get();
```

همچنین می توانیم از LIMIT به همراه آفست در کوئری استفاده کنیم .

```
$users = DB::table('users')->skip(10)->take(5)->get();
```

در مثال بالا کوئری میگوید که از رکورد دهم در جدول users راانتخاب کن و تا ۵ رکورد را واکشی کن. (شماره گزاری رکوردها از صفر شروع میشود)

JOINکردن

با متد join می توانید دو یا چند جدول را باهم JOIN کنید. این متد ۴ پارامتر می گیرد که اولی جدولی که میخواهیم به آن پیوند بزنیم و پارامترهای بعدی فیلدهایی که باید باهم مساوی باشند را قرار میدهیم.

```
DB::table('users')
    ->join('contacts', 'users.id', '=', 'contacts.user_id')
    ->join('orders', 'users.id', '=', 'orders.user_id')
    ->select('users.id', 'contacts.phone', 'orders.price')
    ->get();
```

در مثال بالا به سه جدول orders, users, contacts پیوند زده شده است.

با کوئری بیلدر می توانیم با توابع جمعی (count, max, min, ...) تمام مقادیر اسکالر یک ستون را محاسبه کرده و مقداری اسکالر تولید می کند

```
$users = DB::table('users')->count();

$price = DB::table('orders')->max('price');

$price = DB::table('orders')->min('price');

$price = DB::table('orders')->avg('price');

$total = DB::table('users')->sum('votes');
```

درج کردن(INSERT)

با استفاده از متد insert می توانیم در جدول مورد نظر مقادیری را درج کنیم. مقادیر را در آرایه قرار می دهیم و به عنوان پارامتر به آن می دهیم. کلیدهای آرایه نام ستون جدول مورد نظر است.

```
DB::table('users')->insert(
  ['email' => 'john@example.com', 'votes' => 0]
);

$id = DB::table('users')->insertGetId(
  ['email' => 'john@example.com', 'votes' => 0]
);

DB::table('users')->insert([
  ['email' => 'taylor@example.com', 'votes' => 0],
  ['email' => 'dayle@example.com', 'votes' => 0]
]);
```

اگر در جدولتان فیلد id به صورت Auto-increment است می توانید از متد insertGetId استفاده کنید که بعد از درج کوئری id که تولید شده را به عنوان خروجی برمیگرداند. در مثال سوم در بالا هم همانظور که می بینید در صورتی که بخواهید چندین رکورد را باهم درج کنید کافیست رکوردها را به عنوان پارامتر به متد insert بدهیم و با ویرگول از هم جدا کنیم.

به روزرسانی (UPDATE)

با استفاده از متد update که یک آرایه به ان میدهیم که کلید های آن نام ستون موردنظر در جدول و مقادیر آن هم مقدار جدید می باشد رکوردها را آپدیت کنیم.

```
DB::table('users')
        ->where('id', 1)
        ->update(['votes' => 1]);
```

همچنین می توانیم با متد increment مقدار ستونی را یک واحد افزایش دهیم یا با ذکر یک پارامتر دوم تعداد افزایش را به طور مثال در مثال زیر ۵ واحد مشخص کنیم. متد decrement هم مقدار را کاهش می دهد و مانند متد قبلی عمل میکند.

```
DB::table('users')->increment('votes');


DB::table('users')->increment('votes', 5);


DB::table('users')->decrement('votes');


DB::table('users')->decrement('votes', 5);

DB::table('users')->increment('votes', 1, ['name' => 'John']);
```

در مثال آخر هم همانطور که می بینید هم می توانیم عمل افزایش را انجام دهیم و هم آپدیت سایر مقادیر ستون های جدول را که به عنوان پارامتر سوم و از نوع آرایه به آن می دهیم.

حذف کردن(Delete)

با استفاده از متد delete می توانیم رکوردی یا همه رکوردهای جدول را حذف کنیم. اگر از شرط استفاده نکنیم همه رکوردهای جدول حذف می شوند. با استفاده از متد truncate هم می توانیم همه مقادیر یک جدول را حذف کنیم با این تفاوت که truncate هیچ شرطی نمیگیره و سریعتر از delete هست یا تفاوت دیگر آن این است که id های اختصاص داده شده به رکوردها را هم reset میکند ولی در delete اینگونه نیست.

```
DB::table('users')->where('votes', '<', 100)->delete();

DB::table('users')->delete();

DB::table('users')->truncate();
```

با استفاده از متد union می توانیم دو کوئری را باهم اجتماع کنیم:

```
$first = DB::table('users')->whereNull('first_name');

$users = DB::table('users')->whereNull('last_name')->union($first)->get();
```

قفل کردن جدول هنگام اجرای عملیات

در صورتی که قصد دارید در هنگام انجام عملیات SELECT جدول قفل شود میتوانیم به صورت زیر عمل کنیم:

```
DB::table('users')->where('votes', '>', 100)->sharedLock()->get();

DB::table('users')->where('votes', '>', 100)->lockForUpdate()->get();
```

با استفاده از متد sharedLock جدول را به ظور کامل قفل می کنیم و با متد lockForUpdate جدول را هنگام عملیات SELECT فقط برای به روزرسانی قفل می کنیم.

برای دیدن مثال های بیشتر به اینجا مراجعه کنید

کار با دیتابیس و Eloquent

در لاراول می توانیم با استفاده از Eloquent که پیاده سازی شده از الگوی طراحی ActiveRecord است خیلی ساده تر با دیتابیس کار کنیم. در این روش هر جدول در دیتا بیس با یک کلاس Model در ارتباط است.

برای شروع کار با Eloquent باید ابتدا یک کلاس مدل از جدول ایجاد کنیم. کلاس های مدل را داخل پوشه app قرار میدهیم. با تایپ این دستور در ترمینال می توانیم یک مدل ایجاد کنیم:

```
php artisan make:model User
```

نام مدل را همیشه به صورت PascalCase بنویسید. به طور پیش فرض مدل با جدولی که مشابه نام مدل است اما فقط یک s به آخر اضافه شده متناظر است. مثلا مدل User با جدول users در دیتابیس مرتبط است. کلاس ایجاد شده باید محتوای آن به شکل زیر باشد:

```
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model {}
```

در صورتی که نام جدولی که با کلاس مدل از قانونی که در بالا گفتم تبعیت نمیکند میتوانید در کلاس با استفاده از پراپرتی table نام مورد نظر را ست کنیم.

```
class User extends Model {

    protected $table = 'my_users';

    public $timestamps = false;

}
```

همچنین در جداول باید دو ستون تاریخ و زمان به نام های created_at , updated_at وجود داشته باشند که هنگام ایجاد رکورد یا به روزرسانی آن مقداردهی می شوند اما شما می توانید با false قراردادن پراپرتی timestamps از ایجاد این ستون ها در جدول صرف نظر کنید.

حالا می توانیم به راحتی از کلاس مدل برای عملیات های دیتابیس استفاده کنیم. در Eloquent می توانیم از همه متدهای Query Builder استفاده کنیم.

برای بازیابی کلیه رکوردها از متد all استفاده میکنیم.

```
$users = User::all();
```

رکوردی را با داشتن id آن می توانیم با متد find بازیابی کنیم:

```
$user = User::find(1);

var_dump($user->name);
```

درصورتی که هنگام بازیابی رکوردها مقداری یافت شد و خواستیم یک خطای استثناء تولید شود کلمه OrFailرا به انتهای متد موردنظر اضافه میکنیم:

```
$model = User::findOrFail(1);

$model = User::where('votes', '>', 100)->firstOrFail();
```

از تمام متدهایی که در Query Builder یاد گرفتیم میتوانیم در Eloquent هم استفاده کنیم:

```
$users = User::where('votes', '>', 100)->take(10)->get();


foreach ($users as $user)

{

   var_dump($user->name);

}
```

درج کردن با Eloquent

ابتدا یک شی از کلاس مدل ایجاد میکنیم و سپس با استفاده از شی ایجاد شده attribute های مدل که همان نام ستونهای جدول هستند را با مقدار جدید مقداردهی میکنیم و سپس با صدازدن متد save رکورد جدید را ایجاد میکنیم.

```
$user = new User;


$user->name = 'John';


$user->save();


$insertedId = $user->id;
```

بعد از درج هم میتوانیم به id اختصاص داده شده به این رکورد دسترسی داشته باشیم.

همچنین می توانیم با استفاده از متد create یک رکورد جدید را به جدول اضافه کنیم که به این روش mass-assignmentگفته میشود که البته بایستی در کلاس مدل یک پراپرتی protected به نام guarded ایجاد کنیم که یک لیست سیاه می باشد و اجازه تغییر فیلدهای موردنظر را به کاربر نمیدهد. پراپرتی fillableبرعکس guarded است و یک لیست سفید برای عملیات mass-assignment ایجاد میکند.

```
// Create a new user in the database...

$user = User::create(['name' => 'John']);


// Retrieve the user by the attributes, or create it if it doesn't exist...

$user = User::firstOrCreate(['name' => 'John']);


// Retrieve the user by the attributes, or instantiate a new instance...

$user = User::firstOrNew(['name' => 'John']);
```

همچنین می توانیم از متدهای جادویی هم استفاده کنیم که مثلا در مثال دوم رکوردی را با این مقدار بازیابی کند و اگر وجود نداشت آن را ایجاد کند.

به روزرسانی رکوردها

برای آپدیت هم مشابه درج کردن عمل میکنیم فقط با این تفاوت که به جای ایجاد شی از کلاس مدل باید رکورد مورد نظر را ابتدا بازیابی کنید. مثلا در مثال زیر رکورد با id برابر ۱ را بازیابی کرده و سپس فیلد email را با مقدار جدیدی آپدیت میکند:

```
$user = User::find(1);


$user->email = 'john@foo.com';

$user->save();
```

حذف رکوردها

با استفاده ازمتد delete می توانید رکورد بازیابی شده را به راحتی حذف کنید.

```
$user = User::find(1);


$user->delete();
```

همچنین روش آسانتر استفاده از متد destroy است که id رکورد را به ان میدهیم. در صورتی که تعداد idها بیش از یکی بود هم می توانید در آرایه قرار دهید و به عنوان پارامتر به متد دهید و یا اینکه هرکدام را با ویرگول ازهم جدا کنید.

```
User::destroy(1);


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


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

مباحث پایه کار با Eloquent را در این پست گفتم. سایر مباحث را انشاالله در پست های بعدی ادامه خواهم داد

برای اطلاعات بیشتر به اینجا مراجعه کنید

ارتباطات (Relationships)

تقریبا همه جداول موجود در دیتابیس بایکدیگر ارتباط دارند. ارتباطات می تواند از انواع یک به یک و یک به چند و چند به چند باشد. در لاراول با Eloquent به راحتی می توانید این ارتباط ها را مدیریت و با آنها کار کنید. در این پست دو نمونه رایج ارتباط یک به چند (One-to-Many) و چند به چند (Many-to-Many) که اکثر ارتباطها در جداول دیتابیس به این صورت است را مثال خواهیم زد.

ارتباطOne To Many

برای مثال یک وبلاگ را در نظر بگیرید که دارای یک جدول به نام posts و یک جدول هم به نام commentsهست. هر پست در بلاگ می تواند دارای چند کامنت و هرکامنت هم فقط به یک پست تعلق دارد پس این ارتباط یک به چند است .

داخل کلاس مدل Post که سمت یک ارتباط است) ابتدا ارتباط به مدل Comment را با افزودن یک متد همنام با جدول متناظرش مثلا به نام comments به این صورت پیاده سازی میکنیم:

```
class Post extends Model {


  public function comments()
  {
    return $this->hasMany('App\Comment');
  }


}
```

با استفاده از متد hasMany کلاسی که با آن ارتباط چندی دارد را به عنوان پارامتر به آن میدهیم.

همچنین باید داخل کلاس مدل Comment هم متدی همنام کلاس متناظرش مثلا post ایجاد کرده و سپس با استفاده از متد belongsTo کلاس Post را به عنوان پارامتر به آن میدهیم.

```
class Comment extends Model {


   public function post()
   {
      return $this->belongsTo('App\Post');
   }


}
```

اکنون همانند مثال زیر می توانید تمام کامنت های پستی با id برابر ۱ را بازیابی کنید. همچنین می توانید از سایر متدها همچون شرط هم استفاده کنید.

```
$comments = Post::find(1)->comments;


$comments = Post::find(1)->comments()->where('title', '=', 'foo')->first();
```

نکته : نام کلید خارجی باید به صورتی باشد که ابتدا نام جدولی که از آن ارجاع می شود بدون s و سپس کلمه _id به انتهای آن افزوده شود مثلا برای مثال بالا کلید خارجی باید post_id باشد وگرنه باید در متد hasMany کلید خارجی را هم مشخص کنیم:

```
return $this->hasMany('App\Comment', 'foreign_key');
```

ارتباطMany To Many

برای پیاده سازی این نوع ارتباط فرض کنید یک جدول به نام users داریم و یک جدول هم به نام roles. هر کاربر می تواند چندین نقش داشته باشد و هر نقش هم میتواند به چندین کاربر تعلق داشته باشد. پس باید یک جدول واسط هم برای این دو جدول به نام role_user داشته باشیم. دقت کنید نام این جدول باید ترکیبی از نام دوجدول قبلی اما بدون s آخر آنها باشد که با _ از هم جدا شده اند. سپس کلید های خارجی user_id و role_id هم در این جدول ایجاد می شوند.

در مدل User یک متد همنام جدولی که با آن ارتباط دارد ایجاد میکنیم و سپس با استفاده از متد belongsToManyکلاس مدل Role را به آن میدهیم.

```
class User extends Model {


   public function roles()
   {
      return $this->belongsToMany('App\Role');
   }


}
```

در کلاس مدل Role هم مانند بالا عمل میکنیم:

```
class Role extends Model {


   public function users()
   {
      return $this->belongsToMany('App\User');
   }


}
```

حالا به راحتی می توانیم تمامی نقش های یک کاربر را بازیابی کنیم:

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

درج کردن در جدول رابطه دار

فرض کنید می خواهیم یک کامنت را در جدول commentsدرج کنیم. همانطور که قبلا مثال زدیم جدول posts با جدول comments دارای ارتباط یک به چند است و ستون post_id در جدول comment کلید خارجی است. همانند مثال زیر می توانید به روش mass-assignment رکوردی را در ج کنید به طوری که در فیلد post_id به طوراتوماتیک با توجه به پست مورد نظر id آن ثبت خواهد شد.

```
$comment = new Comment(['message' => 'A new comment.']);


$post = Post::find(1);


$comment = $post->comments()->save($comment);
```

نکته : در این روش درج باید حتما پراپرتی $guarded را هم در کلاس مدل مورد نظر که میخواهید عمل درج را انجام دهید ست کنید تا ستون هایی که قرار نیست توسط کاربر درج شود محافظت شوند. به طور مثال در زیر من اینگونه آن را تعریف کردم:

```
public $guarded = ['id' , 'post_id'];
```

همچنین می توانید تعداد زیادی کامنت را هم به روش بالا درج کنید. هررکورد را داخل یک آرایه قرار می دهیم و همچنین به جای متد save از saveMany استفاده میکنیم.

```
$comments = [
   new Comment(['message' => 'A new comment.']),
   new Comment(['message' => 'Another comment.']),
   new Comment(['message' => 'The latest comment.'])
];


$post = Post::find(1);


$post->comments()->saveMany($comments);
```

بعضی مواقع نیاز داریم که هنگام select کردن رکوردها خروجی را در قالب آرایه یا JSON داشته باشیم که Eloquent دارای متدهایی برای این کار می باشد.

با استفاده از متد toArray می توانیم خروجی هر کوئری را به یک آرایه تبدیل کنیم

```
$user = User::with('roles')->first();


return $user->toArray();
```

با متد toJson هم خروجی را به JSON تبدیل می کنیم:

```
return User::find(1)->toJson();
```

Eloquent دارای مباحث بسیار زیادی می باشد که گنجاندن همه آنها در این آموزش میسر نمی باشد و من مباحث اصلی را ذکر کردم

برای اطلاعات بیشتر به اینجا مراجعه کنید

صفحه بندی کردن(Pagination)

هنگامی که تعداد رکوردهایی که می خواهید در یک صفحه وب نمایش دهید زیاد می باشد بهترین روش برای مدیریت تعداد نمایش در هر صفحه صفحه بندی کردن است. در لاراول شما آسان تر از سایر فریمورک ها می توانید این کار را انجام دهید. کد HTML ای هم که برای نمایش صفحه بندی تولید می شود سازگار با Bootstrap Twitter می باشد.

هنگام بازیابی رکوردها از دیتابیس کافی است از متد paginate استفاده کنیم و تعداد آیتم های قابل نمایش در هر صفحه را هم به عنوان پارامتر به آن بدهیم:

```
$users = DB::table('users')->paginate(15);

$allUsers = User::paginate(15);


$someUsers = User::where('votes', '>', 100)->paginate(15);
```

در مثال های فوق هم با روش کوئری بیلدر و هم Eloquent اینکار را انجام داده ایم و تعداد آیتم ها را ۱۵ تعیین کردیم.

حالا فرض کنید تمام کاربران را از دیتابیس واکشی کردیم و به صفحه view با متغیری به نام users ارسال کردیم. داخل ویو موردنظر کدهای زیر را قرار می دهیم:

```
<div class="container">
  <?php foreach ($users as $user): ?>
    <?php echo $user->name; ?>
  <?php endforeach; ?>
</div>


<?php echo $users->render(); ?>
```

با استفاده از حلقه foreach نام کاربران را نمایش میدهیم. برای نمایش کد HTML مربوط به صفحه بندی هم از متد render استفاده میکنیم و آن را چاپ میکنیم. البته مثال بالا به روش php نوشته شده و شما بهتر است از موتور قالب Balde استفاده کنید. با CSS میتوانید قالب نمایش صفحه بندی را به دلخواه خودتان تغییر دهید.

غیر از متد render متدهای دیگر هم وجود دارند که می توانید اطلاعات بیشتری را بدست آورید به طور مثال currentPage شماره صفحه جاری را نمایش می دهد و lastPage شماره آخرین صفحه و ...

اگر فقط می خواهید لینک Next و Previous نمایش داده شود و صفحه بندی ساده ای باشد از متد simplePaginateهنگام واکشی استفاده کنید .

```
$someUsers = User::where('votes', '>', 100)->simplePaginate(15);
```

به طور پیش فرض URL هنگام صفحه بندی مثلا به صورت ۲ ?page=خواهد بود شما می توانید با متد setPathیک URL دلخواه هنگام نمایش صفحه بندی ایجاد کنید:

```
$users = User::paginate();

$users->setPath('custom/url');
```

در مثال بالا آدرس URL به صورت http://example.com/custom/url?page=2 نمایش داده می شود.

به انتهای URL می توانیم کوئری استرینگ هم اضافه کنیم. هنگام نمایش صفحه بندی با استفاده از متد appendکه داده های کوئری استرینگ را به عنوان آرایه به آن می دهیم:

```
<?php echo $users->appends(['sort' => 'votes'])->render(); ?>
```

در مثال بالا آدرس URL به صورت http://example.com/something?page=2&sort=votes نمایش داده میشود.

همچنین می توانیم با متد fragment یک آدرس fragment را به انتهای URL اضافه کنیم.

```
<?php echo $users->fragment('foo')->render(); ?>
```

در مثال بالا URL به صورت http://example.com/something?page=2#foo نمایش داده می شود.

در هنگام ایجاد migration می توانید نام جدول را هم در دستور مشخص کنید.

```
php artisan make:migration create_users_table --create=users
```

برای نامگزاری فایل migration معمولا از یک نام با مسما که نشانده عملیات موردنظرمان است استفاده میکنیم مثلا برای افزودن یک ستون جدید به نام votes در جدول users به این صورت فایل را نامگزاری و ایجاد میکنیم:

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

بعضی از عملیات های دیتابیس ممکن است مخرب باشند و هنگامی در جداول دیتابیس داده ای وجود داشته باشد باعث از بین رفتن برخی داده ها شود برای محافظت از این خطرات از دستور migaret به صورت زیر استفاده کنید و در پایان آن --force را قرار دهید:

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

همانطور که در پست قبل دیدید از دستور rollback استفاده کردیم که این دستور روی آخرین عملیات migrationعمل میکند. برای اینکه همه عملیات ها را rollback کنیم از دستور migrate:reset استفاده میکنیم و در صورتی که بخواهیم همه عملیات ها rollback و سپس دوباره اجرا شوند از migrate:refreshاستفاده میکنیم.

```
php artisan migrate:resetphp artisan migrate:refresh
```

فرض کنید میخواهیم ستون ایمیل را به جدول users اضافه کنیم ابتدا فایل migration ای ایجاد و سپس در متد up آن دستور زیر را می نویسیم:

```
Schema::table('users', function($table)

{

   $table->string('email');


//$table->string('name')->after('email');});
```

در متد down هم دستورات زیر را قرار میدهیم:

```
Schema::table('users', function($table)

{

   $table->dropColumn('email');


});
```

حالا با اجرای دستور php artisan migrate ستون موردنظر ایجاد خواهد شد. همچنین می توانیم با استفاده از after آن را بعد از ستون خاصی در دیتابیس قرار دهیم وگرنه به انتهای جدول افزوده می شود.

همانطور که در پست قبل هم گفتم متدهای بسار زیادی برای تعریف ستون ها وجود دارد که می توانید از اینجا با هرکدام آشنا شوید.

برای تغییر نام جدول از متد rename از کلاس Schema مانند مثال اول در زیر استفاده کنید. با استفاده drop هم می توانیم جدولی را حذف کنیم و dropIfExists هم ابتدا بررسی میکند اگر جدول وجود داشت آن را حذف میکند.

```
Schema::rename($from, $to);

Schema::drop('users');


Schema::dropIfExists('users');
```

برای ویرایش یک ستون ابتدا بایستی مطمئن شوید که وابستگی doctrine/dbal روی فریمورک نصب شده باشد در غیر اینصورت مانند زیر عمل کنید:

در بخش require فایل composer.json آن را اضافه کرده:

```
"require": {
   "laravel/framework": "5.0.*",
   "illuminate/html": "~5.0",
   "doctrine/dbal": "2.5.*",
},
```

سپس از دستور composer update در ترمینال برای نصب این وابستگی استفاده کنید.

همانند مثال زیر می توانیم یک ستون را ویرایش کنیم. در مثال زیر طول ستون name را به ۵۰ کاراکتر تغییر داده و آن را قابل NULL بودن تعریف میکنیم:

```
Schema::table('users', function($table)
{
   $table->string('name', 50)->nullable()->change();
});
```

برای افزودن کلید خارجی به یک جدول هم به این صورت عمل میکنیم:

```
$table->integer('user_id')->unsigned();
$table->foreign('user_id')->references('id')->on('users');
```

ابتدا یک ستون unsigned عددی به نام user_id ایجاد کردیم و سپس در پایین آن را کلید خارجی تعریف کردیم و گفتیم که مرجعی از ستون id از جدول users می باشد.

همچنین می توانیم خاصیت onDelete آن راهم تعیین کنیم:

```
$table->foreign('user_id')
    ->references('id')->on('users')
    ->onDelete('cascade');
```

برای حذف کلید خارجی هم همانند مثال زیر عمل میکنیم:

```
$table->dropForeign('posts_user_id_foreign');
```

با استفاده از متد hasTable می توانیم بررسی کنیم آیا جدول موردنظر وجود دارد یا خیر و یا با استفاده از متد hasColumn بررسی کنیم در جدول موردنظر ستون های موردنظرمان وجود دارد یا خیر:

```
if (Schema::hasTable('users'))
{
  //
}

if (Schema::hasColumn('users', 'email'))
{
  //
}
```

برای افزودن و حذف ایندکس می توانیم همانند مثال های زیر عمل کنیم:

```
$table->string('email')->unique();
$table->dropUnique('email');
```

برای حذف ستونهای زمانی هم مانند مثال های زیر عمل کنید:

```
$table->dropTimestamps();
$table->dropSoftDeletes();
```

همچنین می توانیم موتور ذخیره سازی دیتابیس را هم مشخص کنیم:

```
$table->engine = 'InnoDB;'
```

Hashکردن

در لاراول با استفاده از کلاس Hash می توانیم یک رشته را به صورت هش در بیاوریم که مناسب برای هش کردن کلمه عبور کاربران برای ذخیره در دیتابیس می باشد.

از متد make برای هش کردن کلمه عبور استفاده میکنیم و کلمه عبور هش شده را در دیتابیس ذخیره کنیم:

```
$password = Hash::make('secret');
```

همچنین می توانیم از تابع کمکی bcrypt نیز استفاده کنیم:

```
$password = bcrypt('secret');
```

همچنین برای بررسی صحت کلمه عبور وارد شده توسط کاربر با کلمه عبور هش شده ذخیره در دیتابیس به این صورت عمل میکنیم:

```
if (Hash::check('secret', $hashedPassword))
{
    // The passwords match...
}
```

احراز هویت کاربران(Authentication)

پیاده سازی احراز هویت در لاراول بسیار ساده ساده است. تنظیمات مربوط به احراز هویت در پوشه configو فایل auth.php قرار دارد. در این فایل می توانید درایور را eloquent تعیین کنید و کلاس مدلی که به جدول کاربران دسترسی دارد را مشخص کنید و همچنین در بخش table نام جدولی که اطلاعات کاربران در ان ذخیره می شود را مشخص کنید.

به طور پیش فرض در پوشه app یک مدل به نام User وجود دارد که با استفاده از eloquent لاراول یک سیستم احراز هویت را پیاده سازی کرده است که شما می توانید از آن استفاده کنید. در مسیر app/Http/Controllers/Authدوکنترلر برای استفاده در سیستم احراز هویت استفاده می شوند که AuthControllerبرای ایجاد کاربر جدید یا لاگین کردن و PasswordController برای ریست کردن کلمه عبور کاربرانی که آن را فراموش کرده اند به کار می رود. تمام view های مربوطه هم در پوشه resources/views/authقرار دارند و شما می توانید آنها را به دلخواه خودتان ویرایش کنید.

همچنین اگر نیاز دارید تغییراتی در فرم ثبت نام کاربر جدید بدهید کافی است در مسیر App\Services در فایل Registrar.php تغییرات موردنظر را اعمال کنید. در متد validator می توانید قوانین اعتبارسنجی فیلدها و در متد create نیز مقادیر فیلدهارا در دیتابیس و جدول users درج کنید. شما به راحتی می توانید از این سیستم احراز هویت پیش فرض لاراول استفاده کنید.

خود لاراول یک middleware به نام Authenticate ایجاد کرده که در متد handle آن ابتدا بررسی میکند آیا کاربر لاگین کرده یا خیر و در غیر اینصورت آن را به صفحه login هدایت میکند. شما با استفاده از این middleware در سازنده کلاس کنترلری که میخواهید فقط کاربران احراز هویت شده دسترسی داشته باشند به صورت زیر عمل کنیم. به طور مثال در کلاس کنترلر HomeController به همین صورت عمل شده است:

```
public function __construct()
{
   $this->middleware('auth');
}
```

در صورتی که نمیخواهید از این سیستم احراز هویت تهیه شده توسط لاراول استفاده کنید نگران نباشید. خودتان هم می توانید به سادگی آن را پیاده سازی کنید.

برای اینکار باید کلاس Auth را به کنترلر ایمپورت کنید و سپس با استفاده از متد attempt به عنوان پارامتر یک آرایه دریافت می کند و کلیدهای این آرایه نام ستون های موردنظر در دیتابیس و جدول users و مقادیر آن هم مقداروارد شده توسط کاربر است صحت اطلاعات کاربر را بررسی کنید. متد attempt در صورتی که احراز هویت با موفقیت انجام شود true وگرنه false برمیگرداند.

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


use Auth;

use Illuminate\Routing\Controller;


class AuthController extends Controller {


   /**
    * Handle an authentication attempt.
    *
    * @return Response
    */
   public function authenticate()
   {
      if (Auth::attempt(['email' => $email, 'password' => $password]))
      {
         return redirect()->intended('dashboard');
      }
   }


}
```

همچنین می توانیم در متد attempt اطلاعات بیشتری را بررسی کنیم.مثلا در مثال زیر علاوه بر ایمیل و کلمه عبور باید کاربر فیلد تایید آن در دیتابیس هم ۱ باشد:

```
if (Auth::attempt(['email' => $email, 'password' => $password, 'active' => 1]))

{

   // The user is active, not suspended, and exists.

}
```

در هر قسمت از برنامه هم که نیاز دارید بررسی کنید کاربر جاری احراز هویت شده است یا خیر کافی است از متد check اینکار را انجام دهید:

```
if (Auth::check())

{

   // The user is logged in...

}
```

برای logout کردن از برنامه هم از متد logout استفاده میکنیم:

```
Auth::logout();
```

بعداز اینکه کاربر احرازهویت شد به راحتی با استفاده از متد user میتوانید یک آبجکت از کاربر جاری ایجاد کنید:

```
$user = Auth::user();

 echo $user->name;
```

در مثال همانطور که مشاهده کردید به راحتی توانستم به نام کاربر دسترسی داشته و آن را چاپ کنم.

در کنترلر به این صورت هم می توان به روش های زیر یک آبجکت از کاربر ایجاد کنیم:

```
public function updateProfile(Request $request)
  {
    if ($request->user())
    {
      // $request->user() returns an instance of the authenticated user...
    }
  }
//xxxxxxxxxxxxxxxxxxxxxxxx  xxxxxxxxxxxxxxxxxxxxx
public function updateProfile(Authenticatable $user)
  {
    // $user is an instance of the authenticated user...
  }
```

برای مسیرها هم می توانیم middleware احرازهویت را تعریف کنیم تا دسترسی به مسیر فقط برای کاربران احرازهویت شده میسر باشد.

```
Route::get('profile', ['middleware' => 'auth', 'uses' => 'ProfileController@show';(['
```

مباحث اصلی را ذکر کردم و برای اطلاعات بیشتر می توانید به اینجا مراجعه کنید

مثال آپلود فایل

حالا نوبت این است که با یک مثال کاربردی نحوه آپلود فایل در لاراول رو کار کنیم. فرض کنید می خواهیم در جدول posts یک مطلب جدید را اضافه کنیم که این مطلب دارای یک تصویر هم می باشد که قرار است آن را در مسیر public/uploads ذخیره کنیم. فرض میکنیم در جدول posts ستونهای (id, title, body, pic_name)وجود دارد.

یک فایل view به نام form.blade.php در مسیر resources/views ایجاد کنید و کدهای زیر را داخل آن قرار دهید:

کد:HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>form validation</title>
  <style type="text/css">
    .error {
      color: red;
      font-weight: bold;
    }
    .success {
      color: green;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <form action="{{ url('add-post') }}" method="post" enctype="multipart/form-data">
    <input type="hidden" name="_token" value="{{ csrf_token() }}">
    <label for="title">Title</label>
    <input type="text" name="title" id="title" value="{{ old('title') }}">
    <span class="error">{{ $errors->first('title') }}</span><br>
```

```
        <label for="post">Post</label>
        <textarea name="post" id="post">{{ old('post') }}</textarea>
        <span class="error">{{ $errors->first('post') }}</span><br>

        <label for="photo">Select an Image:</label>
        <input type="file" name="photo" id="photo">
        <span class="error">{{ $errors->first('photo') }}</span><br>

        <input type="submit" value="Submit">
    </form>
    <p class="success">{{ session('message') }}</p>
    <p class="error">{{ session('error') }}</p>
</body>
</html>
```

اکنون مسیر های زیر را در فایل routes.php تعریف میکنیم:

```
Route::get('add-post', 'PostController@getAddPost');

Route::post('add-post', 'PostController@postAddPost');
```

همانطور که می بینید باید یک کنترلر به نام PostController داشته باشیم و متدهای getAddPost و postAddPostرا داخل آن تعریف کنیم.

ابتدا برای رندر کردن فایل ویو متد getAddPost را به صورت زیر بنویسید:

```
public function getAddPost()
{
    return view('form');
}
```

کد های زیر را هم در متد postAddPost قرار دهید:

```
public function postAddPost(Request $request)
{
   $rules = [
      'title' => 'required|max:255|unique:posts',
      'post' => 'required',
      'photo' => 'required|image|max:1024',
   ];
   $v = Validator::make($request->all(), $rules);
   if($v->fails()){

      return redirect()->back()->withErrors($v->errors())->withInput($request->except('photo'));

   } else {
      $file = $request->file('photo');
      if($file->isValid()){
         $fileName = time().'_'.$file->getClientOriginalName();
         $destinationPath = public_path().'/uploads';
         $file->move($destinationPath, $fileName);
         $post = new Post;
         $post->title = $request->input('title');
         $post->body = $request->input('post');
         $post->pic_name = $fileName;
         $post->save();

         return redirect()->back()->with('message', 'The post successfully inserted.');
      } else {
         return redirect()->back()->with('error', 'uploaded file is not valid.');
       }
}
```

همانطور که که می بینید ابتدا مقادیر فرم را اعتبارسنجی کردیم. برای فایل هم با قانون max مشخص کردم که فایل فقط می تواند ۱۰۲۴ کیلوبایت سایز داشته باشد و همچنین با قانون image مشخص میکنیم که فایل از نوع تصویر باشد فقط mime type های (jpeg, png, bmp, gif, or svg) را قبول میکند. در صورتی که می خواهید محدودیت بیشتری برای mime type فایل در نظر بگیرید یا اصلا فایل شما تصویر نیست می توانید با استفاده از قانون mime نوع فایل را مشخص کنید. در صورتی که اعتبارسنجی دارای خطا باشد به فرم برگشته و خطاها نمایش داده می شوند .

سپس اطلاعات فایل رو در متغیر $file قرار دادم و با استفاده از متدهای کلاس UploadedFile می توانیم به اطلاعات فایل دسترسی داشته باشیم. نام فایل را تلفیقی از timestamp جاری و نام اصلی فایل تعیین کردم تا احتمال اینکه نام فایل تکراری باشد وجود نداشته باشد و داخل متغیر $fileName قرار دادم. مسیر آپلود فایل را در $destinationPath قرار دادم و با استفاده از متد move فایل را آپلود میکنیم. این متد مسیر آپلود و نام فایل را به عنوان پارامتر میگیرد.

در نهایت سایر مقادیر فرم به همراه نام فایل را در جدول posts درج میکنیم. در صورت موفقیت یا عدم موفقیت نیز پیغام های خطایی را ست و در ویو چاپ میکنیم.

اکنون هرقسمت از وبسایت که می خواهیم پست ها را نمایش دهیم به راحتی می توانیم تصویر را هم با استفاده از نام آن نمایش دهیم:

```
<img src="{{ asset('uploads/'.$post->pic_name) }}" alt="{{ $post->pic_name }}" >
```

مثال ارسال ایمیل

در لاراول ۵ شما به راحتی می توانید با استفاده از کلاس Facade Mail یک ایمیل را ارسال کنید. توی این بخش هم میخوام به صورت کاربردی نحوه ارسال ایمیل را برایتان توضیح بدهم. فرض میکنیم یک فرم تماس با ما داریم که میخواهیم بعد از تکمیل ان توسط کاربر به ایمیل مدیر سایت ارسال شود .

ابتدا باید در فایل .env تنظیمات مربوط به ایمیل هاست خود را ست کنید. در این مثال من تنظیمات جیمیل خودم را قرار دادم:

```
MAIL_DRIVER=smtp

MAIL_HOST=smtp.gmail.com

MAIL_PORT=587

MAIL_USERNAME=**********@gmail.com

MAIL_PASSWORD=**********
```

همچنین در پوشه config و فایل mail.php هم می توانید تنظیمات بیشتری را اعمال کنید.

حالا دوتا مسیر توی فایل routes.php ایجاد میکنیم:

```
Route::get('contact-me', ['as' => 'contact', 'uses' => 'ContactController@contactForm']);


Route::post('contact-me', ['as' => 'contact_send', 'uses' => 'ContactController@contactSend']);
```

همانطور که مشاهده میکنید برای هر مسیر یک نام انتخاب کردم و همچنین به کنترلر ContactControllerو اکشن contactForm برای درخواست های GET و اکشن contactSend برای درخواست های POST نیاز داریم. متد contactForm را به این صورت می نویسیم:

```
public function contactForm()
  {
     return view('emails.contact');
  }
```

همانطور که مشخص کردیم باید فرم تماس با ما را در پوشه emails و فایل contact.blade.php در مسیر resources/views ذخیره میکنیم و کدهای زیر را داخل آن قرار می دهیم:

کد:HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Contact</title>
</head>
<body>
<h1>Contact Me</h1>
<form action="{{ route('contact_send') }}" method="POST">
   <input type="hidden" value="{{ csrf_token() }}" name="_token">

   <label for="name">Your Name: </label>
   <input type="text" name="name" id="name" value="{{ old('name') }}"> <span class="error">{{
$errors->first('name') }}</span> <br>

   <label for="email">Your Email: </label>
   <input type="email" name="email" id="email" value="{{ old('email') }}"> <span class="error">{{
$errors->first('email') }}</span> <br>
```

```
    <label for="message">Message:</label>
    <textarea name="message" id="message">{{ old('message') }}</textarea> <span class="error">{{ $errors->first('message') }}</span> <br>

    <input type="submit" value="Send">
</form>

@if (Session::has('message'))
  {{ Session::get('message') }}
@endif
</body>
</html>
```

توی این مثال از ویژگی کلاس Request هم برای اعتبارسنجی استفاده میکنیم. پس با دستور زیر یک کلاس Request ایجاد میکنیم:

```
php artisan make:request ContactFormRequest
```

این کلاس در مسیر app/Http/Requests ایجاد می شود. آن را باز کرده و در متد rules آن قوانین اعتبارسنجی فرمتان را تعیین کنید.

```
public function rules()
{
  return [
    'name' => 'required',
    'email' => 'required|email',
    'message' => 'required',
  ];
}
```

همچنین متد authorize را که به طور پیش فرض false برمیگرداند true کنید چون نیازی به اهراز هویت در این درخواست نداریم. خب با این کار دیگه نیازی نیست تو کنترلر اعتبارسنجی انجام بدیم فقط کافیه این کلاسی که ساختیم رو به عنوان پارامتر به متد contactSend بدهیم:

```
public function contactSend(ContactFormRequest $request)
  {
     extract($request->all());


     Mail::send('emails.email',
        array(
           'name' => $name,
           'email' => $email,
           'content' => $message
        ), function($message) use($email, $name) {


           $message->from($email, $name);
           $message->to('example@gmail.com')->subject('Test Email');
     });


         return Redirect::route('contact')->with('message', 'Thanks for contacting us!');
  }
```

به این متد فقط درخواست های اعتبارسنجی شده وارد می شوند و اعتبارسنجی داخل کلاس ContactFormRequestانجام می شود. ابتدا همه داده های فرم که به صورت آرایه هست را با دستور extractتبدیل به متغیر کردم و با استفاده از کلاس Mail و متد send ایمیل را ارسال میکنیم. متد send سه تا پارامتر میگیره که اولی یک فایل وبو هست که داخل آن محتویات html برای ارسال فرم را تولید میکنیم و پارامتر دوم داده هایی که نیاز داریم به آن فابل ویو ارسال کنیم را در قالب آرایه میفرستیم و در پارامتر سوم هم یک تابع بی نام ایجاد کرده و اطلاعات فرستنده و گیرنده نامه را تعیین میکنیم. در متد from نام و ایمیل فرستنده و در متد to ایمیل گیرنده نامه و در متد subject موضوع نامه را تعیین میکنیم .

در نهایت به صفحه تماس با ما ریدایرکت میکنیم و پیغامی را هم ارسال و چاپ میکنیم .

همچنین باید یک فایل ویو که در متد send آن را به عنوان پارامتر اول دادیم هم در پوشه emails ایجاد کنیم. پس نام آن را email.blade.php قرار می دهیم و محتویات زیر را داخل آن می نویسیم:

کد HTML:

```
You received a message from hamo.ir:


<p>

Name: {{ $name }}

</p>


<p>

Email address :{{ $email }}

</p>


<p>

  {{ $content }}

</p>
```

RESETکردن کلمه عبورکاربر

توی این پست به درخواست یکی از دوستان نحوه ریست کردن کلمه عبور کاربران رو باهم کار میکنیم. ابتدا در مسیر app/Http/Controllers/Auth کلاس PasswordController را باز کنید و به آن متد getEmailرا اضافه کنید:

```
public function getEmail()
  {
     return view('auth.password');
  }
```

پس بایستی یک فایل view در پوشه auth به نام password.blade.php داشته باشیم که فرم ریست کردن کلمه عبور در آن قرار دارد. محتویات این فایل شبیه زیر است:

کد:HTML

```
@extends('app')


@section('content')
<div class="container-fluid">
  <div class="row">
    <div class="col-md-8 col-md-offset-2">
      <div class="panel panel-default">
        <div class="panel-heading">Reset Password</div>
        <div class="panel-body">
          @if (session('status'))
            <div class="alert alert-success">
              {{ session('status') }}
            </div>
          @endif


          @if (count($errors) > 0)
            <div class="alert alert-danger">
              <strong>Whoops!</strong> There were some problems with your input.<br><br>
              <ul>
```

```
                    @foreach ($errors->all() as $error)
                        <li>{{ $error }}</li>
                    @endforeach
                </ul>
            </div>
        @endif

        <form class="form-horizontal" role="form" method="POST" action="{{
url('/password/email') }}">
            <input type="hidden" name="_token" value="{{ csrf_token() }}">

            <div class="form-group">
                <label class="col-md-4 control-label">E-Mail Address</label>
                <div class="col-md-6">
                    <input type="email" class="form-control" name="email" value="{{ old('email') }}">
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-6 col-md-offset-4">
                    <button type="submit" class="btn btn-primary">
                        Send Password Reset Link
                    </button>
                </div>
            </div>
        </form>
    </div>
  </div>
 </div>
</div>
@endsection
```

این فایل از layout ای که لاراول به طور پیش فرض در پوشه views قرار داده به نام app.blade.php ارث برده می شود که از bootstrap هم استفاده میکند.

در کلاس PasswordController یک متد به نام postEmail هم برای دریافت ایمیل کاربر بعد از ارسال توسط این فرم باید ایجاد کنیم:

```
public function postEmail(Request $request)
 {
    $v = Validator::make($request->all(), [
       'email' => 'required|email|exists:users',
    ]);
    if ($v->fails())
    {
       return redirect()->back()->withErrors($v->errors());
    } else {
       $response = $this->passwords->sendResetLink($request->only('email'), function($m)
       {
          $m->subject($this->getEmailSubject());
       });
       switch ($response)
       {
          case PasswordBroker::RESET_LINK_SENT:
             return redirect()->back()->with('status', trans($response));

          case PasswordBroker::INVALID_USER:
             return redirect()->back()->withErrors(['email' => trans($response)]);
       }
    }
 }
```

نکته : ابتدای کلاس کنترلر این کلاس ها را ایمپورت کنید چون در بدنه کلاس از آنها استفاده میکنیم:

```
use Illuminate\Http\Request;

use Validator;
```

همانطور که مشاهده کردید ابتدا اعتبارسنجی رو انجام دادیم. در اعتبارسنجی هم بررسی کردیم که آیا آدرس ایمیل وارد شده در جدول users وجود دارد یا خیر بعد از ان اقدام به ارسال ایمیل به کاربر میکنیم و یک پاسخی دریافت میکنیم که این پاسخ را در حلقه switch قرار میدیم به این صورت که اگر linkریست کردن به درستی ارسال شده بود یا ایمیل کاربر نامعتبر بود به صفحه قبلی ریدایرکت شود و پیغام خطای مناسبی را در صفحه ویو چاپ کند.

حالا وارد ایمیل خودتان بشوید و بر روی لینکی که برایتان ارسال شده است کلیک کنید. این لینک حاوی یک توکن است که آن توکن دوباره در کنترلر بررسی میشود و اگر درست بود وارد صفحه ای می شوید که می توانید کلمه عبور خودتان را ریست کنید. فایل view آن در همان پوشه views/auth و به نام reset.blade.php است و حاوی کدهای زیر است:

کد:HTML

```
@extends('app')


@section('content')

<div class="container-fluid">

  <div class="row">

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

      <div class="panel panel-default">

        <div class="panel-heading">Reset Password</div>

        <div class="panel-body">

          @if (count($errors) > 0)

            <div class="alert alert-danger">

              <strong>Whoops!</strong> There were some problems with your input.<br><br>

              <ul>

                @foreach ($errors->all() as $error)

                  <li>{{ $error }}</li>
```

```
                @endforeach
                </ul>
            </div>
        @endif

        <form class="form-horizontal" role="form" method="POST" action="{{
url('/password/reset') }}">
            <input type="hidden" name="_token" value="{{ csrf_token() }}">
            <input type="hidden" name="token" value="{{ $token }}">

            <div class="form-group">
                <label class="col-md-4 control-label">E-Mail Address</label>
                <div class="col-md-6">
                    <input type="email" class="form-control" name="email" value="{{ old('email') }}">
                </div>
            </div>

            <div class="form-group">
                <label class="col-md-4 control-label">Password</label>
                <div class="col-md-6">
                    <input type="password" class="form-control" name="password">
                </div>
            </div>

            <div class="form-group">
                <label class="col-md-4 control-label">Confirm Password</label>
                <div class="col-md-6">
                    <input type="password" class="form-control" name="password_confirmation">
                </div>
            </div>
```

```
                <div class="form-group">
                    <div class="col-md-6 col-md-offset-4">
                        <button type="submit" class="btn btn-primary">
                            Reset Password
                        </button>
                    </div>
                </div>
            </form>
        </div>
    </div>
</div>
</div>
@endsection
```

بعد از اینکه کلمه عبور را تغییر دهید به طور اتوماتیک به صفحه کاربری خود ریدایرکت می شوید که این صفحه در لاراول home می باشد که می توانید با استفاده از پراپرتی redirectTo آن را تغییر دهید:

```
protected $redirectTo = '/dashboard';
```

در مثال بالا آن را به مسیر dashboard تغییر دادم.

در پایان باید یادتان باشد که تنظیمات مربوط به ایمیل برنامه تان را در فایل .env و config/mail.php به درستی اعمال کنید وگرنه ممکن است در ارسال ایمیل دچار خطا شوید.

افزودن کلاس و پکیج به لاراول

ممکن است شما کلاسی رو خودتون نوشته باشید و قصد دارین از اون توی فریمورک لاراول استفاده کنید. توی لاراول ۵ به راحتی میتونید از کلاستون استفاده کنید. یک پوشه توی پوشه app به نام Classesایجاد میکنیم و یک فایل مثلا به نام Common.php ایجاد میکنیم و کلاس Common رو داخلش تعریف میکنیم:

```
<?php namespace App\Classes;


class Common
{


  public static function pre($array)
  {
        echo '<pre>' . print_r($array, true) . '</pre>';
  }
```

همانطور که مشاهده میکنید ایتدا یک namespace برای کلاس تعریف کردم و برای مثال داخل کلاس متدی استاتیک به نام pre تعریف کردم. حالا هرجای پروژه به راحتی می تونید به این صورت با این متد کار کنید:

```
$cars = ['volvo', 'toyota', 'bmw'];

    \App\Classes\Common::pre($cars);
```

یا مثلا در کنترلر بهتره اونو use کنیم و اینجوری استفاده کنیم:

```
use App\Classes\Common;


class SiteController extends Controller
{
   public function index()
   {
      $cars = ['volvo', 'toyota', 'bmw'];
      return Common::pre($cars);
   }
//xxxxxxxxxxxxxxxxxxxxxxxxx xxxx
```

افزودن پکیج به لاراول

برای لاراول پکیج های زیادی نوشته میشه که شما میتونید با مراجعه به این آدرس پکیج مورد نظرتون سرچ کنید و اونو معمولا با composer به فریمورک اضافه کنید:

http://packalyst.com/

مثلا یکی از پکیج های خوبش اینه که کار با تصویر رو براتون آسون میکنه:

http://packalyst.com/packages/package/intervention/image

یا پکیج debug-bar لاراول که خیلی کاربردیه

http://packalyst.com/packages/package/barryvdh/laravel-debugbar

چند زبانه کردن برنامه

به طور پیش فرض در مسیر resources/lang یک پوشه به نام en وجود دارد که فایل های زبانی انگلیسی در آن قرار دارند. برای چند زبانی کردن کافی است به ازای هرتعداد زبان یک پوشه ایجاد کرده و دقیقا فایل های موجود در پوشه en را در آن هم کپی کنیم. هر فایل زبانی یک آرایه را return میکند که اندیس ها در همه فایل ها باید به انگلیسی باشند اما مقادیر آنها باتوجه زبان موردنظر مقداردهی می شوند. ما در این مثال قصد داریم یک برنامه ساده دوزبانه فارسی و انگلیسی را پیاده سازی کنیم پس با من همراه باشید .

ابتدا دوتا پوشه fa و en در پوشه resources/lang ایجاد کنید و فایل messages.php را ایجاد کنید به این صورت:

```
/resources
  /lang
    /en
      messages.php
    /fa
      messages.php
```

محتویات فایل های messages.php را هم به این صورت تغییر دهید:

```
برای فایل انگلیسی//
<?php


return [
  'welcome' => 'Welcome to our application',
  'text'    =>    'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
        tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
        quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
        consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
        cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
        proident, sunt in culpa qui officia deserunt mollit anim id est laborum.'
];
```

برای فایل فارسی//

```
return [
   'welcome' => 'به برنامه ما خوش آمدید',
   'text'    => '
```

لورم ایپسوم یا طرح‌نما (به انگلیسی Lorem ipsum: به متنی آزمایشی و بی‌معنی در صنعت چاپ، صفحه‌آرایی

و طراحی گرافیک گفته می‌شود. طراح گرافیک از این متن به عنوان عنصری از ترکیب بندی برای پر کردن صفحه

و ارایه اولیه شکل ظاهری و کلی طرح سفارش گرفته شده استفاده می نماید، تا از نظر گرافیکی نشانگر

چگونگی نوع و اندازه فونت و ظاهر متن باشد. معمولا طراحان گرافیک برای صفحه‌آرایی، نخست از متن‌های

آزمایشی و بی‌معنی استفاده می‌کنند تا صرفا به مشتری یا صاحب کار خود نشان دهند که صفحه طراحی یا

صفحه بندی شده بعد از اینکه متن در آن قرار گیرد چگونه به نظر می‌رسد و قلم‌ها و اندازه‌بندی‌ها

چگونه در نظر گرفته شده‌است. از آنجایی که طراحان عموما نویسنده متن نیستند و وظیفه رعایت حق

تکثیر متون را ندارند و در همان حال کار آنها به نوعی وابسته به متن می‌باشد آنها با استفاده

از محتویات ساختگی، صفحه گرافیکی خود را صفحه‌آرایی می‌کنند تا مرحله طراحی و صفحه‌بندی را به

پایان برند.

```
',
];
```

حالا یک فایل view به نام test.blade.php هم ایجاد کنید و کدهای زیر را در آن قرار دهید:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Multi Languages-Hamo</title>
</head>
<body>

    <a href="{{ url('/language') }}">{{ Session::get('locale', 'fa') == 'fa' ? 'English' : 'فارسی' }}</a>
    <h1>{{ trans('messages.welcome') }}</h1>
    <p>{{ trans('messages.text') }}</p>

</body>
</html>
```

با استفاده از تابع کمکی trans متن موردنظرمان را چاپ میکنیم. مقداری که این تابع میگرد به این صورت است که ابتدا نام فایل بعد یک نقطه و سپس اندیس آرایه را به آن میدهیم و مقدار ان براساس localeبرنامه چاپ می شود مثلا اگر زبان برنامه fa باشد welcome و text فارسی وگرنه انگلیسی نمایش داده خواهد شد. یک لینک هم برای تغییر زبان تعریف کردم که با یک سشن به نام locale که جلوتر تعریف میکنیم زبان جاری را در آن قرار می دهیم و مقدار لینک بر اساس نوع زبان جاری تعیین می شود مثلا اگر زبان فارسی بود لینک زبان انگلیسی نمایش داده می شود و بالعکس این لینک را میتوانید در منوی وبسایت و در قالب اصلی آن قراردهید که بستگی به سلیقه خودتان دارد. این لینک به مسیر language هدایت میشود پس مسیر را در فایل routes.php تعریف میکنیم:

```
//language

Route::get('/language', 'WelcomeController@language');
```

مسیر را به کنترلر WelcomeController و اکشن language ارسال کردیم. پس اکشن language را در کلاس کنترلر موردنظر ایجاد میکنیم:

```
public function language(ChangeLocaleCommand $ChangeLocaleCommand)
  {
     $this->dispatch($ChangeLocaleCommand);
     return redirect()->back();
  }
```

همانطور که می بینید از یک کلاس command استفاده کردیم که در این کلاس نوع زبان را در سشن تغییر می دهیم. یادتان باشد برای استفاده از هرکلاسی در کنترلر باید ابتدا آن را با namespace آن ایمپورت کنید پس ایتدای کلاس این عبارت را اضفه کنید:

```
use App\Commands\ChangeLocaleCommand;
```

اکنون با دستور زیر کلاس ChangeLocaleCommand را ایجاد کنید:

```
php artisan make:command ChangeLocaleCommand
```

در مسیر app/commands کلاس ایجاد شده را باز کنید و متد handle را به این صورت تغییر دهید:

```
<?php namespace App\Commands;

use App\Commands\Command;

use Illuminate\Contracts\Bus\SelfHandling;

class ChangeLocaleCommand extends Command implements SelfHandling {

   /**
    * Execute the command.
    *
    * @return void
    */
   public function handle()
   {
      session()->set('locale', session('locale') == 'fa' ? 'en' : 'fa');
   }

}
```

همانطور که می بینید در کلاس بالا مقدار سشن اگر fa بود به en یا بالعکس تغییر میدهیم.

حالا باید یک کلاس command دیگر برای ست کردن locale برنامه ایجاد کنیم پس با دستور زیر آن را ایجاد میکنیم:

```
php artisan make:command SetLocaleCommand
```

حالا فایل آن را بازکنید و به این صورت تغییر دهید:

```
<?php namespace App\Commands;

use App\Commands\Command;
use Request;
use Illuminate\Contracts\Bus\SelfHandling;

class SetLocaleCommand extends Command implements SelfHandling {

   /**
    * The availables languages.
    *
    * @array $languages
    */
   protected $languages = ['en','fa'];

   /**
    * Execute the command.
    *
    * @return void
    */
   public function handle()
   {
      if(!session()->has('locale'))
      {
         session()->put('locale', Request::getPreferredLanguage($this->languages));
      }
      app()->setLocale(session('locale'));
   }
}
```

همانطور که مشاهده میکنید در کلاس بالا locale برنامه را با توجه مقدار سشن تغییر می دهیم اگر مقدار سشن fa بود پس locale برنامه هم fa و یا بالعکس شود. در صورتی که سشن ست نشده بود با استفاده از متد getPreferredLanguage و پراپرتی languages که زبان های موردنظرمان را در ان ست کردیم آن را مقداردهی میکنیم.

حالا بایستی از این کلاس command استفاده کنیم پس یک middleware هم در مسیر app/Http/Middlewareبه نام App ایجاد میکنیم با دستور زیر:

```
php artisan make:middleware App
```

فایل آن را باز کنید و به این صورت تغییر دهید:

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

use Closure;

use App\Commands\SetLocaleCommand;

use Illuminate\Bus\Dispatcher as BusDispatcher;

class App {

  /**
   * The command bus.
   *
   * @array $bus
   */
  protected $bus;
```

```
/**
 * The command bus.
 *
 * @array $bus
 */
protected $setLocaleCommand;


/**
 * Create a new App instance.
 *
 * @param  Illuminate\Bus\Dispatcher $bus
 * @param  App\Commands\SetLocaleCommand $setLocaleCommand
 * @return void
*/
public function __construct(
   BusDispatcher $bus,
   SetLocaleCommand $setLocaleCommand)
{
   $this->bus = $bus;
   $this->setLocaleCommand = $setLocaleCommand;
}


/**
 * Handle an incoming request.
 *
 * @param  Illuminate\Http\Request  $request
 * @param  Closure  $next
 * @return mixed
 */
public function handle($request, Closure $next)
{
```

```
        $this->bus->dispatch($this->setLocaleCommand);


        return $next($request);
    }


}
```

در سازنده کلاس دوتا پراپرتی $bus و $setLocalCommand را با نمونه از کلاس های BusDispatcher و SetLocaleCommandمقداردهی کردیم و در متد handle کلاس SetLocaleCommand را برای اجرای فرمان به bus میدهیم .

در پایان باید این middleware را به فریمورک معرفی کنیم همانظور که در بخش موردنظرش هم توضیح دادم چون میخواهم این middleware عمومی باشد و در کل برنامه اجرا شود پس در فایل Kernel.phpدر مسیر app/Http به پراپرتی $middleware که مقدارش آرایه است این middleware را هم اضافه کنید.

```
protected $middleware = [
    'Illuminate\Foundation\Http\Middleware\CheckForMai  ntenanceMode',
    'Illuminate\Cookie\Middleware\EncryptCookies',
    'Illuminate\Cookie\Middleware\AddQueuedCookiesToRe  sponse',
    'Illuminate\Session\Middleware\StartSession',
    'Illuminate\View\Middleware\ShareErrorsFromSession  ',
    'App\Http\Middleware\VerifyCsrfToken',
    'App\Http\Middleware\App',
];
```

اگر فایل test.blade.php را رندر کنید به راحتی با کلیک کردن بر لینک تغییر زبان می توانید زبان برنامه را تغییر دهید. امیدوارم که مطلب برایتان مفید باشد.

مثالی کاربردی از AJAX در لاراول

امروز با توجه به درخواست یکی از دوستان یک مثال کاربردی با AJAX را کار خواهیم کرد. برای کار با AJAX شما ابتدا باید با جاوااسکریپت و یا یکی از کتابخانه های جاوااسکریپت آشنایی داشته باشید. من در این مثال از jQuery برای کار با AJAX استفاده میکنم.

ابتدا دوتا route در فایل routes.php ایجاد میکنیم:

```
Route::get('ajax-form', 'WelcomeController@ajaxForm');

Route::post('ajax-form', 'WelcomeController@postAjaxForm');
```

داخل کلاس WelcomeController متدهای ajaxForm برای درخواست های GET و متد postAjaxForm را برای درخواست های POST ایجاد میکنیم. متد ajaxForm فرم را به این صورت می نویسیم:

```
public function ajaxForm()
  {
    return  view('ajax');
  }
```

حالا یک فایل ویو به نام ajax.blade.php در مسیر resources/views ایجاد میکنیم و کدهای زیر را در آن قرار دهید:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>AJAX Example</title>
  <style type="text/css">
    .error {
      color: red;
      font-weight: bold;
    }
```

```
        .success {
            color: green;
            font-weight: bold;
        }
    </style>
</head>
<body>
<form action="{{ url('/ajax-form') }}" method="post" id="form1">
    <input type="hidden" name="_token" value="{{ csrf_token() }}">
    Your Name:<input type="text" name="name"> <span class="error" id="name"></span>
    Your Email:<input type="text" name="email"> <span class="error" id="email"></span>
    Your Website:<input type="text" name="website"> <span class="error" id="website"></span>
    <input type="submit" value="Submit">
</form>
<div id="results" class="success"></div>

<script src="{{ asset('js/jquery.js') }}"></script>
<script type="text/javascript">
    $(document).ready(function (){
        $("#form1").submit(function (event){
            event.preventDefault();
            var $this = $(this);
            var url = $this.attr('action');

            $.ajax({
                url: url,
                type: 'POST',
                dataType: 'JSON',
                data: $this.serialize(),
            })
            .done(function( response ) {
```

```
            $('span').empty();
            $("div#results").empty();
            $.each(response, function(index, val) {
                /* iterate through array or object */
                switch(index){
                    case "name":
                        $('span#name').html(val);
                        break;
                    case "email":
                        $("span#email").html(val);
                        break;
                    case "website":
                        $("span#website").html(val);
                        break;
                    case "success":
                        $("div#results").html(val);
                        break;
                }
            });
        })
        .fail(function() {
            console.log('error');
        });


    });
  });
</script>
</body>
</html>
```

همانطور که مشاهده می کنید ابتدا در css دوتا کلاس error و sucess برای نمایش زیباتر پاسخ و پیغام های خطا تعریف کردم. فرم موردنظرمان را ایجاد کردیم و کتابخانه جی کوئری را به صفحه ضمیمه کرده و در نهایت کد AJAX را نوشتم . در کد ای جکس متد درخواست یا همون پارامتر type رو POST قراردادم و همچنین نوع داده ای که می خواهیم دریافت کنیم رو JSON گزاشتم چون قرار است پاسخ ما از نوع json باشد url .آن راهم به صورت داینامیک تعریف کردم و از خصوصیت action فرم گرفتم که شما می توانید به صورت دستی هم مقدار بدهید چون من میخواهم این کد ای جکسم قابل استفاده مجدد در هرپروژه دیگر باشد اینکار را کردم. همه داده های فرم راهم به صورت سریالایز فرستادم که بازهم شما می توانید به صورت دستی اینکار را بکنید. متد done هم اگر درخواست ای جکس با موفقیت به پایان برسد پاسخ دریافتی رو دریافت میکنیم و متد fail هم در صورتی که کد ای جکس دارای خطا باشد عملیات موردنظرمان را در آن می نویسیم.

قبل اینکه کدهای داخل متد done رو توضیح بدم بریم اکشن postAjaxForm در کنترلر WelcomeControllerرو مشاهده کنیم:

```
public function postAjaxForm(Request $request)
{
    if($request->ajax()){
        //validation
        $rules = [
            'name' => 'required|min:3|max:100',
            'email' => 'required|email',
            'website' => 'url'
        ];

        $v = Validator::make($request->all(), $rules);
        if($v->fails()){
            return Response::json($v->errors());
        } else {
            $html = '<p>Your name: '. $request->input('name') .'</p>';
            $html .= '<p>Your email: '. $request->input('email') .'</p>';
            $html .= '<p>Your website: '. $request->input('website') .'</p>';
            return Response::json(['success'=> $html]);
        }
```

```
    } else {
        return 'Request invalid!';
    }
}
```

ابتدا بررسی کردیم اگر نوع درخواست ajax بود ادامه کار را انجام دهیم و سپس داده های فرم را اعتبارسنجی کردیم و اگر اعتبار سنجی دارای خطا بود خطا ها را با استفاده از کلاس Response و متد json به صورت JSON تولید میکنیم. خطاها را به عنوان آرگومان به این متد می دهیم. اندیس ها نام فیلد فرم و مقدار آن پیغام خطای مورد نظر است.

در صورتی که اعتبارسنجی بدون خطا باشد یک پاسخ در قالب json برای تست ایجاد کردم به این صورت که آرایه ای که به متد json دادم اندیس آن را success قرار دادم که به این اندیس ها در کد ای جکس نیاز پیدا میکنیم.

حالا یکبار دیگه داخل کد AJAX متد done رو نگاه کنید:

```
.done(function( response ) {
        $('span').empty();
        $("div#results").empty();
        $.each(response, function(index, val) {
            /* iterate through array or object */
            switch(index){
                case "name":
                    $('span#name').html(val);
                    break;
                case "email":
                    $("span#email").html(val);
                    break;
                case "website":
                    $("span#website").html(val);
                    break;
                case "success":
                    $("div#results").html(val);
```

```
                break;
            }
        });
    })
```

همانطور که می بینید response ای که دریافت کردیم را به راحتی میتوانیم مدیریت کنیم. ابتدا مقدار spanها و تگ با آیدی results را در هربار که درخواست ای جکس ارسال میکنیم خالی میکنیم تا پیغام های درخواست قبلی پاک شوند سپس همانند دستور foreach در php با استفاده از دستور each می توانیم به اندیس ها و مقادیر هر آیتم دسترسی داشته باشیم. من در این مثال از switch استفاده کردم تا هر اندیسی که به عنوان پاسخ برایمان ارسال شده را در جای مناسب خودش نمایش بدهم .index همان اندیس مورنظرمان و val هم مقدار آن می باشد.

بعضی مواقع شما درخواست های ای جکس را بدون استفاده از فرم ارسال میکنید در نتیجه چون توکنی ارسال نمیکنید middleware عمومی که توکن را بررسی میکند درخواست را reject میکند برای حل این مشکل کافی است توکن را مثلا در تگ meta تولید کرده سپس و با استفاده از ajaxSetup توکن را به هدر اضافه کنید:

کد:HTML

```
<meta name="csrf-token" content="{{ csrf_token() }}" />
```

```
$.ajaxSetup({
    headers: {
      'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
  });
```

کار با کلاس های Html و Form

دو تا کلاس در لاراول ۴ بودند که کار رو برای نوشتن کدهای تکراری HTML برایمان آسان تر میکردن که در لاراول ۵ به طور پیش فرض وجود ندارد و باید به آن اضافه کنیم. چون در خیلی از مثال های موجود در وب از این کلاس ها استفاده شده لازم دیدم این کلاس ها را معرفی کنم. ابتدا نحوه افزودن آن به لاراول ۵ رو توضیح میدم.

در پوشه اصلی لاراول ۵ یک فایل به نام composer.json را باز کرده و خط "illuminate/html": "5.*" به بخش require اضافه کنید به این صورت:

```
"require": {
    "illuminate/html": "5.*",
    "laravel/framework": "5.0.*"
  },
```

سپس در ترمینال به پوشه پروژه خود رفته و دستور زیر را اجرا کنید:

```
composer update
```

سپس فایل app.php در پوشه config را باز کنید و به انتهای آرایه providers مقدار 'Illuminate\Html\HtmlServiceProvider'را اضافه کنید به این صورت:

```
'providers' => [
  /* more already here */
  'Illuminate\Html\HtmlServiceProvider',
```

همچنین دو خط زیر را هم به انتهای آرایه aliases اضافه کنید به این صورت:

```
'aliases' => [
  /* more already here */
   'Html' => 'Illuminate\Html\HtmlFacade',
   'Form' => 'Illuminate\Html\FormFacade',
 ],
```

اکنون می توانید از این کلاس ها در برنامه خودتان در view ها استفاده کنید. از هرکدام چند تا از کاربردی هایش را مثال خواهم زد.

کلاسHtml

ایجاد تگ :script از متد script استفاده میکنید و پارامتر اولی مسیر اسکریپت و پارامتر دوم هم که به صورت آرایه هست شامل attribute های تگ می باشد

```
{!! Html::script('js/jquery.js', ['type' => 'text/javascript']) !!}
```

```
//output: <script type="text/javascript" src="http://laravel.dev/js/jquery.js"></script>
```

ایجاد تگ : link مشابه دستور بالا است و تگ لینک را ایجاد میکند

```
{!! Html::style('css/style.css') !!}
```

```
//output:  <link media="all" type="text/css" rel="stylesheet" href="http://laravel.dev/css/style.css">
```

ایجاد تگ : image با استفاده از متد image و پارامتر اول مسیر تصویر و پارامتر دوم متن جایگزین و پارامتر سوم هم attribute ها می باشند.

```
{!! Html::image('images/1.jpg' , 'alternate', ['class' => 'img'])!!}
```

```
//output:  <img src="http://laravel.dev/images/1.jpg" class="img" alt="alternate">
```

ایجاد تگ : a با استفاده از متد link و پارامتر اول url مورد نظر و پارامتر دوم عنوان تگ که اگر null قرار دهیم همان url عنوان در نظر گرفته می شود و پارامتر سوم هم attribute های تگ می باشند.

```
{!! Html::link('user/profile', 'Go User Profile', ['calss' => 'btn btn-primary']) !!}
```

```
//output: <a href="http://laravel.dev/user/profile" calss="btn btn-primary">Go User Profile</a>
```

ایجاد تگ ul: با استفاده از متد ul و پارامتر اول آرایه ای از لیست ها و پارامتر دوم هم آرایه ای از attributeها می باشد.

```
{!! Html::ul(['Item1', 'Item2', 'Item3'], ['calss' => 'nav']) !!}

//output:  <ul calss="nav"><li>Item1</li><li>Item2</li><li>Item3</li></ul>
```

کلاسForm

ایجاد تگ شروع و پایان : form با استفاده از متد open و یک پارامتر آرایه ای میگیرد که attribute ها را در آن ست میکنیم url .همان اکشن فرم را مقداردهی میکند و همچنین files اگر true قرار دهیم فرم برای آپلود فایل مناسب می باشد. این متد فیلد توکن را هم ایجاد میکند. متد close هم تگ فرم را می بندد.

```
{!! Form::open(['url' => 'conatct', 'id'=>'form1', 'files'=> true]) !!}

//output:  <form method="POST" action="http://laravel.dev/conatct" accept-charset="UTF-8"
id="form1" enctype="multipart/form-data"><input name="_token" type="hidden"
value="FzsNKPfXXLbuD1YoMCfgJXbEsYW7Z2CTAohEyiG0">


{!! Form::close() !!}
```

ایجاد تگ label و input از نوع : text در متد label پارامتر اول نام فیلدی است که میخواهیم برای آن لیبل تعریف کنیم و پارامتر دوم مقدار لیبل است و پارامتر سوم هم آرایه ای از attribute ها می باشد. در صورتی که خصوصیت id برای فیلد متناظر تگ label تعریف نکرده باشیم به طور خودکار id با مقدار هم نام با فیلد input متناظرش ایجاد خواهد کرد. در متد text هم یک input از نوع text ایجاد کرده که پارامتر اول آن نام آن و پارامتر دوم مقدار آن و پارامتر سوم هم آرایه ای از attribute ها می باشد.

```
{!! Form::label('name', 'Your Name', ['class' => 'label']) !!}

//output: <label class="label" for="name">Your Name</label>

{!! Form::text('name', null, ['calss' => 'test']) !!}

//output: <input id="name" type="text" name="name" calss="test">
```

نحوه تعریف متدهای hidden, email, url, textarea, number هم مشابه متد text می باشد.

ایجاد input از نوع : file پارامتر اول آن نام فیلد و پارامتر دوم هم آرایه ای از attribute ها است.

```
{!! Form::file('photo') !!}
//output: <input type="file" name="photo">
```

ایجاد input از نوع :submit پارامتر اول آن مقدار فیلد و پارامتر دوم آرایه ای از attribute ها است.

```
{!! Form::submit('Register' , ['calss' => 'btn btn-primary']) !!}
//output: <input type="submit" value="Register" calss="btn btn-primary">
```

افزودن Captcha و کار با آن

برای کار با کپچا شما می توانید از کپچاهای معروف زیادی همچون reCaptcha استفاده کنید اما من از یک کپچای خوب که کار با آن آسان است در این آموزش استفاده خواهم کرد .

ابتدا با ترمنیال به مسیر پروژه لاراول بروید و دستور زیر را تایپ و اجرا کنید:

```
composer require mews/captcha
```

سپس در فایل config/app.php بخش providers خط زیر را به انتهای آن اضافه کنید:

```
'Mews\Captcha\CaptchaServiceProvider',
```

همچنین خط زیر را هم به انتهای بخش aliases اضافه کنید:

```
'Captcha' => 'Mews\Captcha\Facades\Captcha',
```

در پایان هم دستور زیر را احرا کنید:

```
php artisan vendor:publish
```

اکنون می توانیم از کپچا در فرم ها استفاده کنیم. با یک مثال کاربردی نحوه استفاده از آن را توضیح خواهم داد. طبق خواسته یکی از کاربران این مثال را در فرم لاگین انجام می دهم.

ابتدا فایل login.blade.php در مسیر resources/views/auth رو به این صورت ویرایش میکنیم:

کد:HTML

```
<form class="form-horizontal" role="form" method="POST" action="{{ url('/auth/login') }}">
                <input type="hidden" name="_token" value="{{ csrf_token() }}">

                <div class="form-group">
                  <label class="col-md-4 control-label">E-Mail Address</label>
                  <div class="col-md-6">
                    <input type="email" class="form-control" name="email" value="{{ old('email') }}">
```

```
    </div>
</div>

<div class="form-group">
    <label class="col-md-4 control-label">Password</label>
    <div class="col-md-6">
        <input type="password" class="form-control" name="password">
    </div>
</div>

<div class="form-group">
    <label class="col-md-4 control-label">Captcha Code</label>
    <div class="col-md-6">
        {!! Captcha::img('flat') !!}
        <input type="text" class="form-control" name="captcha">
    </div>
</div>

<div class="form-group">
    <div class="col-md-6 col-md-offset-4">
        <div class="checkbox">
            <label>
                <input type="checkbox" name="remember"> Remember Me
            </label>
        </div>
    </div>
</div>

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

```
                    <button type="submit" class="btn btn-primary">Login</button>

                    <a class="btn btn-link" href="{{ url('/password/email') }}">Forgot Your Password?</a>
                </div>
            </div>
        </form>
```

همانطور که مشاهده می کنید با دستور Captcha::img('flat') تصویر کپچا نمایش داده می شود و من چون می خواستم ابعاد تصویر کمی بزرگتر باشد از flat استفاده کردم که می توانید در فایل config/captcha.phpتنظیمات مربوط به ابعاد و رنگ ها را به دلخواه خودتان تغییر دهید.

حالا یک کلاس Request هم برای اعتبار سنجی با استفاده از دستور زیر ایجاد میکنیم:

```
php artisan make:request LoginRequest
```

قوانین اعتبار سنجی را در متد rules قرار می دهیم و متد authorize را هم true می کنیم چون نیازی به احراز هویت در این درخواست نمی باشد. کلاس LoginRequest در مسیر app/Http/Requests باید مشابه مثال زیر ویرایش شود:

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

use App\Http\Requests\Request;

class LoginRequest extends Request {

    /**
     * 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()
    {
        return [
            'email'    => 'required|email',
            'password' => 'required|min:3|max:17',
            'captcha'  => 'required|captcha',
        ];
    }

}
```

همانطور که می بینید برای فیلد captcha یک قانون اعتبارسنجی جدید به نام captcha قرار دادیم که میاد ورودی کاربر را با کد داخل تصویر مطابقت میده و در صورتی که مغایرت داشته باشد خطایی صادر میکند. پس باید داخل مسیر resources/lang/en فایل validation.php را باز کنیم و برای آن یک پیغام مناسب ست میکنیم. پس خط زیر را به آن اضافه کنید:

```
"captcha"              => "The :attribute field entered is wrong",
```

در ادامه باید در کنترلر AuthController در مسیر app/Http/Controllers/Auth متد postLogin را بازنویسی کنیم. ابتدا به کنترلر کلاس های زیر را ایمپورت می کنیم:

```
use App\Http\Requests\LoginRequest;

use Request;

use Auth;
```

حالا متد postLogin را به این صورت اضافه کنید:

```
public function postLogin(LoginRequest $request)

  {

    $credentials = Request::only('email', 'password');


    if(Auth::attempt($credentials)){


      return redirect()->intended('/home');


    } else {


      return redirect()->back()->withErrors(['invalid' => 'The username or email invalid!']);

    }

  }
```

خب دیگه مثال به پایان رسید و الان می تونید از کپچا در فرم هایتان به همین آسانی استفاده کنید. فقط دیدم این کپچا برای رفرش کردن کد توسط کاربر گزینه ای نداره خودم یک تابع با جی کوئری براش نوشتم . ابتدا یک تصویر برای رفرش کپچا در کنار تصویر کپچا قرار دادم:

کد:HTML

```
<img src="{{ asset('img/refresh.svg') }}" alt="refresh" id="refresh" width="50" height="50">
```

در انتها هم دستورات jquery زیر را به صفحه اضافه کردم تا با هربار کلیک روی تصویر رفرش کپچا تغییر کند.

```
<script type="text/javascript">
  $('document').ready(function(){
    $('#refresh').click(function(){
      var src = "{{ Captcha::src() }}";
      var timestamp = new Date().getTime();
      $("img[alt=captcha]").attr("src", src + '?' +timestamp);
    });
  });
</script>
```

۱۰ پکیج کاربردی فریم ورک Laravel

احتمالاً برای شما پیش آمده است که در گیر پروژه ای باشید و برای نوشتن بخش‌هایی از پروژه با مشکل رو برو شوید یا حتی فرصت و زمان انجام اون بخش پروژه را نداشته باشید. شاید هم حال نوشتن اون بخش رو نداشتید. معمولاً در اینترنت دنبال راه حل می‌گردیم که نیاز اون بخش خاص از پروژه را بر طرف کنید. در این پست قصد دارم ۱۰ پکیج کاربری لاراول را معرفی کنم، امیدوارم در پروژه های بعدی خودتان از این پکیج های کاربردی استفاده کنید.

۱- Laravel 5 log viewer

به وسیله این پکیج می‌توانید رخدادهای سیستم خود را مدیریت و کنترل نمایید خوشبختانه این پکیج قابلت نصب بر روی نسخه های ۴ و ۵ لاراول را دارا می‌باشد.

۲- Laravel Breadcrumbs 3

معمولاً برای نمایش موقعیت فعلی هر کاربر از Breadcrumb استفاده می‌نماییم با استفاده از این پکیج به راحتی می‌توانید Breadcrumb را در سیستم خود پیاده‌سازی نمایید.

برای مطالعه مستندات این پکیج به صفحه Laravel breadcrumbs 3 مراجعه نمایید.

۳- Backup mangaer

اطلاعات دیتابیس ها از مهمترین بخش‌های هر سیستم به حساب می‌آیند، زیرا اگر آن‌ها را از دست بدهیم جایگاه فعلی سیستم زیر سؤال می‌رود. چنانچه قصد داشته باشید از دیتابیس سیستم خود پشتیبان تهیه کنید پکیج Backup manager می‌تواند شما را در این امر یاری نماید.

۴- noCAPTCHA – new reCAPTCHA

این پکیج شمارا قادر می‌سازد تا به سادگی کپچای گوگل را در سیستم خود استفاده نمایید.

ReCAPTCHA سرویس رایگانی است که سایت شما را از سوءاستفاده اسپمر ها محافظت می‌نماید. برای کسب اطلاعات بیشتر در مورد نحوه تشخیص و الگوریتم شناسایی این سرویس به پست روش جدید recaptcah برای محافظت از سایت شما که چند ماه پیش پیش توسط محسن عزیز نوشته شد مراجعه نمایید.

۵- Laravel Excel

یک پکیج قدرتمند جهت دست و پنجه نرم کردن با فایل‌های اکسل!

برخی از امکانات این پکیج عبارتند از:

- ورد اطلاعات از فایل اکسل به سیستم
- خروجی گرفتن اطلاعات به صورت فایل CSV و Excel
- قابلیت سفارشی سازی
- ورود اطلاعات به صورت دسته‌جمعی
- ویرایش فایل‌های اکسل

۶- Laravel Setting

این پکیج شما را قادر می‌سازد تا تنظیمات مورد نیاز پروژه تان را راحت‌تر مدیریت نمایید .

۷- Laravel Migrations Generator

عموما قبل از شروع هر پروژه دیتابیس آن را طراحی و پیاده‌سازی می‌کنیم. چنانچه قصد داشته باشیم از دیتابیس ایجاد شده Migrate ایجاد نماییم و دیتابیس مورد نظر جداول زیادی داشته باشد تولید Migrate آن بسیار کسل کننده خواهد بود. اگر قصد داشته باشید این کار به صورت خودکار انجام شود به شما پیشنهاد می‌کنم از این پکیج استفاده کنید.

۸- Snappy PDF/Image Wrapper

جهت تبدیل داده‌های خود به عکس و یا فایل PDF می‌توانید از امکانات این پکیج بهره ببرید..

۹- Laravel Messenger

به کمک این سیستم می‌توانید در پروژه لاراول خود از سیستم پیام رسان استفاده نمایید.

برخی از ویژگی‌های این سیستم به شرح زیر است:

- امکان گفتگو همزمان برای هر کاربر
- قابلیت سفارشی سازی سطح دسترسی
- قابلیت ایجاد تالار گفتگو
- قابلیت ایجاد موضوع جدید
- قابلیت ارسال پیام شخصی و عمومی
- و برخی امکانات دیگر

۱۰- Laravel HTMLMin

Laravel HTMLMin پکیج کار آمدی است که حجم صفحات وب‌سایت شما را کاهش می‌دهد و امکانات مناسبی را برای فشرده‌سازی HTML، CSS و JS در اختیار شما قرار می‌دهد.

آموزش ساخت یک وبلاگ ساده با لاراول

خب در این مقاله ما قصد داریم یه وبلاگ ساده با استفاده از laravel 5 ایجاد کنیم . وبلاگی که خواهان ایجاد اون هستیم دارای ویژگی های زیر می باشد :

- نمایش پست با لینک ادامه مطلب در صفحه اصلی.
- قابلیت سرچ در پست های وبسایت توسط کاربر .
- نمایش یک پست کامل به همراه بخش نظرات .
- مدیر توانایی درج ، حذف ، آپدیت و ویرایش پست ها و نظرات رو داشته باشه .
- مدیر توانایی جواب دادن نظرات در پنل مدیریت رو داره باشه .

قدم اول : نصب و راه اندازی سریع laravel 5

ما فرض رو بر این میزاریم که شما کاملا با نصب و راه اندازی laravel 5 آشنایی دارید و مشغول به کار با اونید اگه اینطوره نیاز به بخش زیر ندارید . در غیر این صورت میتونید بخش زیر رو دنبال کنید .

- دستورالعمل نصب laravel 5 رو میتونید در این صفحه پیدا کنید : **اینجا**
- ساخت یه دیتابیس با استفاده از تریمنال mysql

```
┌─]usm4n@usm4n-desktop[─]~[
└─•mysql -u root -p
Enter password:
mysql< create database laravel;
Query OK, ١ row affected ٠.٠٠) sec(
```

پیکربندی پایگاه داده اتون در بخش /config/database.php :

```
'mysql' <= array)
    'driver'    <= 'mysql,'
    'host'      <= 'localhost,'
    'database'  <= 'laravel,'
    'username'  <= 'root,'
    'password'  <= 'very_secret_password,'
    'charset'   <= 'utf8,'
    'collation' <= 'utf8_unicode_ci,'
    'prefix'    <= ,''
,(
```

ایجاد جداول مختلف با استفاده از Migrations :

در این بخش ، ما جدول های پایگاه داده مورد استفاده در وبلاگمونو با استفاده از Migrations ایجاد میکنم . برنامه ما از جدول Posts و comments برای ارسال پست روی وبلاگ و بخش نظرات استفاده میکنه . (البته اگه اطلاع زیادی در مورد Migrations ندارید میتونید از **این پست** استفاده کنید تا باهاش آشنا بشین)

نکته سریع : ما از دستورات `artisan migrate:make create_tablename_table` و `artisan migrate` به ترتیب برای ایجاد Migrations و اجرایی اون Migrations استفاده میکنیم .

کلاس Migrations برای جدول Posts :

```
?>php

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

class CreatePostsTable extends Migration
}

    **/
 *   Run the migrations.
*
@ *   return void
/*
    public function up()
    }
        Schema::create')posts,' function )Blueprint $table( }
            $table<-increments')id;('
            $table<-string')title;('
            $table<-string')read_more;('
            $table<-text')content;('
            $table<-unsignedInteger')comment_count;('
            $table<-timestamps;()
            $table<-engine = 'MyISAM;'
        ;({
        DB::statement')ALTER TABLE posts ADD FULLTEXT search(title,
content;('(
    {


    **/
 *   Reverse the migrations.
*
@ *   return void
/*
    public function down()
    }
        Schema::table')posts,' function )Blueprint $table( }
            $table<-dropIndex')search;('
            $table<-drop;()
        ;({
    {

{
```

این کلاس در بخش /database/Migrations/ قرار میگیره .

توجه داشته باشید که من با استفاده کردن از $table->engine = 'MyISAM' و با اضافه کردن شاخص مرکب به ستون های title, content این قابلیتو ایجاد کردم که کاربر وقتی سرچ میکنه مطلب مورد نظرش هم تو متن و هم تو عنوان پست جستجو بشه .

کلاس Migrations برای جدول comments

```
?>php
```

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

class CreateCommentsTable extends Migration
}

    **/
 *  Run the migrations.
*
@ *  return void
/*
    public function up()
    }
        Schema::create')comments,' function )Blueprint $table( }
            $table<-increments')id;('
            $table<-unsignedInteger')post_id;('
            $table<-string')commenter;('
            $table<-string')email;('
            $table<-text')comment;('
            $table<-boolean')approved;('
            $table<-timestamps;()
        ;({
    {

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

{
```

ما با ایجاد فیلد **post_id** این قابلتو برای خودمون ایجاد میکنیم که بتونیم بین دوتا جدول ارتباط ایجاد کنیم و با استفاده از **Eloquent ORM** به هدفه خودمون برسیم . و همینطور از فیلد **approved** برای تایید نظرات کاربرا توسط مدیر استفاده میکنیم که آیا نظر قابل مشاهده تو سایت هست یا خیر . البته یه جدول دیگه هم با اسم **users** و با استفاده از Migrations باید بسازین برای احراز هویت مدیر یا کاربراتون . که این جدول از قبل وقتی که پروژه laravel 5 رو نصب و راه اندازی کردین تو پروژه اتون قرار داده شده .

ایجاد Models با استفاده از Eloquent ORM :

Eloquent ORM همراه با خود فریم ورک لاراول وجود داره و میتونه به سادگی و خیلی زیبا با استفاده از ActiveRecord با جداول پایگاه دادتون کار کنه و عملیات های مورد نظرتونو به بهترین وجه انجام بده . هر جدول که در پایگاه دادتون وجود داره دارای یه Model جدا در پروژتون می باشد که این باعث تعامل بهتر با جداول میشه .

ما از اسم های ساده تری برای جدول ها بعنوان نام Eloqent Model استفاده میکنم . تا خوانایی کد ها بالا بره. این قرار داد کمک میکنه تا Eloquent بتونه به صورت جدول با Model ارتباط برقرار کنه . برای مثال اگه اسم Eloqent Model ما Post باشه ما از اسم جدول posts استفاده میکنیم .

در زیر کد های بخش Post و Comment رو قرار میدیم :

```
?>php
//file: app/Post.php
class Post extends Model }

    public function comments()
    }
        return $this<-hasMany')Comment;('
    {

{
//file: app/Comment.php
class Comment extends Model }

    public function post()
    }
        return $this<-belongsTo')Post;('
    {
{
```

نکته : کد های که تو کلاس این دو مدل استفاده شدن رو بعدا بهتون توضیح میدم که چیه فقط فعلا بدونین که اینا رو هم باید وارد کنید .

وارد کردن اطلاعات به جداول دیتابیس با استفاده از Seeding :

ما از یه کلاس به اسم PostCommentSeeder برای پر کردن جدول های posts و comments استفاده میکنیم .

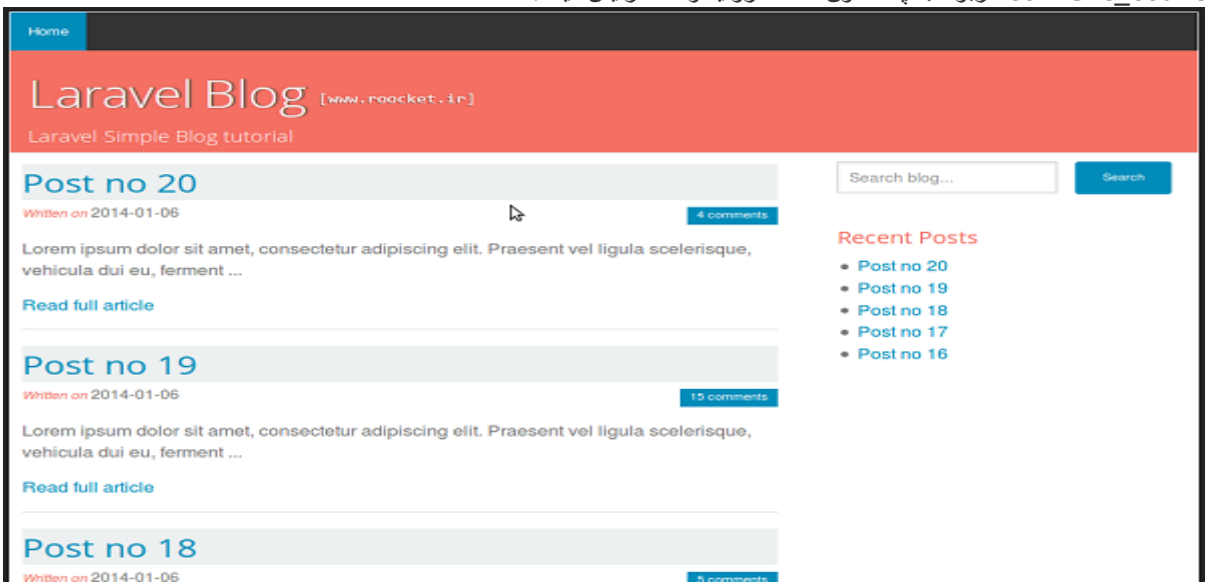
نکته سریع : برای اجرای Seed از کد php artisan db:seed در ترمینالمون استفاده کنید و اگه بازم اطلاعاتی در مورد Seeding ندارید تو پست های آینده بطور کامل توضیح میدیم که چیه و چیکار میکنه .

کد PostCommentSeeder :

```
 ?>php

use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;
use App\post;
use App\Comment;

class PostCommentSeeder extends Seeder }

    public function run()
    }
        $content = 'Lorem ipsum dolor sit amet, consectetur adipiscing
elit.
                    Praesent vel ligula scelerisque, vehicula dui eu,
fermentum velit.
                    Phasellus ac ornare eros, quis malesuada augue.
Nunc ac nibh at mauris dapibus fermentum.
```

```
                    In in aliquet nisi, ut scelerisque arcu. Integer
tempor, nunc ac lacinia cursus,
                    mauris justo volutpat elit,
                    eget accumsan nulla nisi ut nisi. Etiam non
convallis ligula. Nulla urna augue,
                    dignissim ac semper in, ornare ac mauris. Duis nec
felis mauris.';
        for) $i = ١ ; $i => ٢٠ ; $i++ (
        }
            $post = new Post;
            $post<-title = "Post no $i;"
            $post<-read_more = substr$)content, ,٠ ;(١٢٠
            $post<-content = $content;
            $post<-save;()

            $maxComments = mt_rand;(٣,١٥)
            for) $j = ١ ; $j => $maxComments; $j(++
            }
                $comment = new Comment;
                $comment<-commenter = 'xyz;'
                $comment<-comment = substr$)content, ,٠ ;(١٢٠
                $comment<-email = 'xyz@xmail.com;'
                $comment<-approved = ;١
                $post<-comments<-()save$)comment;(
                $post<-increment')comment_count;('
            {
        {
    {
{
```

for اول برای بوجود آوردن پست های مختلف بدونه نیاز به تنظیمات خاصیه . و حلقه داخلی هم مربوط به ایجاد کامنت برای اون پست در حال ساخت میشه . که کامنت های مختلفی رو برای پستها قرار میده . همچنین در حلقه داخلی بعد از به وجود امدن کامنت فیلد comment_count مربوط به پست اون کامنت رو یه واحد افزایش میده .



laravel با استفاده از این دستور یه ارتباط تعاملی آسان رو از طریق خط فرمان ترمینال با پروژه ایجاد میکنه که شما میتونید

دستورات مختلف و جالبی رو تو خط فرمانتون استفاده کنید که اجازه بدید چند تا از این دستورات رو براتون مثال بزنم :

```
┌]usm4n@usm4n-desktop[─]~[
└─•artisan tinker
<
```

پیدا کردن یه آیدی با استفاده از find()

```
<$post = Post::find;(۲)
<$post<-setHidden'])content','read_more','updated_at;(['
<echo $post;
{"id":","۲"title":"Post no 2","comment_count":","۷"created_at": ۰۶-۰۱-۲۰۱۴"
"۰۹:۴۳:۴۴}
```

محدود کردن بازیابی رکورد ها با استفاده از take() وskip()

```
<$post = Post::skip<-(۵)take<-(۲)get;()
<foreach)$post as $value( echo "post id:$value<-id;"
post id:۶ post id:۷
```

استفاده از select()وfirst()

```
<$post = Post::select')id','title<-('first;()
<echo $post;
{"id":","۱"title":"Post no 1"}
```

استفاده از where() با select()

```
<$post = Post::select')id','title<-('where')id<-(۱۰,'=','first;()
<echo $post;
{"id":","۱۰"title":"Post no 10"}
```

گرفتن خصوصیات یه فیلد کامنت از یه پست

```
<$post = Post::find;(۴)
<echo $post<-comments<-[۰]commenter;
xyz
```

خوب از شما ممنونم که تا اینجای این آموزش با ما بودید اگه نظر یا پیشنهاد برای بهتر شدن این سری آموزش دارید حتما با ما تو بخش

نظرات در میون بزارید .

- **ساخت کنترل ها**

در لاراول ما کنترل ها رو با گسترش کلاس Controller که در دایرکتوری (app/http/controllers) قرار داره ایجاد میکنیم . همه کنترل های بوجود اومده در (app/http/controllers) قرار میگیرن .

نکته : لاراول هیچ محدودیتی در ساختار دایرکتوری خودش نداره و ، توسعه دهنده های میتونن به هر صورتی که بخوان دایرکتوری لاراول رو تنظیم مجدد کنن .

یک مثال از یه کنترل ساده :

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

use ...
//file: app/Http/controllers/IndexController.php

class IndexController extends Controller }

    public function showIndex()
    }
            //generates response from index.blade.php
        return view')index;('
    {
{

//file: app/Http/routes.php
//registering route to controller actions

get')index','IndexController@showIndex;('

//In general
get')route.name','SomeController@someAction;('
post')route.name','SomeController@someAction;('
```

بنابراین هر action در داخل کنترل ها دارای حدافل یه مسیر در فایل app/http/routes.php هستن . همچنین در بعضی مواقع ما با قرار دادن پیشوند به اول هر action بصورت ساده مسیر دلخواهو میسازیم . به کد زیر توجه کنید

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

use ...

class IndexController extends Controller }

    public function getAction()
    }
       // get request handling
    {
    public function postAction()
    }
            //post request handling
    {
{

//registering route
Route::controller')index,''IndexController;(
```

به کد بالا نگاه کنید ما برای ثبت مسیر برای هر عمل در کنترل یه پیشوند برای هر عمل قرار دادیم و با استفاده از Route::controller('index','IndexController') بطور خودکار مسیر دهی ها رو با توجه به اون پیشوند ها خود برنامه انجام میده .

چند نکته :

- بطور معمول هر action با یه view همراه که این view کار نمایش دادن اطلاعات به کاربر رو انجام میده.
- هنگامی که ما یه آرایه رو با استفاده از return برمیگردونیم . این آرایه بطور خودکار با JSON کد گذاری و به ما نشون داده میشه .
- ما با استفاده از متد nest() متونیم یه view رو به یه متغیر نسبت بدیم .

- **ساخت Controller ها برای برنامه**

ما در قدم اول برای شروع کنترل BlogController رو میسازیم که وظیفه رسیدگی به درخواست اول رو داره : نشون دادن صفحه اصلی / محتوایی کلی وبسایت ، قسمت سرچ و نمایش سرچ رو به عهده داره . در قدم بعد ما به ترتیب مسئولیت رسیدگی به عملیات CRUD پست ها و نظرات رو به PostsController و CommentsController میدیم .

BlogController

کد زیر متعلق به کنترل BlogController که در قسمت app/http/Controllers/BlogController.php قرار داره .

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

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

class BlogController extends Controller
}

    **/
*   select 10 post
@*    return Home page
/*
    public function getIndex()
    }
        $posts = post::orderBy')id,' 'desc<-('paginate;(١٠)
        return view')home<-('nest')content,' 'index,' compact')posts','title((('
            <-with')title','Home Page | Laravel 5 Blog;('

    {

    **/
@*    param request serch
@*    return result serch
```

```
/*
    public function getSearch)Request $request(
    }
        $searchTerm = $request']s;['

        $posts = Post::whereRaw')match(title,content) against(? in
boolean mode,'( $]searchTerm([
            <-paginate;(۱۰)
        $posts<-appends'])s' <= $searchTerm;([
        return view')home('
            <-nest')content,' 'index,' $)posts<-isEmpty(() ? ']notFound' <=
true[ : compact')posts((
            <-with')title,' 'Search': . $searchTerm;;(
    {

{
```

زمانی که ما مسیر خودمونو با Route::controller() به صورت Route::controller('/','BlogController') ثبت کنیم . getIndex() از مسیر نقشه برداری میکنه . بعد از اینکه آدرس صفحه اصلی سایت رو اجرا کردیم این action عمل میکنه و محتوایی مورد نظر رو بهتون نشون میده . در داخل getIndex() ما با استفاده از مدل post پست های خودمونو بازیابی میکنم . حالا این کار با استفاده از orderBy('id', 'desc') که باعث مرتبط شدن پست ها براساس شماره id و همینطور برای اینکه تمام مطالب تو یه صفحه نباشه از متد paginate() استفاده میکنیم و عدد ۱۰ رو براش در نظر میگیریم تا در هر صفحه فقط ۱۰ تا پست نمایش داده بشه . بعد از بازیابی اطلاعات اونو تو متغییر $posts قرار میدیم تا تو بخش view بتونیم با استفاده از یه حلقه به راحتی محتوای هر پست رو جدا کنیم و نمایش بدیم حالا قسمت view رو در آینده توضیح میدم . خوب ما با استفاده ازreturn و متد view() صفحه Home.blade.php رو بازیابی و با متد nest() اطلاعات مورد نظرو براش میفرستیم .

ما با استفاده از getSearch() قابلیت جستجو در سایت رو برای کاربرا به وجود میاریم . ما از مدل Post برای ارسال کوئری برای جستجو استفاده میکنیم و همچنین از متد whereRaw() برای پیدا کردن محتوای مورد نظرمون بر اساس شرایط نوع جستجومتنی که مد نظرمون هست استفاده میکنیم و در آخر هم برای صفحه بندی نتایج جستجو از متد paginate() بهره میبریم . بقیه کد ها هم که مشخصه برای ارسال و نمایش اطلاعات بازیابی شده استفاده میشه .

The PostsController

کد زیر مربوط به کنترل PostsController می باشد :

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

use App\Http\Requests;
use App\Http\Requests\PostRequest;
use App\Post;

class PostController extends Controller }

    */get functions/*
    public function listPost()
```

```
    }
        $posts = post::orderBy')id,' 'desc<-('paginate;(١٠)
        return view')dash<-('nest')content,' 'posts.list,' compact')posts((''
        <-with')title','Post listings;('
    {

    **/
@ *   param post $post
@ *   return $this
/*
    public function showPost)post $post(
    }

        $comments = $post<-comments<-()where')approved,' ,'=' <-('get;()
        return view')home<-('nest')content,' 'posts.single,' compact')post,''
'comments((''
            <-with')title$,'post<-title;(
    {

    public function newPost()
    }
        return view')dash<-('nest')content,' 'posts.new('
            <-with')title,' 'New Post;('
    {

    public function editPost)post $post(
    }
        return view')dash<-('nest')content,' 'posts.edit,' compact')post((''
            <-with')title,' 'Edit Post;('
    {

    public function deletePost)post $post(
    }
        $post<-delete;()
        return redirect<-()route')post.list<-('with')success,' 'Post is
deleted;('!
    {


    **/
@ *   param PostRequest $request
 *   post functions
@ *   return \Illuminate\Http\RedirectResponse
/*
    public function savePost)PostRequest $request(
    }
        $post = ]
            'title' <= $request']title,['
            'content' <= $request']content,['
        ;[
            $post = new Post$)post;(
            $post<-comment_count = ;٠
            $post<-read_more = )strlen$)post<-content( < (١٢٠ ?
substr$)post<-content, ,٠ (١٢٠ : $post<-content;
            $post<-save;()
```

```
            return redirect')admin/dash-board<-('with')success,' 'Post is
saved;('!

    {

    public function updatePost)post $post,PostRequest $request(
    }
        $data = ]
            'title' <= $request']title,['
            'content' <= $request']content,['
        ;[

        $post<-title = $data']title;['
        $post<-content = $data']content;['
        $post<-read_more = )strlen$)post<-content( < (۱۲۰ ? substr$)post-
<content, ,۰ (۱۲۰ : $post<-content;
        if )count$)post<-getDirty(() < (۰ */avoiding resubmission of
same content/* }
            $post<-save;()
            return redirect<-()back<-()with')success,' 'Post is updated;('!
        { else
            return redirect<-()back<-()with')success,' 'Nothing to
update;('!
    {


{
```

نکته : اگه به کد های بالا نگاه کرده باشین تو بعضی از این عمل ها post $post رو به عنوان پارامتر میگیره ! خب این کار برای چیه !؟ خوشبختانه لاراول یه خوبی داره اینکه شما می تونید پارامتر های که برای Route ها مشخص میکنین و احتاج به اتصال به model دارن رو به راحتی به هم متصل کنید و در view بطور مستقیم ازش استفاده کنید . برای مثال ما از $router- >model('post','App\Post') که تو فایل RouteServiceProvider.php (این فایل در app/Providers) ثبت میشه استفاده میکنیم . به این کار **Route Model Bindings** که ایشالله در موردش بعدا صحبت میکنیم فعلا به کد زیر دقت کنید:

```
?>php namespace App\Providers;

use Illuminate\Routing\Router;
use Illuminate\Foundation\Support\Providers\RouteServiceProvider as
ServiceProvider;

class RouteServiceProvider extends ServiceProvider }

    **/
     * This namespace is applied to the controller routes in your
routes file.
     *
     * In addition, it is set as the URL generator's root namespace.
     *
     @ * var string
     /*
    protected $namespace = 'App\Http\Controllers;'
```

```
**/
 * Define your route model bindings, pattern filters, etc.
*
@ * param\ Illuminate\Routing\Router$ router
@ * return void
/*
public function boot)Router $router(
}
        parent::boot$)router;(

    $router<-model')post','App\Post;('
```

حالا زمانی که ما یه مسیر رو بازدید کنیم که پارامتری داره و اون پارامتر باید با مدلش ارتباط بر قرار کنه فقط با گذاشتن post $ در action به عنوان پارامتر و استفاده از اون در view کارمونو خیلی ساده و راحت میکنیم .

تنها action عمومی که در این کنترل قرار داره showPost که ما با استفاده از اون برای نمایش یه پست تنها استفاده میکنیم . در داخل showPost() ما با استفاده از Route Model Bindings که در بالا گفتیم پست مورد نظرمونو بازیابی میکنیم و با استفاده از $post->comments() و متد where نظرات مرتبط به اون پست که توسط مدیر تایید شدن رو هم بازیابی میکنیم خب اگه براتون سوال شد که چطوری این دوتا جدول با به هم اتصال پیدا کردن باید بگم که با استفاده از Relationships که تو قسمت اول کد اتصال دو جدول رو در هر مدل نوشتیم . البته Relationships رو تو یه پست جدا بصورت کامل توضیح میدم اما فعلا دونستن چطوری ارتباط این دو تا جدول کافیه براتون و در آخرم اطلاعات رو به صفحه (app/views/posts/single.blade.php) میفرسیتم .

action های باقی مونده در این کنترل برای انجام عملیات CRUD در بخش مدیریت هستن . برای مثال ما از listPost برای نشون دادن لیست پست های موجود در سایت بصورت یه جدول استفاده میکنیم ک همین جدول خودش شامل مسیر های برای حذف و آپدیت میشه .

ما از عمل newPost برای نشون دادن یه فرم به ادمین استفاده میکنیم تا ادمین بتونه با پرکردن و ارسال اون یه پست جدید بوجود بیاره . در داخل اون فرم برای ارسال مسیر savePost قرار داره که کار ذخیره کردن اون پست ارسالی از طرف ادمین رو انجام میده البته بعد از ارسال فرم در savePost که دارای پارامتر PostRequest اعتبار سنجی و اگه مشکلی نداشت در دیتابیس ثبت میشه .

نکته : Request به تازگی در لاراول ۵ معرفی شدن و کارایی و سرعت عمل توسعه دهنده رو خیلی بالا میبره ،البته بعدا بصورت کامل به توضیح این ویژگی جدید لاراول ۵ می پردازیم .

CommentsController

کد مربوط به کنترل CommentsController :

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

use App\Http\Requests;
use App\Http\Controllers\Controller;
use App\Comment;
use App\Post;
```

```
use App\Http\Requests\CommentRequest;
use Illuminate\Http\Request;

class CommentController extends Controller }


     */get functions/*
    public function listComment()
    }
        $comments = Comment::orderBy')id,' 'desc<-('paginate;(٢٠)
        return view')dash<-('nest')content,' 'comments.list,'
compact')comments((''
        <-with')title','Comment Listings;('
    {

    public function newComment)Post $post,CommentRequest $request (
    }

        Comment::create$)request<-all;(()
         */redirect back to the form portion of the page/*
        return redirect<-()back()
            <-with')success,' 'Comment has been submitted and waiting for
approval;('!
    {

    public function showComment)Comment $comment,Request $request(
    }
        if $)request<-ajax(()
            return view')comments.show,' compact')comment;((''
       // handle non-ajax calls here
       // else{}
    {

    public function deleteComment)Comment $comment(
    }
        $post = $comment<-post;
        $status = $comment<-approved;
        $comment<-delete;()
        $)status === 'yes(' ? $post<-decrement')comment_count(' : ;"
        return redirect<-()back<-()with')success,' 'Comment deleted;('!
    {

     */post functions/*

    public function updateComment)Comment $comment,Request $request(
    }

        $comment<-approved = $request']status;['
        $comment<-save;()

        $comment<-post<-comment_count = Comment::where')post_id,' ,'='
$comment<-post<-id(
            <-where')approved,' ,'=' <-(`count;()
        $comment<-post<-save;()
```

```
        return redirect<-()back<-()with')success,' 'Comment'  . $))comment-
<approved === 'yes(' ? 'Approved' : 'Disapproved;(('
    {

{
```

در کد بالا : listComment برای نمایش لیست کامنت ها به مدیر استفاده و شامل یه جدوله که توش لینک اعمال CRUD برای هر کامت وجود داره . زمانی که یه دیدگاه جدید برای یه پست که تو وبسایت قرار داره بخواد ثبت بشه از عمل newComment برای اعتبار سنجی محتوایی اون کامنت و ثبت اون کامنت در دیتابیس استفاده میشه . newComment بعد از اعتبار سنجی و زمانی که در حال ثبت تو دیتابیسه ما approved برابر با 'No' قرار میدیم تا قبل تایید به نمایش در نیاین . بعد مدیر میتونه در پنل مدیریت با استفاده از updateComment اون کامنت رو تایید کنه البته updateComment بعد از هر تغییر تعداد کامنت های تایید شده برای اون پست رو میشماره و تو جدول پست مربوط به همون نظر تو ستون comment_count ثبت میکنه . deleteComment هم برای حذف کامنت مورد استفاده قرار میگیره .

در آخر عمل showComment که کارش نشون دادن یه دیده کلی از مشخصات ارسال کننده کامنت و محتوایی کامنته که بصورت آژاکس این درخواست ارسال و نمایش داده میشه . که ما این رو تو قسمت های بعد توضیح میدیم .

ممنون از این که تا اینجایی این قسمت هم همراه ما بودید . در بخش بعد ما در مورد routing ها صحبت میکنیمو میگیم برای این پروژه به چه routing نیازه .

- Routing در لاراول

مسیریابی ، نقش اساسی در عملکرد هسته هر فریم ورک MVC ایفا میکنه . در حقیقت Route یه نگاشت بین موتور requests و response اگه بخوایم خیلی ساده تعریفش کنیم آدرس های URL ی که شما در مرورگرتون وارد میکنید با route مدیریت میشه و نقاط ورودی برنامه اتونو تعریف میکنه . داشتن یه routing انعطاف پذیر می تونه به شما کمک کنه تا کنترل بهتر و کاربردی تر رو برنامه اتون داشته باشین .

لاراول دارای یک مسیریابی قدرتمنده ، که بر اساس مسیریابی Symfony نوشته شده که کار باهاش خیلی خیلی راحته همینطور رابط کاربری راحت و قابلیت های زیادی داره که برنامه نویسی با اونو براتون لذت بخش تر میکنه .

- تعریف Route ها

در لاراول Route های برنامه ما در مسیر app\http\routes.php تعریف میشن . یک مثال ساده از فایل routes.php :

```
?>php

//will be used to handle GET requests.
Route::get')index,'function()
{
    echo 'this is index page;'
;({

Route::get')login,'function()
{
    echo 'GET login requests will be hndled here;'.
;({

//will be used to handle POST requests.
Route::post')login,' function()
{
    echo 'POST login requests will be handled here;'.
;({
```

ما همچنین میتونیم از متد کنترل ها استفاده کنیم مثل زیر :

```
?>php
Route::get')users,' 'UsersController@getIndex;('
```

در کد بالا زمانی که ما درخواستی بصورت /users داشته باشیم این درخواست بوسیله route به متد getIndex از کنترل UserController متصل میشه و داده ها رو به نسبت کدی که در getIndex قرار داره به کاربر نشون میده ما همچنین میتونیم داده ها رو بطور مستقیم بوسیله افعال خود http حذف یا اضافه کنیم با استفاده از Route::put و Route::delete

پارامتر ها در Route

```
?>php

//parameter {id} will be passed to the closure.
Route::any')post/{id,'{function$)id(
{
    echo "post with id: $id;"
;({
```

```
//A model with given post id will be passed to closure for any HTTP
request.
Route::any')post/{post,'{function$)post(
}
    echo "post with id: $post<-id;"
;({
```

نکته : filter که در نسخه ۴ لاراول در route ها مورد استفاده قرار میگرفت در نسخه ۵ جایی خودشو به Middleware داده

که تو آموزش های بعد بطور کامل اونو توضیح میدم .

ما می تونیم در هر route با تعریف کلید در آرایه ای که در زیر میبینید یک نام روتر دوم داشته باشیم .

```
?>php

Route::get')admin'],'as'<='admin.home','middleware' <= 'auth,'function()
}
    return 'is already called;'
;([{

//another example using controller action.
Route::get/')post/list'],'as' <= 'post.list','uses' <=
'PostController@listPost;(['
```

در view ها هم ما میتویم با استفاده از route() مسیرهای موجود رو به لینک قابل کلیک تبدیل کنیم بصورت زیر

```
?>php
route')post.list;('
```

شما میتویند برای توضیحات کامل تر در مورد route ها به اسناد خود لاراول مراجعه کنید یا منتظر بمونید تا ما بطور کامل در پست

های دیگه route ها رو توضیح بدیم . البته شما میتونید از فیلم ویدئویی که در این مورد در سایت وجود داره هم استفاده کنید .

- ساخت Route های مورد نیاز برای وبلاگ

در زیر مسیر ههای که برای برنامه اومون استفاده میکنیم رو میتونید ببینید :

```
?>php
//file: app/http/routes.php
 Route::controllers])
   'auth' <= 'Auth\AuthController,'
   'password' <= 'Auth\PasswordController,'
;([

 */User routes/*
get/')post/{post}/show,' ']as' <= 'post.show,' 'uses' <=
'PostController@showPost;(['
post/')post/{post}/comment,' ']as' <= 'comment.new,' 'uses' <=
'CommentController@newComment;(['

 */Admin routes/*
Route::group'])prefix' <= 'admin,' 'middleware' <= 'auth,[' function () }
    */get routes/*
    get')dash-board,' function () }
        $username = Auth::user<-()name;
        return view')dash<-('with')content,' "Hi $username, Welcome to
Dashboard("!
            <-withUsername')username$,'username(
            <-withTitle')title','DashBoard;('
```

```
    ;({
    get/')post/list,' ']as' <= 'post.list,' 'uses' <=
'PostController@listPost;(['
    get/')post/new,' ']as' <= 'post.new,' 'uses' <= 'PostController@newPost;(['
    get/')post/{post}/edit,' ']as' <= 'post.edit,' 'uses' <=
'PostController@editPost;(['
    get/')post/{post}/delete,' ']as' <= 'post.delete,' 'uses' <=
'PostController@deletePost;(['
    get/')comment/list,' ']as' <= 'comment.list,' 'uses' <=
'CommentController@listComment;(['
    get/')comment/{comment}/show,' ']as' <= 'comment.show,' 'uses' <=
'CommentController@showComment;(['
    get/')comment/{comment}/delete,' ']as' <= 'comment.delete,' 'uses' <=
'CommentController@deleteComment;(['

    */post routes/*
    post/')post/save,' ']as' <= 'post.save,' 'uses' <=
'PostController@savePost;(['
    post/')post/{post}/update,' ']as' <= 'post.update,' 'uses' <=
'PostController@updatePost;(['
    post/')comment/{comment}/update,' ']as' <= 'comment.update,' 'uses' <=
'CommentController@updateComment;(['

;({

 */Home routes/*
Route::controller,'/') 'BlogController;('

 */View Composer/*
View::composer')sidebar,' function $)view( {
    $view<-recentPosts = App\post::orderBy')id,' 'desc<-('take<-(۵)get;()
;({
```

در کد بالا برای اعتبار سنجی مدیریت ما از کنترل auth استفاده میکنیم و همینطور در route های بعدی یعنی post.show و comment.new ما به مخاطب ها اجازه میدیم تا پست ها رو در صفحات تکی (single page) ببینن و اگه خواستن نظر خودشون در مورد اون پست رو ارسال کنن . در route بعد ما یه route:group میسازیم که تمام مسیر های مربوط به مدیریت در اون قرار میگیره و یک پیشوند admin هم برای route های که در این گروه قرار دارن تعریف میکنیم .تا هر route که درخواست شد قبلش admin بیاد برای مثال زمانی که مسیر post.edit فراخوانی بشه به url به این صورت بازگردانی میشه (http://localhost/admin/post/12/edit)

ایجاد فرم تماس با ما با لاراول

چند نکته

ابتدا در فایل app.php در مسیر app/config مقدار debug برابر true قرار دهید، تا اگر خطایی بوجود آمد بتوانید آن را به سادگی تشخیص دهید.

ممکن است مطالبی که در این مطلب گفته می‌شود، در سری آموزش لاراول به آن اشاره‌ای نکرده باشیم و جدید باشد.

```
Route::get('contact-us', array('as' => 'get.contact-us', function() {

 return View::make('contact-us');

}));
```

در کد بالا یک مسیر را مشخص کردیم (خط یک) و زمانی که کاربر این مسیر را باز کند (به صورت(get ، فایل) contact-us خط دو) را به خروجی میفرستیم. همچنین به مسیر مشخص شده یک نام تحت عنوان) get.contact-us خط یک) دادیم که برای آدرس دهی به این مسیر کار ما را در ادامه راحتتر میکند.

در همان فایل routes.php مسیر زیر را نیز ایجاد میکنیم که در واقع post مسیر بالا است و آن را با نام post.contact-us نام گذاری میکنیم اما فعلاً درون آن کاری انجام نمیدهیم.

```
Route::post('contact-us', array('as' => 'post.contact-us', function() {


}));
```

حالا اگر به مسیر /contact-us برویم، با خطا مواجه می‌شویم، زیرا ما هنوز فایلی را که به خروجی فرستادیم ایجاد نکردیم. برای اینکار یک فایل با نام contact-us.blade.php در مسیر app/views میسازیم، سپس درون این فایل ساختار HTML را ایجاد کرده و درون تگ body کدهای زیر را قرار میدهیم.

```
{{ Form::open(array('route' => 'post.contact-us')) }}
 <p>
  {{ Form::label('name', 'نام:') }}
  {{ Form::text('name') }}
 </p>
 <p>
  {{ Form::label('email', 'ایمیل') }}
  {{ Form::email('email') }}
 </p>
 <p>
  {{ Form::label('context', 'متن:', array('style' => 'float:left;')) }}
  {{ Form::textarea('context') }}
 </p>
 <p>
  {{ Form::submit('ارسال') }}
 </p>
{{ Form::close() }}
```

کد بالا در واقع فرم ما است که با استفاده از کدهای لاراول نوشته شده، میتوان هر قسمت از فرم را به صورت HTML معمولی نیز نوشت.

در خط اول، تگ فرم را ایجاد کرده و action آن را برابر مسیر post.contact-us قرار دادیم. در خطهای بعدی نیز ساختار فرم را قرار میدهیم.

حالا شما باید فرم را بدون خطا در مرورگر مشاهده کنید و همچنین زمانی که فرم را پر کنید و ارسال را بزنید، فقط یک صفحه سفید مشاهده کنید، زیرا ما هنوز در مسیر post.contact-us هیچکاری انجام ندادیم.

اعتبارسنجی

خب اولین کاری که باید بر روی اطلاعاتی که از طرف کاربر میرسد انجام دهیم چیست؟ مطمئناً اعتبارسنجی یا Validation.

اما قبل از اینکه وارد اعتبارسنجی اطلاعات شویم، ابتدا بهتر است تگهایی را برای نمایش خطاهایی که کاربر ایجاد میکند درون فایل HTML ایجاد کنیم. برای اینکار Laravel یک شیء به نام $errors دارد که متن خطاهایی که باید به کاربر نمایش دهیم را درون خود نگه میدارد.

```
{{ $errors -> first('name', '<span class="error">:message</span>') }}
```

برای مثال در کد بالا، در صورتی که کاربر در فیلدی با نام name خطایی در ورورد اطلاعات داشته باشد، متن خطای ایجاد شده درون تگ span با کلاس error نمایش داده میشود. پس باید برای هر کدام از فیلدهایمان یک خط کد مانند بالا ایجاد کنیم. پس کدهای HTML به صورت زیر تغییر میکند.

```
{{ Form::open(array('route' => 'post.contact-us')) }}
 <p>
  {{ Form::label('name', 'نام:') }}
  {{ Form::text('name') }}
  {{ $errors -> first('name', '<span class="error">:message</span>') }}
 </p>
 <p>
  {{ Form::label('email', 'ایمیل') }}
  {{ Form::email('email') }}
  {{ $errors -> first('email', '<span class="error">:message</span>') }}
 </p>
 <p>
  {{ Form::label('context', 'متن:', array('style' => 'float:left;')) }}
  {{ Form::textarea('context') }}
  {{ $errors -> first('context', '<span class="error">:message</span>') }}
 </p>
 <p>
  {{ Form::submit('ارسال') }}
 </p>
{{ Form::close() }}
```

حالا به فایل routes.php می‌رویم و مسیر contact-us قسمت post را در نظر میگیریم. ابتدا برای اعتبارسنجی داده‌ها باید قوانینی را تعیین کنیم که این قوانین باید به صورت یک آرایه باشد. مانند کد زیر:

```
$rules = array(
 'name' => 'alpha|max:10',
 'email' => 'required|email',
 'context' => 'required'
);
```

در کد بالا در هر خط یکی از فیلدهایمان را آوردیم و روبروی آن قوانینی را برای آن فیلد تعیین کردیم. هر قانون با استفاده از علامت | از یکدیگر جدا میشوند.

برای فیلد name اولین قانونی که آوردیم alpha است. این قانون مشخص میکند که داده‌های کاربر در فیلد name فقط از کاراکترهای الفبایی باشد و در صورتی که کاربر برای مثال در این فیلد عدد وارد کند با خطا مواجه می‌شود. دومین قانون max است که مقدار جلوی آن ۱۰ است که مشخص میکند حداکثر تعداد حروفی که کاربر میتواند وارد کند ۱۰ است.

برای فیلد email اولین قانون required است، به این معنی که این فیلد باید حتماً توسط کاربر پر شود، در حالی که فیلد قبلی اختیاری بود. دومین قانون email است که مشخص میکند این فیلد باید حتماً یک آدرس ایمیل باشد.

برای فیلد context فقط یک قانون تعریف کردیم و آنهم اینکه کاربر حتماً باید آن را وارد کند.

به این صورت میتوان قوانین مختلفی را ایجاد کرد. لیست تمام قوانین در این صفحه موجود میباشد.

حالا که قوانین تعیین شدند اعتبارسنجی را انجام میدهیم.

```
$validator = Validator::make(Input::all(), $rules);
```

در کد بالا متد make از namespace یا فضای نام Validator را فراخوانی کردیم. این متد دو پارامتر دریافت میکند. اولین پارامتر داده‌های کاربر است که ما تمام اطلاعات وارد شده توسط کاربر را با استفاده از Input::all() به این متد میفرستیم. دومین پارامتر قوانینی است که میخواهیم روی آن‌ها بررسی و اعمال شود که آن را هم به این متد میدهیم که نتیجه را درون متغیر $validator میریزد.

در انتها فقط کافیست با یک شرط بررسی کنیم که آیا قوانین ما را کاربر رعایت کرده است یا خیر، که اگر رعایت نکرده بود دوباره به صفحه قبل بازگردد و مجدداً اطلاعات را وارد کند.

```
if($validator -> fails()) {

 return Redirect::back() -> withErrors($validator);

} else {

 return 'true';

}
```

در کد بالا میبنید که در صورتی که کاربر قوانین را رعایت نکرده باشد به صفحه قبل بازمیگردد همچنین متن خطاهای ایجاد شده را نیز به آن صفحه میفرستیم تا کاربر از خطاهای ایجاد شده مطلع شود.

اما اگر کاربر هیچ خطایی در ورود اطلاعات نداشته باشد مقدار true را فعلاً چاپ میکنیم.

کار اعتبارسنجی داده‌ها به پایان رسید. حالا شما باید آن را با استفاده از داده‌های اشتباه تست کنید.

در پایان فایل routes.php باید دارای کدهای زیر باشد.

```
Route::get('contact-us', array('as' => 'get.contact-us', function() {
 return View::make('contact-us');
}));


Route::post('contact-us', array('as' => 'post.contact-us', function() {
 $rules = array(
  'name' => 'alpha|max:10',
  'email' => 'required|email',
  'context' => 'required'
 );

 $validator = Validator::make(Input::all(), $rules);
 if($validator -> fails()) {
  return Redirect::back() -> withErrors($validator);
 } else {
  return 'true';
 }
}));
```

محلی کردن لاراول

اگر تا اینجا برنامه را تست کرده باشید متوجه خواهید شد که متن خطاهایی که به کاربر نمایش داده می‌شود انگلیسی است. خوشبختانه لاراول در این قسمت هم قوی عمل کرده است. برای اینکه خطاها به زبانی که میخواهید نمایش داده شود ابتدا به مسیر app/lang بروید و از پوشه en یک کپی گرفته و در همان مکان با نام دیگری (در اینجا با نام fa) ذخیره کنید.

سپس به فایل app.php در مسیر app/config رفته و قسمت locale را برابر نام همان پوشه جدید (در اینجا fa) قرار دهید.

اگر پوشه fa را مشاهده کنید خواهید دید که حاوی سه فایل است. اما فایلی که ما فعلاً به آن احتیاج داریم validation.php است. تمام پیغام های خطا درون این فایل وجود دارد.

شما باید این فایل را فارسی کنید. نترسید. لازم نیست به یکباره تمام آن را به فارسی تبدیل کنید، فقط کافیست در طول برنامه نویسی برنامه هر کجا که پیغامی انگلیسی نمایش داده شد، همان پیغام را به فارسی تبدیل کنید.

قسمت‌هایی که ما در این برنامه استفاده کردیم در زیر آمده است فقط کافیست آنها را پیدا کرده و جایگزین کنید.

```
"alpha"          => "فیلد :attribute فقط باید حاوی حروف باشد.",

"email"          => "این فیلد فقط باید حاوی ایمیل باشد.",

"required"       => "پر کردن فیلد :attribute ضروری است.",

"max"            => array(
 "numeric" => "The :attribute may not be greater than :max.",
 "file"    => "The :attribute may not be greater than :max kilobytes.",
 "string"  => "فیلد :attribute نباید بیشتر از :max حرف باشد.",
 "array"   => "The :attribute may not have more than :max items.",
),
```

همچنین نام فارسی شده فیلدها را نیز باید وارد بکنیم. بنابراین آخرین آرایه موجود در این فایل یعنی آرایه attributes باید مشابه زیر شود.

```
'attributes' => array(
 'name' => 'نام',
 'email' => 'ایمیل',
 'context' => 'متن'
),
```

برگشت داده‌ها

اگر به صورت دقیق تست کرده باشید متوجه می‌شوید که زمانی که کاربر داده‌ها را اشتباه وارد میکند و به صفحه اول برگشت داده می‌شود، تمام فیلد ها خالی هستند. برای اینکه فیلدها مقادیر گذشته خودشان را حفظ کنند باید، ابتدا، در زمان برگشت صفحه، اطلاعات فیلدها را نیز برگشت دهیم.

```
return Redirect::back() -> withErrors($validator) -> withInput();
```

سپس خاصیت value فیلدها را برابر Input::old('name') قرار دهیم که به جای name باید نام همان فیلد گذاشته شود مانند زیر:

```
{{ Form::text('name', Input::old('name')) }}

{{ Form::email('email', Input::old('name')) }}

{{ Form::textarea('context', Input::old('name')) }}
```

نتیجه‌گیری

در این بخش ما با Routing و Validation آشنا شدیم و کار کردیم. تمام بخش‌هایی که در این مطلب به آن اشاره کردیم را میتوان از راه‌های دیگری نیز انجام داد که بسته به سلیقه برنامه نویس متفاوت است.

در بخش بعدی اطلاعات را درون پایگاه داده ذخیره و همچنین یک ایمیل به کاربر میفرستیم.

تنظیمات پایگاه داده

در ابتدا یک دیتابیس با نام laravel_test میسازیم. سپس توسط کوئری زیر یک جدول با نام contact_us میسازیم (این کوئری را میتوانید در بخش SQL از phpMyAdmin اجرا کنید)

```
CREATE TABLE IF NOT EXISTS <code>contact_us</code> (
 <code>id</code> int(11) NOT NULL AUTO_INCREMENT,
 <code>name</code> varchar(10) COLLATE utf8_persian_ci NOT NULL,
 <code>email</code> varchar(100) COLLATE utf8_persian_ci NOT NULL,
 <code>context</code> text COLLATE utf8_persian_ci NOT NULL,
 PRIMARY KEY (<code>id</code>)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_persian_ci AUTO_INCREMENT=5 ;
```

حالا که پایگاه داده و جدول موردنظر ساخته شد به فایل database.php در مسیر app/config میرویم.
در این فایل در آرایه mysql که درون آرایه connections است اطلاعات دیتابیس و نحوه ورود را مشخص میکنیم.

```
'mysql' => array(
   'driver'    => 'mysql',
   'host'      => 'localhost',
   'database'  => 'laravel_test',
   'username'  => 'root',
   'password'  => '',
   'charset'   => 'utf8',
   'collation' => 'utf8_unicode_ci',
   'prefix'    => '',
),
```

اکنون لاراول به دیتابیس دسترسی دارد. در این مطلب برای کار با دیتابیس از Eloquent در لاراول استفاده میکنیم. برای اطلاعات بیشتر در این مورد، به مطلب :Laravel کار با Eloquent مراجعه کنید.

همانطور که میدانید در Eloquent برای هر جدول باید یک فایل درون پوشه app/models بسازیم. به همین دلیل یک فایل با نام Contact.php در این پوشه میسازیم و کدهای زیر را درون آن قرار میدهیم.

```
<?php


class Contact extends Eloquent {


  protected $table = 'contact_us';

  public $timestamps = false;


}

&#91;/php&#93;</div>
```

حالا براحتی میتوانیم برای دسترسی به جدول contact_us از کلاس Contactدر هر جای برنامه استفاده کنیم.

</h2>ذخیره اطلاعات<h2>

در مطلب قبلی تا اینجا برنامه را پیش بردیم که اگر تمام اطلاعات وارد شده توسط کاربر درست بود مقدار true را برمیگرداند (فایل-<span class="en
app).در مسیر words">routes.php

```
<div class="mycode">[php]

$validator = Validator::make(Input::all(), $rules);

if($validator -> fails()) {

  return Redirect::back() -> withErrors($validator) -> withInput();

} else {

  return 'true';

}
```

خطی که مقدار true را برمیگرداند (خط ۵)، پاک میکنیم و از این پس تمام کدهایی که مینویسیم را در این بخش به جای آن قرار میدهیم.

کار با Eloquent بسیار ساده است. توسط کدهای زیر اطلاعات وارد شده توسط کاربر را در جدول contact_usذخیره میکنیم.

```
$contact = new Contact;

$contact -> name = Input::get('name');

$contact -> email = Input::get('email');

$contact -> context = Input::get('context');

$contact -> save();
```

ارسال ایمیل

پس از ذخیره اطلاعات وارد شده، میخواهیم یک ایمیل نیز که حاوی همین اطلاعات است به مدیریت سایت ارسال کنیم.

ابتدا باید تنظیمات ایمیل را درون لاراول درست کنیم. لاراول از روش‌های مختلف ارسال ایمیل مانند سرور SMTP یا sendmail و یا درایور معمول PHP با نام mail پشتیبانی میکند، که بستگی به انتخاب شما دارد. در استفاده های معمولی، استفاده از درایور معمول PHP با نام mail کفایت میکند.

برای تنظیم لاراول در استفاده از درایور mail یا هر گزینه دیگر به فایل mail.php در مسیر app/config میرویم.

در این فایل در گزینه driver نوع درایور را مشخص میکنیم. در صورتی که mail انتخاب شود ضرورتی به تنظیم گزینه های بعدی نیست. ما این گزینه را روی mail میگذاریم.

توجه کنید که در صورتی که در localhost برنامه را اجرا میکنید، امکان ارسال ایمیل وجود ندارد و احتمال دارد با خطای لاراول مواجه شوید.

گزینه ای دیگر در این فایل با نام from وجود دارد. در صورتی که میخواهید تمام آدرس‌ها و نام های ارسال شده از برنامه شما دارای یک آدرس و نام باشد این گزینه را پر کنید.

قبل از اینکه به فایل routes.php برگردیم، ابتدا یک فایل در مسیر app/views/emails با نام contact-us.blade.php میسازیم که این فایل همان محتوای ایمیل ارسالی است. درون این فایل کدهای زیر را قرار میدهیم.

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="UTF-8">
 <title>Email</title>
</head>
<body>
 <h1>تماس با ما</h1>
 <h2>ارسال ایمیل توسط: {{$name}}</h2>
 <h3>ایمیل کاربر: {{$email}}</h3>
 <p>{{$context}}</p>
</body>
</html>
```

حالا به فایل routes.php میرویم. در ادامه کدهایی که نوشتیم کد زیر را اضافه میکنیم.

```
$data = array(
 'name' => Input::get('name'),
 'email' => Input::get('email'),
 'context' => Input::get('context'),
);


Mail::send('emails.contact-us', $data, function($message) {
 $message->from('us@example.com', 'Laravel');


 $message->to('mohsen.sh12@hotmail.com') -> subject('Welcome!');
});
```

در بالا از متد send از کلاس Mail استفاده کردیم. اولین پارامتر آن اسم و مسیر فایلی است که در View ساختیم. دومین پارامتر آن آرایه ای از اطلاعات است که قرار است در فایل View از آن استفاده کنیم. آخرین پارامتر هم یک تابع است که درون آن اطلاعات فرستنده و گیرنده را مشخص میکنیم.

کلاس Mail دارای متدهای مختلفی است که از جمله آن میتوان به صف ها اشاره کرد، برای اطلاعات بیشتر میتوانید به سایت لاراول مراجعه کنید.

در انتها که کار به پایان میرسد، میتوانید به آدرسی دیگری تغییر مسیر بدهید (توجه کنید که مسیر را ابتدا ساخته باشید)

```
return Redirect::to('success');
```

و یا به همان صفحه قبل بازگردید.

```
return Redirect::back();
```


آموزش فارسی
www.AmozeshFarsi.ir

امیدوارم از کتاب استفاده کرده وآن را به رایگان در اختیار دوستانتان قرار دهید.