---Sheme-1. **Laravel-logo**

Разработка **MVC**-**МНОГОПОЛЬЗОВАТЕЛЬСКИХ**-**РАСПРЕДЕЛЕННЫХ** систем в **Laravel**.

---Sheme-2. **Laravel**-**MVC**-**МНОГОПОЛЬЗОВАТЕЛЬСКИХ**-**РАСПРЕДЕЛЕННЫХ**

-----------------------------------------------------------------------------------

1.**MVC**

**Любое** современное **приложение в web** должно быть реализовано на основе концепции **MVC**.

---Sheme-3. **MVC**

Если вы пишите на **native**-языке, вам придется **разработать** и **реализовать** эту концепцию в своем коде **самостоятельно (products - native-php-MVC) + ПРОДУМАТЬ РОУТИНГ**.

Если же вы используете **MVC-ФРЕЙМВОРК**, эта концепция реализовано в нем **АВТОМАТОМ** (ее нужно просто **грамотно использовать**) - **http://laravel-goods-group1/** :

-**routes\web.php**

Route::name('home')->**get('/', 'GoodController@index')**;

-app\Http\**Controllers\GoodController.php**

public function **index**(Request $request)

{

**$goods = $this→repository→funcSelect($request); //ОБРАЩЕНИЕ В МОДЕЛЬ**

// Ajax response

if ($request->ajax()) {

return response()->json([

'table' => view("good.brick-standard", ['goods' => $goods])->render(),

]);

}

**return view('good.index', ['goods' => $goods]); //ВЫВОД ДАННЫХ ВО ВЬЮ**

}

-app\**Repositories\GoodRepository.php** - ПРЕД**МОДЕЛЬ**

public function **funcSelect**($request)

{

$query = $this->model\_product

**->select('id', 'name', 'price', 'image')**; **//ПОСТРОИТЕЛЬ ЗАПРОСОВ Eloquent**

//->where('top9', 1);

//->orderBy('price', 'desc');

if(isset($request->topothers)) $query->where('top9', $request->topothers);

else $query->where('top9', 1);

**return $query->get();**

}

-**resources\views\good\index.blade.php**

<tbody id="pannel">

**@include('good.brick-standard')**

</tbody>

-**resources\views\good\brick-standard.blade.php**

**@foreach($goods as $good)**

<tr>

<td>

**<img class="img\_little" src="{{ asset('public/images/' . $good->image) }}" alt />**

</td>

**<td class="cart\_name">{{ $good->name }}</td>**

**<td class="cart\_price">{{ $good->price }}</td>**

</tr>

**@endforeach**

...**в результате** — **на экране** вы видите …, а **в БД** это …

+НЕМНОГО **ПОДРОБНЕЕ** хотелось бы остановиться на **BLADE-ШАБЛОНИЗАТОРЕ**, который используется в **Laravel** для формирования **ВЬЮ**

---Sheme-4-**Blade**

**layout-@yield-view(js, css)-include**

Кроме **РОУТИНГА** + **ПОСТРОИТЕЛЬ ЗАПРОСОВ Eloquent + Blade-шаблонизатора** Laravel содержит в себе **ЕЩЕ МАССУ** полезных **КЛАССОВ-ШАБЛОНОВ**, которые позволяют разработчику **ПРОСТО ИСПОЛЬЗОВАТЬ** их, **НЕ** разрабатывая каждый раз заново.

Например — **ВАЛИДАЦИЯ** (при **записи/корректировке в БД**).

-app\Http\**Controllers\GoodController.php**

public function **tocart(CartRequest $request)**

{

$this->repository->store($request);

return redirect(route('home'));

}

-app\Http\**Requests\CartRequest.php**

public function **rules**()

{

return **$rules = [**

**'name' => 'bail|required|max:255',**

**'price' => 'bail|required|numeric',**

**'image' => 'bail|required|max:255',**

**];**

}

Или — **МИГРАЦИИ** и **СЕДЕРЫ** для **СОЗДАНИЯ** **mysql**-таблиц и их **НАПОЛНЕНИЯ** начальными данными — **database\migrations и ...\seeds**

...**СЕРВИС-ПРОВАЙДЕРЫ** для, например, **ВЫПАДАЮШЕГО МЕНЮ** и **ДИРЕКТИВ** и т.д....

--------------------------------------------------------------------------------------------

2.**МНОГОПОЛЬЗОВАТЕЛЬСКИХ-РАСПРЕДЕЛЕННЫХ**

Предположим, есть **интернет-магазин**, который продает **авто** - **http://laravel-auto-group1/**

Но, каждый интернет-магазин должен иметь **ДВЕ ЧАСТИ** — **FRONT-КЛИЕНТСКУЮ** (которую видят **ВСЕ** пользователи - **!ЭТО ПОКА НЕ С ТОЧКИ ЗРЕНИЯ КОДА, А ФУНКЦИОНАЛА**) и **BACK-АДМИНЧАСТЬ** (которая нужна для **НАПОЛНЕНИЯ** сайта товарами и, которую может видеть **ТОЛЬКО АДМИНИСТРАТОР**).

Тогда, в таблице **users** для каждого пользователя добавляется поле **role**...

...и далее, используя **ВСТРОЕННЫЕ** возможности **(!БЕЗ написания ОГРОМНОГО ДОПОЛНИТЕЛЬНОГО кода (select, if…)**) (Kernel.php, Middleware) мы можем **ОЧЕНЬ ПРОСТО РАЗДЕЛИТЬ ЛОГИКУ РАЗРЕШЕННЫХ ДЕЙСТВИЙ ПОЛЬЗОВАТЕЛЕЙ В ЗАВИСИМОСТИ ОТ ПРИСВОЕННОЙ ЕМУ(ПОЛЬЗОВАТЕЛЮ) РОЛИ**.

Причем, такое **РАЗДЕЛЕНИЕ** проводится как на **FRONT-ЧАСТИ !УЖЕ КОДА(html)**, так и на **BACK-ЧАСТИ !ТАКЖЕ УЖЕ КОДА(php)**:

-**FRONT(html)** — **resources\views\front\navbar.blade.php**

**@admin**

<a class="dropdown-item" href="{{ route('dashboard') }}">**Dashboard**</a>

**@endadmin**

**@redac**

<a class="dropdown-item" href="{{ route('dashboard') }}">**Dashboard**</a>

**@endredac**

...**НО ТОЛЬКО ЭТОГО МАЛО** — в **url …/dashboard**

-**BACK(php)** — **routes\web.php**

/\*

|--------------------------------------------------------------------------

| **Frontend**

|--------------------------------------------------------------------------|

\*/

// Home

Route::name('home')->**get('/', 'Front\AutoController@index')**;

/\*

|--------------------------------------------------------------------------

| **Backend**

|--------------------------------------------------------------------------|

\*/

Route::prefix('')->namespace('Back')->group(function () {

//Route::middleware('admin')->group(function () {

**Route::middleware('redac')**->group(function () {

Route::name('dashboard')->**get('/dashboard'**, 'AdminController@index');

Route::resource('autos', 'AdminController');

Route::name('upload')->post('/autos/create', 'AdminController@upload');

});

});

...**закомментировать** и под **serg** на **…/dashboard**

Но, и это еще **НЕ ВСЕ** — сейчас на **dashboard** могут попасть **ДВА ВИДА ПОЛЬЗОВАТЕЛЕЙ** — **С РОЛЯМИ admin** и **redac**.

Но, например, **по функционалу** системы у них **РАЗНЫЕ ДОСТУПА К ДЕЙСТВИЯМ** на самой **dashboard**:

-**admin** может делать **ВСЕ (create-store, edit-update, destroy)**;

-**redac** может только **РЕДАКТИРОВАТЬ** записи **(edit-update)**.

...Опять таки, разделение **!ТЕПЕРЬ УЖЕ ПО ДЕЙСТВИЯМ (actions)** делаем на **FRONT-ЧАСТИ (КАСАТЕЛЬНО КОДА — html)** и **BACK-ЧАСТИ(ТАКЖЕ, КАСАТЕЛЬНО КОДА — php)**:

-**FRONT(html)** — resources\views\**back\layout.blade.php**

**@admin**

<li>

<a href="{{ route('autos.create') }}">

<i class="fa fa-folder-o"></i> <span>**Add new Auto**</span>

</a>

</li>

**@endadmin**

+тут же **brick-standard.blade.php**

**@admin**

<td class="center widthbutton"><a class="btn btn-danger listbuttonremove" href="{{ route('autos.**destroy**', [$auto->id]) }}"><i class="fa fa-trash-o" aria-hidden="true"></i></a></td>

**@endadmin**

**...В РЕЗУЛЬТАТЕ ВЫ ВИДИТЕ…, НО ЭТОГО МАЛО**

-**BACK(php)** — app\Http\**Controllers\Back\AdminController.php**

public function \_\_construct(AdminRepository $repository)

{

//$this->middleware('auth');

**$this->middleware('admin')->only('create', 'store', 'destroy');**

$this->repository = $repository;

}

...тогда попытка для **alex(redac)** **…/autos/create**

## **This action is not accessed for you!**

...**закомментировать** и для **alex(redac) …/autos/create**

-------------------------------------------------------

3.**native-php** или **Laravel**?

Предположим, нужно создать систему со следующим функционалом:

-пользователь отправляет с **МОБИЛЬНОГО** сообщение;

-сообщение записывается в **БД CRM** (т. е. - **СЛОЖНОЙ** системы — **МНОГОПОЛЬЗОВАТЕЛЬСКИЙ РАСПРЕДЕЛЕННЫЙ ДОСТУП, АНАЛИТИКА, ОТЧЕТЫ...**)

Тогда, имеет смысл:

-**МОБИЛЬНУЮ** часть написать на **native-php** - **!СДЕЛАТЬ ЕЕ МАКСИМАЛЬНО «ЛЕГКОЙ»** (**mobi-message/**);

-**CRM** — конечно, **фреймворк — Laravel**, например (**laravel-message/**);

-**СВЯЗАТЬ** их по **API**.

+тут же — **РАСПРЕДЕЛЕННАЯ по РОЛЯМ** система — пользователь (**alex**, например), когда он отправил сообщение **С МОБИЛЬНОГО**, может зайти **УЖЕ В CRM, УВИДЕТЬ** его, **ЗАЛОГИНИТЬСЯ и ПОЛУЧИТЬ ПОЛНЫЙ ДОСТУП (УДАЛИТЬ, КОРРЕКТИРОВАТЬ) !НО ТОЛЬКО К СВОИМ СООБЩЕНИЯМ**.