**Auth (Chapter 13)**

**Разграничение доступа:базовые инструменты**

Laravel предоставляет несколько готовых инструментов для выполнения регистрации новых пользователей, входа на сайт, проверки, имеет ли текущий пользователь привилегии на выполнение какой-либо операции, и выхода с сайта, а также для сброса пароля, проверки существования адреса электронной почты и подтверждения пароля.

Это UI, Jetstream, Fortify, Breeze.

https://blog.codecourse.com/laravel-authentication-ui-vs-jetstream-vs-fortify-vs-breeze

Они вынесены в отдельные библиотеки, каждая со своими особенностями и требуют инсталяции в проект. В книге описан UI, но в официальной документации по Laravel 9 его нет, а описан Breeze.

**- Настройки подсистемы разграничения доступа(13.1):**

Настройки хранятся в модуле config\auth.php. Модуль присутствует до установки библиотек. Описание настроек смотри в книге, сокращать там особо нечего, и описание не очень понятное.

**- Установка Nodejs и NPM(не из книги):**

View библиотек разграничения доступа используют frontend frameworks(Vue, Bootstrap...), для работы которых необходим Nodejs и NPM(node package manager) установленные на уровне системы.

https://losst.pro/ustanovka-node-js-ubuntu-18-04

Удаление предустановленной старой версии и установка новой версии Node в Ubuntu :

**dpkg --get-selections | grep node** - список установленных в системе пакетов Node

**node -v, npm -v** - посмотреть версии

**sudo apt purge nodejs** - удалить Node

**sudo apt remove** NodePackageName - удаление оставшихся пакетов Node

**sudo apt install nodejs** - установить Node

**sudo apt install npm** - установить node package manager

**- Создание недостающих модулей, реализующих разграничение доступа(13.2.):**

Для реализации разграничения доступа используется ряд программных модулей: контроллеров, провайдеров, посредников и шаблонов. Часть из них уже присутствует во вновь созданном Laravel-пpoeктe, остальные придется создавать отдельно.

Для создания остальных модулей нам понадобится установить одну из библиотек. Установка выполняется в папке проекта!!!

Библиотека устанавливается командой:

**composer require laravel/ui**

**composer require laravel/breeze --dev**

Модули создаются командой для библиотеки:

**php artisan ui vue --auth**

**php artisan ui bootstrap --auth**

--views- создать только шаблоны (контроллеры созданы не будут);

--force - перезаписать существующие модули, не спрашивая разрешения.

**php artisan breeze:install**

**php artisan breeze:install --dark**

**php artisan breeze:install vue**

**php artisan breeze:install react**

Установить модули Node в поект:

**npm install**

**npm run dev**

**npm run build**

Примечание: если для view библиотек разграничения доступа не будут используют frontend frameworks, то установить их можно:

**composer require laravel/ui**

**php artisan ui:auth**

и удалить во view resources\views\layouts\app.blade.php строку

@vite(['resources/sass/app.scss', 'resources/js/app.js'])

**- Маршруты, ведущие на контроллеры разграничения доступа(13.3):**

команда **php artisan ui:auth** добавляет в модуль routes\web.php, два маршрута:

**Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');**

выводит страницу с разделом пользователя после успешной авторизации.

**Auth::routes();** - вызывает у фасада Illurninate\Support\Facades\Auth метод routes(), который может принимать ассоциативный массив и создает группу маршрутов.

В **ассоциативном массиве** ключи элементов должны соответствовать именам параметров, а их значения должны иметь логический тип:

register - если true, создаются маршруты на контроллер Auth\Registercontroller (поведение по умолчанию), если false - не создаются;

reset - если true, создаются маршруты на контроллеры Auth\ForgotPasswordController и Auth\ResetPasswordController (поведение по умолчанию), если false -не создаются;

verify - если true, создаются маршруты на контроллер Auth\VerificationController (поведение по умолчанию), если false -не создаются;

confirm - если true, создаются маршруты на контроллер Auth\confirrnPasswordController (поведение по умолчанию), если false - не создаются.

Пример: Auth::routes(['register' => false, 'verify' => false]);

**Маршруты** (записаны в формате

допустимый НТТРметод - шаблонный путь - целевое действие контроллера - имя маршрута):

GET - /login - Auth\LoginController@showLoginForrn - login

Вывод страницы с веб-формой входа на сайт;

POST - /login - Auth\LoginController@login

Собственно выполнение входа;

POST - /logout - Auth\LoginController@logout - logout

Выполнение выхода с сайта;

Следующие два маршрута не создаются, если параметру register было дано значение false

GET - /register - Auth\RegisterController@showRegistrationForrn - register

Вывод страницы с веб-формой регистрации нового пользователя;

POST-/register-Auth\RegisterController@register

Собственно регистрация нового пользователя в списке

Следующие четыре маршрута не создаются, если параметру reset было дано значение false

GET - /password/reset - Auth\ForgotPasswordController@showLinkRequestForm - password.request

Вывод страницы с веб-формой отправки электронного письма с гиперссылкой для

сброса пароля;

POST - /password/email - Auth\ForgotPasswordController@sendResetLinkErnail - password.email

Собственно отправка электронного письма с rиперссылкой для сброса пароля;

GET - /password/reset/{token} - Auth\ResetPasswordController@showResetForm - password.reset

Вывод страницы с веб-формой сброса пароля. В URL-параметре token передается

электронный жетон сброса пароля;

POST - /password/reset - Auth\ResetPasswordController@reset - password.update

Собственно сброс пароля

Следующие три маршрута не создаются, если параметру verify было дано значение false

GET - /email/verify - Auth\VerificationController@show - verification.notice

Вывод страницы с веб-формой проверки существования адреса электронной почты,

указанного при регистрации;

POST - /email/resend - Auth\VerificationController@resend - verification.resend

Отправка по указанному адресу электронного письма с гиперссылкой проверки этого

адреса;

GET - /email/verify{id}/{hash} - Auth\VerificationController@verify - verification. verify

Собственно подтверждение существования адреса электронной почты после

перехода по гиперссылке, полученной в электронном письме

Следующие два маршрута не создаются, если параметру confirm было дано значение false

GET - /password/confirm - Auth\ConfirmPasswordController@showConfirmForm - password.confirm

Вывод страницы с веб-формой подтверждения пароля;

POST - /password/confirm - Auth\ConfirmPasswordController@confirm

Собственно подтверждение пароля.

**- Служебные таблицы и модель(13.4):**

Кратко описывается миграция таблицы пользователей и App\Models\User.

В Laravel есть возможность добавления необходимых полей в таблицу пользователей и связывания с другой таблицей/моделью!!!

**- Регистрация новых пользователей(13.5):**

Подробно описывается функционал Auth/RegisterController. Далее кратко методы:

showRegistrationForm() - выводит страницу на основе шаблона auth\register.blade.php, с полями для ввода данных о новом пользователе и кнопкой "отправить", ведущей на следующий метод.

registered(Request $request, $user) - производит валидацию введенных данных, сохраняет в БД и редиректит на домашнюю страницу нового пользователя.

**- Вход на веб-сайт(13.6):**

Подробно описывается функционал Auth\LoginController. Далее кратко методы:

showLoginForm() - выводит страницу на основе шаблона auth\login.blade.php.

login() - Проверяет занесенные в веб-форму данные на корректность. Выясняет, не превышено ли количество допустимых попыток входа. Выполняет вход. Сбрасывает счетчик безуспешных попыток.

authenticated (Request $request, $user) - Должен возвращать серверный ответ после успешного входа с редиректом на домашнюю страницу пользователя.

Для изменения способа входа на пару логин\пароль необходимо в классе LoginController переопределить метод трейта:

public function username(){

return 'name';

}

Но дополнительно необходимо внести изменения: во view регистрации и логина, валидацию данных в контроллерах, таблицу users DB(логин должен быть уникальным), модель User (подробно опишу позже).

**- Раздел пользователя(13.7):**

Реализуется контроллером Homecontroller. Изначально содержит одно действие - index(), выводящее страницу раздела пользователя, на основе шаблона home.blade.php.

В конструкторе контроллера выполняется связывание всех действий с посредником auth - таким образом, войти в раздел пользователя можно лишь после авторизации.

**- Разграничение доступа с помощью посредников(13.8.1.1):**

auth - допускает только авторизированных пользователей.

guest - допускает только не авторизированных пользователей.

**- Гейты(13.8.2. ):**

Гейт это функция или метод класса определяющая, имеет ли указанный пользователь привилегии на выполнение заданной операции.

**- Написание гейтов(13.8.2.1):**

Гейты регистрируются в провайдере App\Providers\AuthServiceProvider, в методе boot(), после вызова метода registerPolicies() вызовом метода Gate::dеfinе('nameGate', <сам гейт>).

<Сам гейт> по умолчанию первым параметром получает залогиненного пользователя. Написать гейт можно двумя способами:

- Непосредственно в провайдере в виде анонимной функцией

namespace App\Providers;

use Illuminate\Support\Facades\Gate;

Gate::define('nameGate', function($user, $item){

return $user->id == $item->user\_id;

}

);

- Отдельным классом с регистрацией его в провайдере:

namespace App\Gates;

class NameGateClass{

public function owner($user, $item){

return $user->id == $item->user\_id;

}

}

namespace App\Providers;

use App\Gates\NameGateClass;

Gate::define('nameGate', [NameGateClass::class, 'owner']);

В книге не написано, но по идеи в этом случае, правильнее гейту присваивать метод политики (такой себе гейт - недополитика).

**- Разграничение доступа посредством гейтов(13.8.2.2):**

В контроллере у фасада Gate вызывается один из методов в формате:

namespace App\Http\Controllers;

use Illuminate\Support\Facades\Gate;

public function methodeName(Item $item){

if (Gate::allows('nameGate', $item)) { - если больше параметров см. книгу

//разрешено

}

else{

//запрещено

}

}

Методы фасада Gate::

**allows()** - возвращает true, если гейт разрешил выполнение операции,и false если нет;

**denies()** - возвращает false, если гейт разрешил выполнение операции,и true если нет;

**check()** - возвращает true, если все гейты дали разрешение

if (Gate::check(['nameGateOne', 'nameGateTwo'], [$user, $item])) {

//разрешить

}

**any()** - и возвращает true, если хотя бы один гейт дал разрешение;

**none()** - и возвращает true, если ни один гейт не дал разрешение;

**authorize()** - если гейт разрешил выполнение операции, ничего не делает, в противном случае возбуждает исключение AuthorizationException;

**has(**'nameGate'**)** - возвращает true, если с операцией с заданным наименованием был связан гейт;

Все приведенные выше методы проверяют привилегии текущего пользователя. Чтобы выполнить аналогичные проверки другого пользователя:

$anotherUser = User::firstWhere('name', 'editor');

if (Gate::forUser($anotherUser)->allows('nameGate', $item)) {

//разрешить

}

**- Предварительные и завершающие проверки(13.8.2.3):**

Внимание!!! Если были созданы - выполняются ВСЕГДА без вызова, перед Gate и Policy с возможным игнорированием их правил.

- Предварительная:

Gate::before(function($user){

if ($user->name == 'admin') {

return true; - разрешит админу все игнорируя правила Gate

}

elseif ($user->name == 'user2') {

return false; - запретит все user2 игнорируя правила Gate

}

else{

return null; - ничего не будет делать, передаст запрос Gate

}

});

- Завершающие: не разбирался(по идеи для работы необходимо передать больше параметров из Gate, см. книгу), не очень представляю где их использовать.

**- Гейты с развернутыми ответами(13.8.2.4):**

По умолчанию Gate должен возвращать true, false, null. Если этого не достаточно(х.з. только в каких случаях) можно вернуть "развернутый ответ" объектом класса Illuminate\Auth\Access\Response, функционал довольно богатый - смотри книгу. По эксперементировал - результат такой-же как и с простыми Gate, наверное расширенный ответ нужно чем-то отлавливать как Exeption, или передавать в ответ.

**- Политики(13.8.3):**

Политика это класс, который организует логику авторизации для конкретной модели или ресурса.

Политика связывается с моделью в AuthServiceProvider или автоматически при выполнении соглашения по именованию ItemPolicy => Item. При вызове модели Laravel сам применит связанный с ней класс политики, его вызывать не нужно. Метод политики необходимо вызывать в роутере, или методе контроллера, или конструкторе контроллера (для всех или определенных методов) используя middleware can, или метод can класса Route, или метод can объекта user, или метод authorize трейта AuthorizesRequests с передачей необходимых параметров. Метод класса политики по умолчанию первым параметром получает залогиненного пользователя, его передавать не нужно.

**- Создание и регистрация политики(13.8.3.1):**

Класс политики создается в App\Policies. Логика авторизации описывается в методе класса и может возвращать либо логическую величину, либо развернутый ответ в виде объекта класса Response (см. разд. 13.8.2.4).

- Создать "пустой" класс политики:

**php artisan make:policy NamePolicy**

- Написание метода политики:

class NamePolicy{

...

public function isOwner(User $user, Item $item){

return $user->id == $item->user\_id;

- Предварительная проверка:

Внимание!!! Если была создана - выполняется ВСЕГДА без вызова при использовании любого метода NamePolicy с возможным игнорированием его правил.

class NamePolicy{

use HandlesAuthorization;

//если имя текущего пользователя admin вернет trueи и проигнорирует остальные правила

public function **before**($user){

return $user->name == 'admin';

- Завершающую проверку в политике создать невозможно.

- Регистрация политики выполняется в файле AuthServiceProvider:

class AuthServiceProvider extends ServiceProvider{

protected $policies = [

'App\Models\Model' => 'App\Policies\ModelPolicy',

Name::class => NamePolicy::class,

];

Даже при выполнении соглашения по именованию Политики и Модели для увеличения производительности связывание лучше сделать явно.

**- Разграничение доступа посредством политик(13.8.3.2):**

Может быть выполнено одним из способов:

- В маршрутах при помощи middleware can с объектом модели:

Route::get('/address/{item}', [NameController::class, 'controllerMethodName'])->middleware('can:policyMethodName,item');

- В маршрутах при помощи middleware can без объекта модели:

Route::get('/address/{item}', [NameController::class, 'controllerMethodName'])->middleware('can:policyMethodName,App\Models\ModelName');

Передать политике более двух параметров способами указанными выше нельзя.

- В маршрутах при помощи метода can класса Route с объектом модели:

Route::get('/address/{item}', [NameController::class, 'controllerMethodName'])->can('policyMethodName', 'item');

- В маршрутах при помощи метода can класса Route без объекта модели:

Route::get('/address/{item}', [NameController::class, 'controllerMethodName'])->can('policyMethodName', ModelName::class);

- В маршрутах при помощи метода can класса Route с объектом модели передавая более двух параметров (не пробовал, может не работать):

Route::get('/address/{item}', [NameController::class, 'controllerMethodName'])->can('policyMethodName', ['item', $rubric]);

- В методе контроллера вызвав у объекта текущего пользователя метод can() с объектом модели:

public function update(Request $request, Model $model){

if ($request->user()->can('policyMethodName', $model)) {

return view('Success');

}

else{

return view('Denied');

//throw(new AuthorizationException());

- В методе контроллера вызвав у объекта текущего пользователя метод can() без объекта модели:

public function create(Request $request){

if ($request->user()->can('policyMethodName', Model::class)) {

return view('Success');

}

else{

return view('Denied');

//throw(new AuthorizationException());

- В методе контроллера вызвав у объекта текущего пользователя метод can() с объектом модели передавая более двух параметров:

public function destroy(Request $request, Model $model, Rubric $rubric){

if ($request->user()->can('policyMethodName', [$model, $rubric])) {

return view('Success');

}

else{

return view('Denied');

Так-же у объекта текущего пользователя можно воспользоваться методами cant() и cannot() логика работы которых обратна методу can().

- В методе контроллера вызвав метод authorize() трейта AuthorizesRequests с объектом модели:

public function update(Model $model){

$this->authorize('policyMethodName', $model);

return view('Success');

- В методе контроллера вызвав метод authorize() трейта AuthorizesRequests без объекта модели (не пробовал, может не работать):

public function create(){

$this->authorize('policyMethodName', App\Models\ModelName);

return view('Success');

- В методе контроллера вызвав метод authorize() трейта AuthorizesRequests с объектом модели передавая более двух параметров (не пробовал, может не работать):

public function create(Model $model, Rubric $rubric){

$this->authorize('policyMethodName', [$model, $rubric]);

return view('Success');

- В конструкторе контроллера при помощи middleware can для всех, некоторых или одного метода контроллера:

public function \_\_construct(){

$this->middleware('can:policyMethodName,item')->only('controllerMethodName');

Политики можно применять в шаблонах для отображения определенной информации только авторизированным пользователям (стр.311).

**- Разграничение доступа в ресурсных контроллерах(13.8.3.3):**

Для привязки ресурсной Policy в конструкторе ресурсного Controller-а необходимо вызвать метод authorizeResource() трейта AuthorizesRequests и передать ему путь к классу модели и имя ИRL-параметра, через который передается ключ записи(автоматически создается в ресурсном Route):

namespace App\Http\Controllers;

use App\Models\ResourcePolicyModel;

class ResourcePolicyController extends Controller{

public function \_\_construct(){

...

$this->authorizeResource(ResourcePolicyModel::class, 'resourcePolicy');

- Создать ресурсную политику для определенной модели:

**php artisan make:policy NamePolicy --model=Name**

Методы класса ресурсной политики:

class NamePolicy{

use HandlesAuthorization;

//проверяет привилегии на просмотр списка записей (без объекта модели)

public function viewAny(User $user){

return $user->role != 'baned'

}

//на просмотр отдельной записи

public function view(User $user, Model $model){

return $user->id == $model->user\_id;

}

//на создание записи

public function create(User $user){...}

//на правку записи

public function update(User $user, Model $model){...}

//на удаление записи

public function delete(User $user, Model $model){...}

//на восстановление «мягко» удаленной записи

public function restore(User $user, Model $model){...}

//на полное удаление записи

public function forceDelete(User $user, Model $model){...}

Каждому методу ресурсного контроллера Laravel привязывает метод ресурсной политики:

|  |  |
| --- | --- |
| **Controller** | **Policy** |
| Index() | viewAny() |
| show() | view() |
| create() | create() |
| store() | create() |
| edit() | update() |
| update() | update() |
| destroy() | delete() |

- Регистрация политики также выполняется в файле AuthServiceProvider (см привер выше).

- При не успешной авторизации возбуждается исключение AuthorizationException() с кодом 403.