Middleware

Middleware в Laravel используется для обработки HTTP-запросов перед тем, как они попадут в контроллер или после того, как они вернутся из контроллера. Промежуточное ПО (middleware) может выполнять различные задачи, такие как аутентификация, проверка прав доступа, запись журналов и многое другое.

Пример простого middleware:

Создание нового проекта Laravel

composer create-project laravel/laravel:^10.0 example-app

Где my-laravel-project - это имя вашего проекта. Замените его на нужное вам имя.

Перейдите в каталог вашего проекта:

cd YourProjectName

Создание представления

Создайте представление home.blade.php в директории resources/views с помощью следующей команды:

php artisan make:view

Откройте созданный файл home.blade.php для редактирования и добавьте в него следующий код:

<!DOCTYPE html>

<html>

<head>

    <title>Имитация авторизации</title>

</head>

<body>

    <h1>Имитация авторизации</h1>

    <p>{{ $auth\_message }}</p>

</body>

</html>

Создайте маршрут в файле routes/web.php для отображения страницы, на которой будет имитация авторизации:

<?php

use Illuminate\Support\Facades\Route;

Route::get('/home', 'HomeController@index')->middleware('auth.simulation');

Создайте контроллер HomeController, если его нет:

php artisan make:controller HomeController

В контроллере HomeController.php создайте метод index для отображения представления:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller

{

    public function index(Request $request) {

        $auth\_message = $request->get("auth\_message");

        return view('home', ['auth\_message' => $auth\_message]);

    }

}

Создайте middleware для имитации авторизации. Вы можете использовать следующую команду для создания middleware:

php artisan make:middleware AuthSimulationMiddleware

Откройте файл app/Http/Middleware/AuthSimulationMiddleware.php и внесите следующие изменения в метод handle:

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

use Symfony\Component\HttpFoundation\Response;

class AuthSimulationMiddleware

{

    /\*\*

     \* Handle an incoming request.

     \*

     \* @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next

     \*/

    public function handle($request, Closure $next)

    {

        // Проверяем, авторизован ли пользователь

        $userIsAuthenticated = false; // Здесь можно добавить условия для проверки авторизации

        if ($userIsAuthenticated) {

            // Если пользователь авторизован, добавьте текст для авторизованных пользователей

            $request->attributes->add(['auth\_message' => 'Вы авторизованы']);

        } else {

            // Если пользователь не авторизован, добавьте текст для неавторизованных пользователей

            $request->attributes->add(['auth\_message' => 'Вы не авторизованы']);

        }

        return $next($request);

    }

}

Зарегистрируйте middleware в файле app/Http/Kernel.php, добавив 'auth.simulation' в массиве $routeMiddleware:

<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel

{

    /\*\*

     \* The application's global HTTP middleware stack.

     \*

     \* These middleware are run during every request to your application.

     \*

     \* @var array<int, class-string|string>

     \*/

    //Сработает для всех маршрутов

    protected $middleware = [

        // \App\Http\Middleware\TrustHosts::class,

        \App\Http\Middleware\TrustProxies::class,

        \Illuminate\Http\Middleware\HandleCors::class,

        \App\Http\Middleware\PreventRequestsDuringMaintenance::class,

        \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,

        \App\Http\Middleware\TrimStrings::class,

        \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,

    ];

    /\*\*

     \* The application's route middleware groups.

     \*

     \* @var array<string, array<int, class-string|string>>

     \*/

    protected $middlewareGroups = [

        'web' => [

            \App\Http\Middleware\EncryptCookies::class,

            \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,

            \Illuminate\Session\Middleware\StartSession::class,

            \Illuminate\View\Middleware\ShareErrorsFromSession::class,

            \App\Http\Middleware\VerifyCsrfToken::class,

            \Illuminate\Routing\Middleware\SubstituteBindings::class,

        ],

        'api' => [

            // \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,

            \Illuminate\Routing\Middleware\ThrottleRequests::class.':api',

            \Illuminate\Routing\Middleware\SubstituteBindings::class,

        ],

    ];

    /\*\*

     \* The application's middleware aliases.

     \*

     \* Aliases may be used instead of class names to conveniently assign middleware to routes and groups.

     \*

     \* @var array<string, class-string|string>

     \*/

    //Сработает только для одиночніх маршрутов ->middleware('myMiddleware')

    protected $middlewareAliases = [

        'auth' => \App\Http\Middleware\Authenticate::class,

        'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,

        'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,

        'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,

        'can' => \Illuminate\Auth\Middleware\Authorize::class,

        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,

        'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,

        'precognitive' => \Illuminate\Foundation\Http\Middleware\HandlePrecognitiveRequests::class,

        'signed' => \App\Http\Middleware\ValidateSignature::class,

        'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,

        'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,

        'myMiddleware' => \App\Http\Middleware\AuthSimulationMiddleware::class,

    ];

}

Практика middleware

Создайте новый проект Laravel с помощью команды:

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

Перейдите в директорию вашего проекта:

cd my-laravel-project

Создайте middleware с помощью Artisan команды:

php artisan make:middleware MyMiddleware

Откройте созданный файл middleware в директории app/Http/Middleware/MyMiddleware.php и добавьте в него логику middleware:

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

use Symfony\Component\HttpFoundation\Response;

class MyMiddleware

{

    /\*\*

     \* Handle an incoming request.

     \*

     \* @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next

     \*/

    public function handle(Request $request, Closure $next)

    {

        if (Auth::check()) {

            // Если пользователь аутентифицирован, продолжаем выполнение запроса.

            return $next($request);

        }

        // Если пользователь не аутентифицирован, перенаправляем его на страницу входа.

        return redirect('/login');

    }

}

Теперь зарегистрируйте созданный middleware в файле app/Http/Kernel.php. Добавьте его в массив $middleware для глобального применения, либо в массив $routeMiddleware для использования в определенных маршрутах:

<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel

{

    /\*\*

     \* The application's global HTTP middleware stack.

     \*

     \* These middleware are run during every request to your application.

     \*

     \* @var array<int, class-string|string>

     \*/

    protected $middleware = [

        // \App\Http\Middleware\TrustHosts::class,

        \App\Http\Middleware\TrustProxies::class,

        \Illuminate\Http\Middleware\HandleCors::class,

        \App\Http\Middleware\PreventRequestsDuringMaintenance::class,

        \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,

        \App\Http\Middleware\TrimStrings::class,

        \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,

        \App\Http\Middleware\MyMiddleware::class, // Ваш middleware MyMiddleware

    ];

    /\*\*

     \* The application's route middleware groups.

     \*

     \* @var array<string, array<int, class-string|string>>

     \*/

    protected $middlewareGroups = [

        'web' => [

            \App\Http\Middleware\EncryptCookies::class,

            \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,

            \Illuminate\Session\Middleware\StartSession::class,

            \Illuminate\View\Middleware\ShareErrorsFromSession::class,

            \App\Http\Middleware\VerifyCsrfToken::class,

            \Illuminate\Routing\Middleware\SubstituteBindings::class,

            \App\Http\Middleware\MyMiddleware::class, // Ваш middleware MyMiddleware

        ],

        'api' => [

            // \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,

            \Illuminate\Routing\Middleware\ThrottleRequests::class.':api',

            \Illuminate\Routing\Middleware\SubstituteBindings::class,

        ],

    ];

    /\*\*

     \* The application's middleware aliases.

     \*

     \* Aliases may be used instead of class names to conveniently assign middleware to routes and groups.

     \*

     \* @var array<string, class-string|string>

     \*/

    protected $middlewareAliases = [

        'auth' => \App\Http\Middleware\Authenticate::class,

        'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,

        'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,

        'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,

        'can' => \Illuminate\Auth\Middleware\Authorize::class,

        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,

        'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,

        'precognitive' => \Illuminate\Foundation\Http\Middleware\HandlePrecognitiveRequests::class,

        'signed' => \App\Http\Middleware\ValidateSignature::class,

        'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,

        'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,

    ];

}

Теперь вы можете использовать middleware в маршрутах. Например, в файле routes/web.php:

<?php

use Illuminate\Support\Facades\Route;

Route::get('/', function () {

    return view('welcome');

});

Route::get('/protected', function () {

    // Этот маршрут будет использовать middleware MyMiddleware

    // для проверки аутентификации пользователя.

    return "Этот маршрут защищен middleware";

})->middleware('mymiddleware');

Запустите встроенный сервер Laravel с помощью команды:

php artisan serve

Перед началом обязательно удалить базу данных Laravel

composer create-project laravel/laravel crud\_app

cd crud\_app

composer require laravel/ui

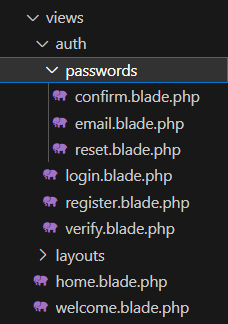
Команда composer require laravel/ui используется в Laravel для установки пакета laravel/ui. Этот пакет предоставляет инструменты для создания пользовательского интерфейса (UI) в вашем Laravel-приложении. Он может быть полезен, если вы хотите добавить фронтенд-фреймворк, такой как Bootstrap

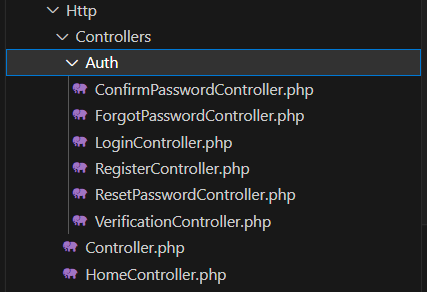
php artisan ui bootstrap –auth

npm install

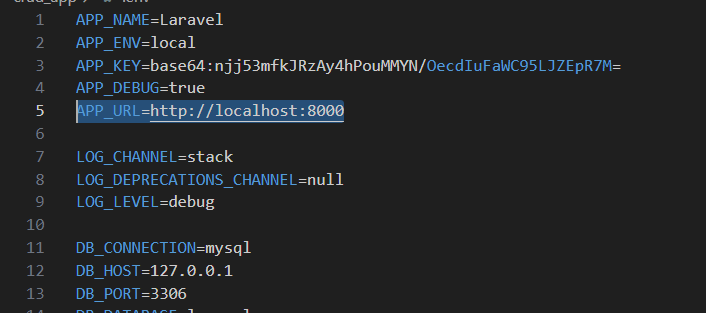
npm run build

После установки пакета laravel/ui, выполните одну из команд, чтобы настроить аутентификацию. Эти команды создадут необходимые файлы и маршруты для аутентификации в вашем приложении.

У нас появится в проекте views:  


Controllers:  


Дальше запустим через vite наш проект, для этого изменим .env:



Запускаем команду:

npm install  
дальше запускаем в отдельной командной строке сам сервер Laravel:  
php artisan serve

и в отдельной cmd запускаем vite:  
npm run dev

Vite

Vite — фронтенд инструмент следующего поколения, который с теперь используется в Laravel по умолчанию. Документация Laravel содержит целый раздел объясняющий как он работает и как его использовать.

Vite можно рассматривать как альтернативу Webpack, особенно в контексте сборки фронтенда для веб-приложений. Оба инструмента предназначены для сборки и оптимизации ресурсов фронтенда, таких как JavaScript, CSS и изображения, но они имеют разные подходы и особенности.

Основные различия между Vite и Webpack

* Скорость разработки: Vite спроектирован с упором на быструю скорость разработки. Он предоставляет живую перезагрузку и горячую замену модулей (HMR), что позволяет мгновенно видеть изменения в коде без полной пересборки проекта. В то время как Webpack, как правило, требует больше времени на сборку и пересборку проекта.
* Инфраструктура разработки: Vite поставляется с собственным сервером разработки, который обрабатывает запросы к фронтенд-ресурсам. Webpack обычно требует настройки сервера разработки отдельно.
* Оптимизированная сборка: Vite оптимизирует сборку для продакшена, минимизируя и объединяя файлы, чтобы обеспечить лучшую производительность. Webpack также обладает мощными возможностями оптимизации.
* Интеграция с фреймворками: Vite легко интегрируется с современными JavaScript фреймворками, такими как Vue.js и React. Webpack также может быть интегрирован, но требует больше настроек.

Оба инструмента имеют свои преимущества и недостатки, и выбор между ними зависит от конкретных потребностей вашего проекта и вашей команды разработки. Если вам важна быстрая скорость разработки и вы работаете с современными фреймворками, Vite может быть хорошим выбором. Однако Webpack остается мощным и гибким инструментом с обширной экосистемой плагинов и настроек для различных сценариев разработки и производства.

Практика vite:

Для создания сборки с использованием Vite и обычного JavaScript (без фреймворков, таких как Vue.js или React), вы можете использовать следующий пример.

Создайте новый проект Vite с помощью команды:

npm init vite@latest my-vite-project -- --template vanilla

Эта команда создаст новый проект Vite с шаблоном vanilla, что означает, что проект будет использовать обычный JavaScript без фреймворков.

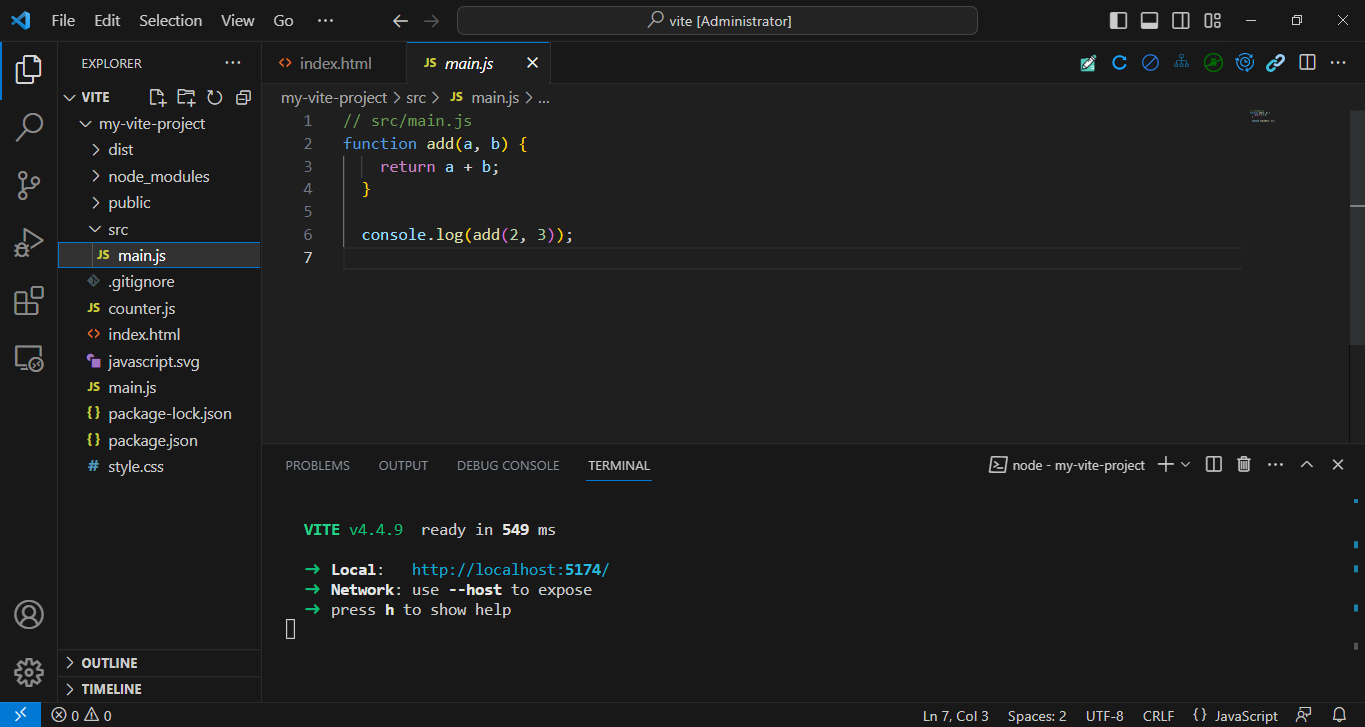
Перейдите в директорию вашего нового проекта:

cd my-vite-project

Установите зависимости проекта:

npm install

Создайте файлы JavaScript в директории src. Например, вы можете создать файл main.js с содержанием:



В файле index.html нужно прописать путь к файлу main.js

<!doctype html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <link rel="icon" type="image/svg+xml" href="/vite.svg" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Vite App</title>

  </head>

  <body>

    <div id="app"></div>

    <script type="module" src="./src/main.js"></script>

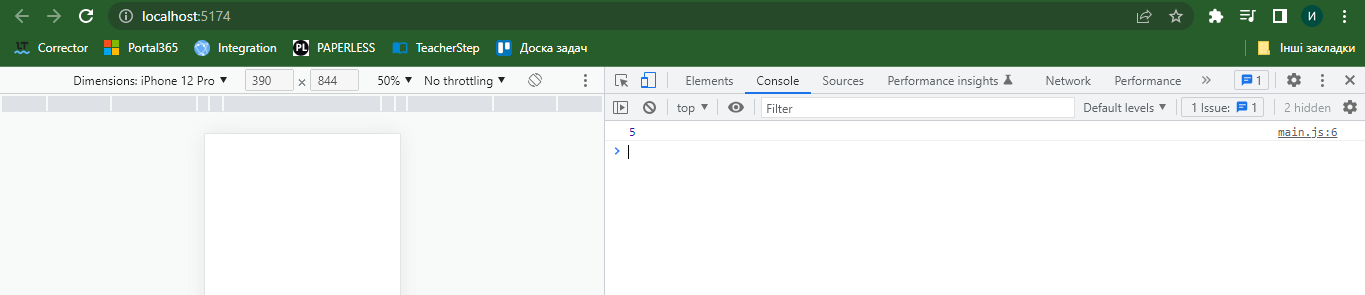
  </body>

</html>

Для теста можно проверить командой:

npm run dev

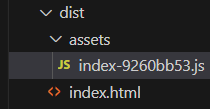
result:



Теперь вы можете выполнить сборку проекта с помощью команды:

npm run build

Эта команда создаст оптимизированные файлы сборки в директории dist. Вы найдете главный файл приложения, например, main.js, а также файлы для стилей и другие ресурсы, если вы используете их.



Ваше приложение готово к развертыванию. Вы можете разместить содержимое директории dist на вашем веб-сервере или хостинге, и ваше веб-приложение будет доступно в браузере.

Вернемся к нашему проекту и подключим css из папки resources/css/app.css:

app.css

.h1 {

    color: red;

}

Добавим разметку в файл resources/views/welcome.blade.php например сразу после body:

        <h1>Test css</h1>

В файле vite.config.js

import { defineConfig } from 'vite';

import laravel from 'laravel-vite-plugin';

export default defineConfig({

    plugins: [

        laravel({

            input: [

                'resources/css/app.css',

                'resources/js/app.js',

            ],

            refresh: true,

        }),

    ],

});

И в файле layouts/app.blade.php

    <!-- Scripts -->

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

Добавить в welcome.blade.php

@extends('layouts.app')

Если Вы хотите запустить стили через сервер Laravel, то нужно выполнить команду:  
npm run build

Если Ві хотите создавать из папки resources\css\app.css диначически стилизацию, то нужно в .env добавить порт в

APP\_URL=http://localhost:8000

Запустить отдельно cmd и vite  
npm install

npm run build

npm run dev

Приложение TaskManager (Регистрация, Авторизация, CRUD, права доступа)

Шаг 1: Установка Laravel

Первым шагом является установка Laravel на вашем сервере или локальной машине. Для этого вам понадобится Composer, инструмент для управления зависимостями PHP. Выполните следующие команды в командной строке:

composer create-project laravel/laravel task-manager

cd task-manager

Это создаст новый проект Laravel с именем "task-manager" и перейдет в его директорию.

Шаг 2: Настройка базы данных

Настройте вашу базу данных в файле .env, который находится в корне вашего проекта. Укажите параметры подключения к базе данных, такие как имя базы данных, имя пользователя и пароль.

Шаг 3: Установка и настройка Laravel Passport

Laravel Passport - это пакет аутентификации и выдачи токенов доступа, который мы будем использовать для аутентификации пользователей и управления правами доступа. Установите его с помощью Composer:

Установка laravel/passport (Внимание если у Вас ошибка, то решение есть ниже):

composer require laravel/passport

Laravel Passport - это официальный пакет аутентификации и выдачи токенов доступа (API-токенов) для фронтенда и мобильных приложений в Laravel.

Passport позволяет вам легко добавлять аутентификацию через токены доступа к вашему Laravel-приложению. Это особенно полезно, когда вы разрабатываете RESTful API или приложение, которое требует аутентификации через API-токены, такие как мобильные приложения и одностраничные приложения (SPA).

Основные возможности Laravel Passport включают:

* Выдачу и управление токенами доступа: Passport предоставляет мощные инструменты для генерации, хранения и управления токенами доступа, включая возможность создания долгосрочных и одноразовых токенов.
* Аутентификацию через токены: Passport позволяет пользователям аутентифицироваться, предоставляя действительный токен доступа в запросе к API.
* Создание OAuth2-сервера: Passport обеспечивает полную реализацию сервера OAuth2 для создания стандартных потоков авторизации, таких как авторизация по паролю, авторизация через код подтверждения и другие.
* Интеграцию с Laravel аутентификацией: Passport интегрируется хорошо с встроенной аутентификацией Laravel, что позволяет вам использовать как сессионную аутентификацию, так и аутентификацию через токены в рамках одного приложения.
* Простую настройку: Установка и настройка Passport с минимальным кодом является относительно простой задачей благодаря готовым миграциям и командам Artisan.

Чтобы использовать Laravel Passport в вашем проекте, после установки с помощью composer require laravel/passport, вам нужно будет выполнить дополнительные шаги настройки, такие как выполнение миграций базы данных, настройка моделей и маршрутов для аутентификации через API-токены. Вы также должны будете рассмотреть документацию Laravel Passport, чтобы узнать больше о его возможностях и настройках.

Passport - мощный инструмент для обеспечения безопасной аутентификации и авторизации в вашем Laravel-приложении и использования его в API.

Токены

Laravel Passport формирует два основных типа токенов: Access Tokens (токены доступа) и Refresh Tokens (токены обновления). Они служат для обеспечения безопасной аутентификации и авторизации в приложении и API.

Access Tokens (API Tokens):

Что это такое: Access Token (токен доступа) - это короткая строка (обычно случайная), которая предоставляет доступ к определенным ресурсам или функциям в вашем приложении или API. Этот токен предоставляется пользователю или клиенту после успешной аутентификации и авторизации.

Как они генерируются: При успешной аутентификации и авторизации, Laravel Passport генерирует уникальный Access Token, который затем предоставляется клиенту. Этот токен обычно имеет ограниченный срок действия (например, 1 час) и должен быть включен в каждый запрос к вашему API, чтобы показать, что пользователь аутентифицирован и имеет доступ.

Зачем они нужны: Access Tokens используются для обеспечения безопасного доступа к ресурсам API. Они позволяют серверу знать, что запрос приходит от аутентифицированного пользователя или клиента, и предоставить соответствующие права доступа.

Refresh Tokens:

Что это такое: Refresh Token (токен обновления) - это также короткая строка, но он используется для обновления Access Token без необходимости повторной аутентификации пользователя. Refresh Token предоставляется вместе с Access Token и имеет долгий срок действия (например, 30 дней).

Как они генерируются: После успешной аутентификации, Laravel Passport генерирует и возвращает как Access Token, так и Refresh Token. Refresh Token сохраняется на стороне клиента и используется для запроса новых Access Tokens при истечении срока действия текущего Access Token.

Зачем они нужны: Refresh Tokens обеспечивают безопасное и удобное обновление Access Tokens. Это особенно полезно в сценариях, где пользователь может оставаться аутентифицированным долгое время (например, в мобильных приложениях). Refresh Token позволяет пользователю оставаться аутентифицированным без необходимости повторной аутентификации при каждом запросе.

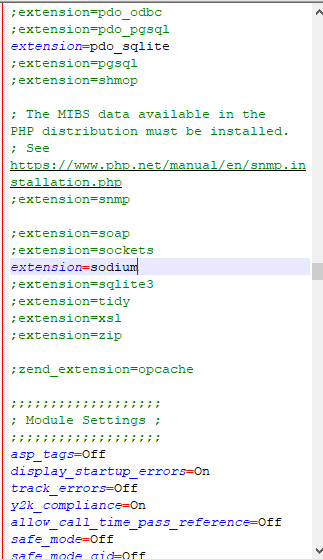
Токены обеспечивают безопасную аутентификацию и авторизацию в вашем приложении или API, а также позволяют разработчикам контролировать доступ к ресурсам и управлять сессиями пользователей. Laravel Passport делает процесс создания и управления этими токенами относительно простым и стандартизированным, что делает его мощным инструментом для защиты вашего API.

Если при установке composer require laravel/passport ошибка, то в php.ini на xampp нужно расскоментировать строку

Ошибка, которую вы видите, связана с несовместимостью версий пакетов и требований, а также отсутствием расширения PHP sodium. Для разрешения этой проблемы вы можете предпринять следующие шаги:

Установите расширение PHP sodium, которое требуется для пакета lcobucci/jwt. Если вы используете XAMPP, проверьте, установлено ли расширение sodium в вашем файле php.ini. Если нет, откройте php.ini и раскомментируйте (уберите ; перед) строку extension=sodium. После этого перезапустите сервер PHP.

После того как расширение sodium установлено и активировано, попробуйте установить laravel/passport снова:



php artisan migrate (yes)

Затем выполните команду passport:install, чтобы создать клиентские и персональные доступы:

php artisan passport:install

Шаг 4: Создание модели и миграции для задач

Создайте модель и миграцию для сущности "Task":

php artisan make:model Task -m

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Task extends Model

{

    use HasFactory;

    protected $fillable = [

        'user\_id',

        'title',

        'description',

        // Другие поля, которые вы хотите разрешить для массового заполнения

    ];

}

-m – создать миграцию

Редактируйте миграцию в файле, созданном в папке database/migrations, чтобы определить поля таблицы для задач.

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreateTasksTable extends Migration

{

    public function up()

    {

        Schema::create('tasks', function (Blueprint $table) {

            $table->id();

            $table->string('title');

            $table->text('description')->nullable();

            $table->boolean('completed')->default(false);

            $table->unsignedBigInteger('user\_id');

            $table->timestamps();

            $table->foreign('user\_id')->references('id')->on('users')->onDelete('cascade');

        });

    }

    public function down()

    {

        Schema::dropIfExists('tasks');

    }

}

?>

php artisan migrate

Теперь у вас есть таблица "tasks" с полями "id", "title", "description", "completed", "user\_id" и временными метками "created\_at" и "updated\_at".

Добавим сразу таблицу ролей:

php artisan make:migration create\_roles\_table

Внесите необходимые изменения в созданные миграции. Вот пример миграции для ролей (create\_roles\_table.php):

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreateRolesTable extends Migration

{

    public function up()

    {

        Schema::create('roles', function (Blueprint $table) {

            $table->id();

            $table->string('name')->unique();

            $table->timestamps();

        });

    }

    public function down()

    {

        Schema::dropIfExists('roles');

    }

}

И миграция для пользователей (create\_users\_table.php):

<?php

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

class CreateUsersTable extends Migration

{

    public function up()

    {

        Schema::create('users', function (Blueprint $table) {

            $table->id();

            $table->string('name');

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

            $table->timestamp('email\_verified\_at')->nullable();

            $table->string('password');

            $table->unsignedBigInteger('role\_id')->default(2); // Роль "обычного пользователя" по умолчанию

            $table->rememberToken();

            $table->timestamps();

        });

        Schema::table('users', function (Blueprint $table) {

            $table->foreign('role\_id')->references('id')->on('roles');

        });

    }

    public function down()

    {

        Schema::dropIfExists('users');

    }

}

Запустите миграции:

php artisan migrate

Создайте модели для ролей:

php artisan make:model Role

Определите отношение между моделью User и моделью Task:

В модели User (app/Models/User.php) добавьте отношение "tasks":

<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Notifications\Notifiable;

use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable

{

    use HasApiTokens, HasFactory, Notifiable;

    /\*\*

     \* The attributes that are mass assignable.

     \*

     \* @var array<int, string>

     \*/

    protected $fillable = [

        'name',

        'email',

        'password',

    ];

    /\*\*

     \* The attributes that should be hidden for serialization.

     \*

     \* @var array<int, string>

     \*/

    protected $hidden = [

        'password',

        'remember\_token',

    ];

    /\*\*

     \* The attributes that should be cast.

     \*

     \* @var array<string, string>

     \*/

    protected $casts = [

        'email\_verified\_at' => 'datetime',

        'password' => 'hashed',

    ];

    // public function tasks()

    // {

    //     return $this->hasMany(Task::class);

    // }

    public function role()

    {

        return $this->belongsTo(Role::class);

    }

     public function hasRole($role)

    {

        return $this->role === $role; // Проверяет, является ли роль пользователя указанной ролью

    }

}

В модели Role.php определите связь с пользователями:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Role extends Model

{

    use HasFactory;

    public function users()

    {

        return $this->hasMany(User::class);

    }

}

Шаг 5: Создание контроллеров

Создайте контроллеры и представления для управления ролями и пользователями.

php artisan make:controller RoleController

php artisan make:controller UserController

Реализуйте действия в контроллерах для создания, редактирования и удаления ролей и для управления пользователями. Например:

RoleController.php:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\Role;

class RoleController extends Controller

{    public function index()

    {

        $roles = Role::all();

        return view('roles.index', compact('roles'));

    }

    public function create()

    {

        return view('roles.create');

    }

    public function store(Request $request)

    {

        $validatedData = $request->validate([

            'name' => 'required|unique:roles|max:255',

        ]);

        Role::create($validatedData);

        return redirect()->route('roles.index')->with('success', 'Role created successfully.');

    }

    public function edit(Role $role)

    {

        return view('roles.edit', compact('role'));

    }

    public function update(Request $request, Role $role)

    {

        $validatedData = $request->validate([

            'name' => 'required|unique:roles|max:255',

        ]);

        $role->update($validatedData);

        return redirect()->route('roles.index')->with('success', 'Role updated successfully.');

    }

    public function destroy(Role $role)

    {

        $role->delete();

        return redirect()->route('roles.index')->with('success', 'Role deleted successfully.');

    }

}

UserController.php:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\User;

use App\Models\Role;

class UserController extends Controller

{

    public function index()

    {

        $users = User::all();

        return view('users.index', compact('users'));

    }

    public function edit(User $user)

    {

        $roles = Role::all();

        return view('users.edit', compact('user', 'roles'));

    }

    public function update(Request $request, User $user)

    {

        $validatedData = $request->validate([

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

            'email' => 'required|email|unique:users,email,' . $user->id,

            'role\_id' => 'required|exists:roles,id',

        ]);

        $user->update($validatedData);

        return redirect()->route('users.index')->with('success', 'User updated successfully.');

    }

    public function destroy(User $user)

    {

        $user->delete();

        return redirect()->route('users.index')->with('success', 'User deleted successfully.');

    }

}

Добавим middleware:

php artisan make:middleware IsModerator

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Http\Request;

use Symfony\Component\HttpFoundation\Response;

use Illuminate\Support\Facades\Auth;

class IsModerator

{

    public function handle($request, Closure $next)

    {

        // Проверяем, является ли текущий аутентифицированный пользователь модератором

        if (Auth::check() && Auth::user()->hasRole('moderator')) {

            return $next($request);

        }

        return redirect()->route('home')->with('error', 'Access denied.');

    }

}

Теперь миддлваре будет проверять, имеет ли текущий аутентифицированный пользователь указанную роль ('moderator' в данном случае).

Создайте контроллер для задач:

php artisan make:controller TaskController

В контроллере TaskController определите методы для CRUD операций:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\Task;

class TaskController extends Controller

{

    public function index()

    {

        $tasks = Task::where('user\_id', auth()->id())->get();

        return view('tasks.index', compact('tasks'));

    }

    public function create()

    {

        return view('tasks.create');

    }

    public function store(Request $request)

    {

        $validatedData = $request->validate([

            'title' => 'required|max:255',

            'description' => 'nullable',

        ]);

        $validatedData['user\_id'] = auth()->id();

        Task::create($validatedData);

        return redirect()->route('tasks.index');

    }

    public function edit(Task $task)

    {

        return view('tasks.update', compact('task'));

    }

    public function update(Request $request, Task $task)

    {

        $validatedData = $request->validate([

            'title' => 'required|max:255',

            'description' => 'nullable',

        ]);

        $task->update($validatedData);

        return redirect()->route('tasks.index');

    }

    public function destroy(Task $task)

    {

        $task->delete();

        return redirect()->route('tasks.index');

    }

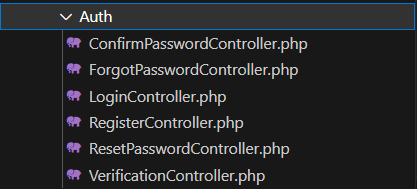
}

Добавим классы для регистрации и аунтефикации, а так же views:

Все шаблоны аутентификации были перенесены в laravel/uiрепозиторий.

composer require laravel/ui

php artisan ui bootstrap –auth



Так же проверим еще HomeController если нет, то добавим:  
php artisan make:controller HomeController  
HomeController:

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller

{

    /\*\*

     \* Create a new controller instance.

     \*

     \* @return void

     \*/

    public function \_\_construct()

    {

        $this->middleware('auth');

    }

    /\*\*

     \* Show the application dashboard.

     \*

     \* @return \Illuminate\Contracts\Support\Renderable

     \*/

    public function index()

    {

        return view('home');

    }

}

И добавим представление главной страницы resources/views/home.blade.php

@extends('layouts.app')

@section('content')

<div class="container">

    <div class="row justify-content-center">

        <div class="col-md-8">

            <div class="card">

                <div class="card-header">{{ \_\_('Dashboard') }}</div>

                <div class="card-body">

                    @if (session('status'))

                        <div class="alert alert-success" role="alert">

                            {{ session('status') }}

                        </div>

                    @endif

                    {{ \_\_('You are logged in!') }}

                    <div class="mt-3">

                        <a href="{{ route('tasks.index') }}" class="btn btn-primary">View Your Tasks</a>

                        <a href="{{ route('tasks.create') }}" class="btn btn-success">Create a New Task</a>

                    </div>

                </div>

            </div>

        </div>

    </div>

</div>

@endsection

Шаг 6: Настройка маршрутов

Настройте маршруты для CRUD операций и аутентификации в файле routes/web.php и routes/api.php.

web.php

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\Auth\LoginController;

use App\Http\Controllers\Auth\RegisterController;

use App\Http\Controllers\Auth\ForgotPasswordController;

use Illuminate\Support\Facades\Auth;

use App\Http\Controllers\HomeController;

use App\Http\Controllers\TaskController;

use App\Http\Controllers\RoleController; // Добавлен RoleController

use App\Http\Controllers\UserController; // Добавлен UserController

Auth::routes();

// Главная страница

Route::get('/', [HomeController::class, 'index'])->name('home');

// Маршруты для аутентификации

Route::get('login', [LoginController::class, 'showLoginForm'])->name('login');

Route::post('login', [LoginController::class, 'login']);

Route::post('logout', [LoginController::class, 'logout'])->name('logout');

Route::get('register', [RegisterController::class, 'showRegistrationForm'])->name('register');

Route::post('register', [RegisterController::class, 'register']);

Route::get('password/reset', [ForgotPasswordController::class, 'showLinkRequestForm'])->name('password.request');

Route::post('password/email', [ForgotPasswordController::class, 'sendResetLinkEmail'])->name('password.email');

Route::get('password/reset/{token}', [ForgotPasswordController::class, 'showResetForm'])->name('password.reset');

Route::post('password/reset', [ForgotPasswordController::class, 'reset'])->name('password.update');

// Маршруты для CRUD операций с задачами

Route::middleware('auth')->group(function () {

    Route::get('tasks', [TaskController::class, 'index'])->name('tasks.index');

    Route::get('tasks/create', [TaskController::class, 'create'])->name('tasks.create');

    Route::post('tasks', [TaskController::class, 'store'])->name('tasks.store');

    Route::get('tasks/{task}/edit', [TaskController::class, 'edit'])->name('tasks.edit');

    Route::put('tasks/{task}', [TaskController::class, 'update'])->name('tasks.update');

    Route::delete('tasks/{task}', [TaskController::class, 'destroy'])->name('tasks.destroy');

});

// Маршруты для управления ролями и пользователями

Route::middleware(['auth', 'moderator'])->group(function () {

    Route::resource('roles', RoleController::class); // CRUD для ролей

    Route::resource('users', UserController::class); // CRUD для пользователей

});

?>

Шаг 7: Создание и настройка представлений

Контент для роли Гость и роль авторизированного пользователя:

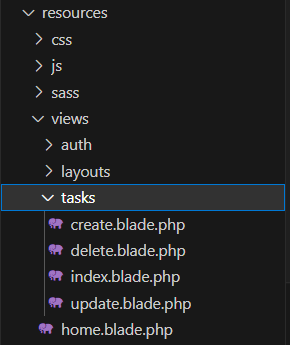
                    @guest

                        <h1>Гость!</h1>

                    @else

                        {{ \_\_('Main page!') }}

                    @endguest



php artisan make:view index

<!-- resources/views/tasks/index.blade.php -->

@extends('layouts.task')

@section('content')

    <h1>Task List</h1>

    <a href="{{ route('tasks.create') }}" class="btn btn-primary mb-3">Create Task</a>

    @if (count($tasks) > 0)

        <table class="table">

            <thead>

                <tr>

                    <th>Title</th>

                    <th>Description</th>

                    <th>Actions</th>

                </tr>

            </thead>

            <tbody>

                @foreach ($tasks as $task)

                    <tr>

                        <td>{{ $task->title }}</td>

                        <td>{{ $task->description }}</td>

                        <td>

                            <a href="{{ route('tasks.edit', ['task' => $task->id]) }}" class="btn btn-primary btn-sm">Edit</a>

                            <form action="{{ route('tasks.destroy', ['task' => $task->id]) }}" method="POST" style="display: inline;">

                                @csrf

                                @method('DELETE')

                                <button type="submit" class="btn btn-danger btn-sm" onclick="return confirm('Are you sure?')">Delete</button>

                            </form>

                        </td>

                    </tr>

                @endforeach

            </tbody>

        </table>

    @else

        <p>No tasks available.</p>

    @endif

@endsection

php artisan make:view create

<!-- resources/views/tasks/create.blade.php -->

@extends('layouts.task')

@section('content')

    <h1>Create Task</h1>

    <form method="POST" action="{{ route('tasks.store') }}">

        @csrf

        <div class="form-group">

            <label for="title">Title</label>

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

        </div>

        <div class="form-group">

            <label for="description">Description</label>

            <textarea class="form-control" id="description" name="description"></textarea>

        </div>

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

    </form>

@endsection

php artisan make:view update

<!-- resources/views/tasks/update.blade.php -->

@extends('layouts.task')

@section('content')

    <h1>Edit Task</h1>

    <form method="POST" action="{{ route('tasks.update', ['task' => $task->id]) }}">

        @csrf

        @method('PUT')

        <div class="form-group">

            <label for="title">Title</label>

            <input type="text" class="form-control" id="title" name="title" value="{{ $task->title }}" required>

        </div>

        <div class="form-group">

            <label for="description">Description</label>

            <textarea class="form-control" id="description" name="description">{{ $task->description }}</textarea>

        </div>

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

    </form>

@endsection

php artisan make:view delete

<!-- resources/views/tasks/delete.blade.php -->

@extends('layouts.task')

@section('content')

    <h1>Delete Task</h1>

    <p>Are you sure you want to delete this task?</p>

    <form method="POST" action="{{ route('tasks.destroy', ['task' => $task->id]) }}">

        @csrf

        @method('DELETE')

        <button type="submit" class="btn btn-danger">Delete</button>

        <a href="{{ route('tasks.index') }}" class="btn btn-secondary">Cancel</a>

    </form>

@endsection

Добавить views/layouts/task.blade.php

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">

    <title>{{ config('app.name', 'Laravel') }}</title>

    <!-- Дополнительные стили и скрипты могут быть подключены здесь -->

</head>

<body>

    <header>

        <nav>

            <ul>

                @guest

                    <!-- Если пользователь не аутентифицирован -->

                    <li><a href="{{ route('login') }}">Login</a></li>

                    <li><a href="{{ route('register') }}">Register</a></li>

                @else

                    <!-- Если пользователь аутентифицирован -->

                    <li><a href="{{ route('tasks.index') }}">Tasks</a></li>

                    <li><a href="{{ route('tasks.create') }}">Create Task</a></li>

                    <!-- Добавьте другие ссылки на редактирование (Edit Task) и удаление (Delete Task), если необходимо -->

                @endif

                @auth

                    <!-- Если пользователь аутентифицирован -->

                    <li>

                        <a href="{{ route('logout') }}"

                           onclick="event.preventDefault();

                                document.getElementById('logout-form').submit();">

                            Logout

                        </a>

                        <form id="logout-form" action="{{ route('logout') }}" method="POST" style="display: none;">

                            @csrf

                        </form>

                    </li>

                @endauth

            </ul>

        </nav>

    </header>

    <main>

        @yield('content')

    </main>

    <footer>

        <!-- Ваш подвал здесь -->

    </footer>

    <!-- Дополнительные скрипты JS могут быть подключены здесь -->

</body>

</html>

Добавим layouts/app.blade.php

<!doctype html>

<html lang="{{ str\_replace('\_', '-', app()->getLocale()) }}">

<head>

    <meta charset="utf-8">

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

    <!-- CSRF Token -->

    <meta name="csrf-token" content="{{ csrf\_token() }}">

    <title>{{ config('app.name', 'Laravel') }}</title>

    <!-- Fonts -->

    <link rel="dns-prefetch" href="//fonts.bunny.net">

    <link href="https://fonts.bunny.net/css?family=Nunito" rel="stylesheet">

    <!-- Scripts -->

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

</head>

<body>

    <div id="app">

        <nav class="navbar navbar-expand-md navbar-light bg-white shadow-sm">

            <div class="container">

                <a class="navbar-brand" href="{{ url('/') }}">

                    {{ config('app.name', 'Laravel') }}

                </a>

                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="{{ \_\_('Toggle navigation') }}">

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

                </button>

                <div class="collapse navbar-collapse" id="navbarSupportedContent">

                    <!-- Left Side Of Navbar -->

                    <ul class="navbar-nav me-auto">

                    </ul>

                    <!-- Right Side Of Navbar -->

                    <ul class="navbar-nav ms-auto">

                        <!-- Authentication Links -->

                        @guest

                            @if (Route::has('login'))

                                <li class="nav-item">

                                    <a class="nav-link" href="{{ route('login') }}">{{ \_\_('Login') }}</a>

                                </li>

                            @endif

                            @if (Route::has('register'))

                                <li class="nav-item">

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

                                </li>

                            @endif

                        @else

                            <li class="nav-item dropdown">

                                <a id="navbarDropdown" class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-pre>

                                    {{ Auth::user()->name }}

                                </a>

                                <div class="dropdown-menu dropdown-menu-end" aria-labelledby="navbarDropdown">

                                    <a class="dropdown-item" href="{{ route('logout') }}"

                                       onclick="event.preventDefault();

                                                     document.getElementById('logout-form').submit();">

                                        {{ \_\_('Logout') }}

                                    </a>

                                    <form id="logout-form" action="{{ route('logout') }}" method="POST" class="d-none">

                                        @csrf

                                    </form>

                                </div>

                            </li>

                        @endguest

                    </ul>

                </div>

            </div>

        </nav>

        <main class="py-4">

            @yield('content')

        </main>

    </div>

</body>

</html>

Добавим представления для ролей:

Представления для ролей:

Сначала создайте папку resources/views/roles (если ее нет) и добавьте следующие представления:

create.blade.php (для создания новой роли):

@extends('layouts.app')

@section('content')

<div class="container">

    <h1>Create Role</h1>

    <!-- Форма для создания новой роли -->

</div>

@endsection

edit.blade.php (для редактирования роли):

@extends('layouts.app')

@section('content')

<div class="container">

    <h1>Edit Role: {{ $role->name }}</h1>

    <!-- Форма для редактирования роли -->

</div>

@endsection

index.blade.php (для отображения списка ролей и их управления):

@extends('layouts.app')

@section('content')

<div class="container">

    <h1>Roles</h1>

    <!-- Таблица со списком ролей и действиями (редактирование, удаление) -->

</div>

@endsection

Представления для пользователей:

Создайте папку resources/views/users и добавьте следующие представления:

create.blade.php (для создания нового пользователя):

@extends('layouts.app')

@section('content')

<div class="container">

    <h1>Create User</h1>

    <!-- Форма для создания нового пользователя -->

</div>

@endsection

edit.blade.php (для редактирования пользователя):

@extends('layouts.app')

@section('content')

<div class="container">

    <h1>Edit User: {{ $user->name }}</h1>

    <!-- Форма для редактирования пользователя -->

</div>

@endsection

index.blade.php (для отображения списка пользователей и их управления):

@extends('layouts.app')

@section('content')

<div class="container">

    <h1>Users</h1>

    <!-- Таблица со списком пользователей и действиями (редактирование, удаление) -->

</div>

@endsection

Шаг 8: Настройка контроля доступа

Используйте политики (Policies) и Gate для настройки прав доступа. Создайте политику для модели "Task" и определите в ней разрешения для операций CRUD.

Разграничение доступа (Регистрация и авторизация)

Любой Laravel-сайт изначально содержит миграцию, создающую таблицу users, которая хранит список зарегистрированных пользователей. Соответственно, эта таблица создается при первом выполнении миграций нам самим формировать ее не придется.

## Межтабличные связи

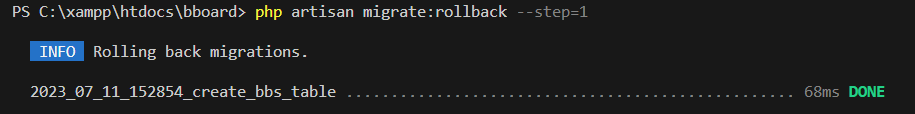
Создание связи между таблицами выполняется в два этапа:

1. формирование поля, хранящего ключ связанной записи первичной таблицы (*по- ля внешнего ключа*), — во вторичной таблице;
2. объявление «прямой» связи (между первичной и вторичной моделями) — в мо- дели первичной таблицы;
3. объявление «обратной» связи (между вторичной и первичной моделями) — в модели вторичной таблицы.

Свяжем каждое объявление с определенным пользователем-автором, для чего уста- новим связь «один-со-многими» между таблицами: users (она станет первичной) и bbs (а она — вторичной).

В командной строке выполним откат последней миграции:

php artisan migrate:rollback --step=1



Количество откатываемых миграций указывается в параметре --step команды migrate:rollback.

Откроем в текстовом редакторе модуль database\migrations\<текущая временная

отметка>\_create\_bbs\_table.php с нашей миграцией и добавим код, создающий поле внешнего ключа:

 public function up()

    {

        Schema::create('bbs', function (Blueprint $table) {

            $table->id();

            $table->string('title', 50);

            $table->text('content');

            $table->float('price');

            $table->foreignId('user\_id')->constrained()

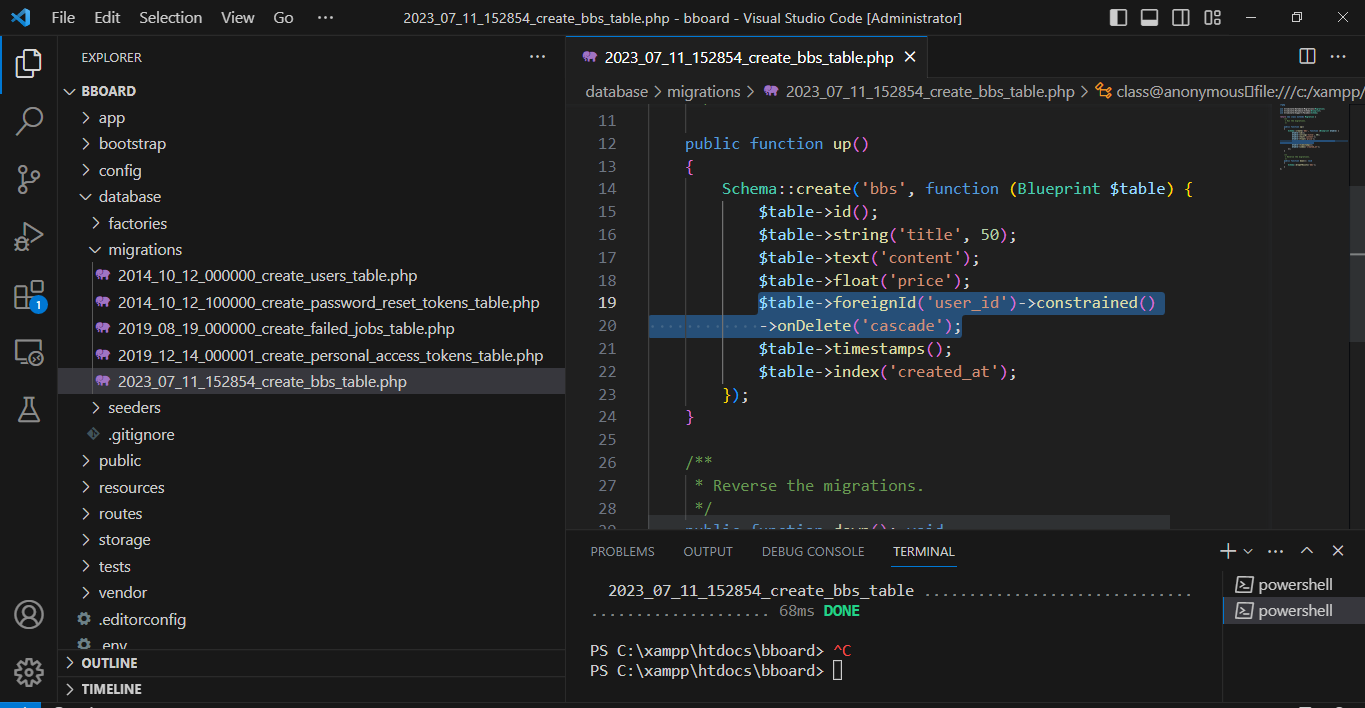
            ->onDelete('cascade');

            $table->timestamps();

            $table->index('created\_at');

        });

    }

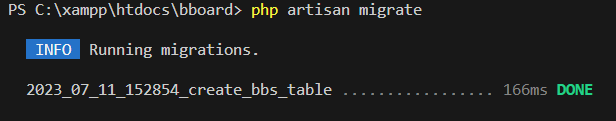


Класс миграции содержит два метода: up и down. Метод up используется для добавления новых таблиц, столбцов или индексов в вашу базу данных, при этом down метод должен отменять операции, выполняемые методом up.

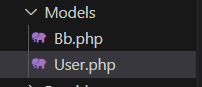
Метод foreignId() объекта структуры таблицы создает поле внешнего ключа и возвращает представляющий его объект. Мы дали этому полю «говорящее» имя user\_id и поэтому для создания собственно связи можем вызвать у возвращенного объекта связи метод constrained(), который извлечет из имени все необходимые ему сведения: имя связываемой первичной таблицы (users) и ее ключевое поле (id). Метод onDelete() укажет операцию, выполняемую над связанными записями вторичной таблицы при удалении записи первичной таблицы (у нас — cascade, т. е. каскадное удаление).

В командной строке применим исправленную миграцию:

php artisan migrate



Откроем в текстовом редакторе модуль app\Models\User.php с классом модели пользователя User и добавим код, объявляющий «прямую» связь между текущей, первичной, и заданной вторичной моделями:



<?php

namespace App\Models;

// use Illuminate\Contracts\Auth\MustVerifyEmail;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Notifications\Notifiable;

use Laravel\Sanctum\HasApiTokens;

use App\Models\Bb;

class User extends Authenticatable

{

    use HasApiTokens, HasFactory, Notifiable;

    /\*\*

     \* The attributes that are mass assignable.

     \*

     \* @var array<int, string>

     \*/

    public function bbs()

    {

        return $this->hasMany(Bb::class);

    }

    protected $fillable = [

        'name',

        'email',

        'password',

    ];

    /\*\*

     \* The attributes that should be hidden for serialization.

     \*

     \* @var array<int, string>

     \*/

    protected $hidden = [

        'password',

        'remember\_token',

    ];

    /\*\*

     \* The attributes that should be cast.

     \*

     \* @var array<string, string>

     \*/

    protected $casts = [

        'email\_verified\_at' => 'datetime',

        'password' => 'hashed',

    ];

}

«Прямая» связь объявляется в виде обычного метода (у нас — bbs()). В нем вызывается метод hasMany() класса модели, принимающий имя класса связываемой вторичной модели и возвращающий объект созданной связи. Последний следует вернуть из метода, объявляющего связь.

Откроем в текстовом редакторе модуль app\Models\Bb.php с классом модели объявления Bb и добавим код, объявляющий «обратную» связь между текущей, вторичной, и заданной первичной моделями:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use App\Models\User;

class Bb extends Model

{

    protected $fillable = ['title', 'content', 'price'];

    public function user()

    {

        return $this->belongsTo(User::class);

    }

}

«Обратная» связь также объявляется с помощью метода (у нас — user()). Метод belongsTo() класса модели принимает имя класса связываемой первичной модели и возвращает объект созданной связи, который следует вернуть из метода, объявляющего связь.

Запустим консоль Laravel

php artisan tinker

и добавим нового пользователя с именем admin, адресом электронной почты admin@bboard.ru (адрес можно указать произвольный, поскольку Laravel по умолчанию не проверяет его существование) и паролем «admin» (задается в виде хеша, вычисленного вызовом метода make() у подсистемы шифровальщика, представляемой фасадом Hash):

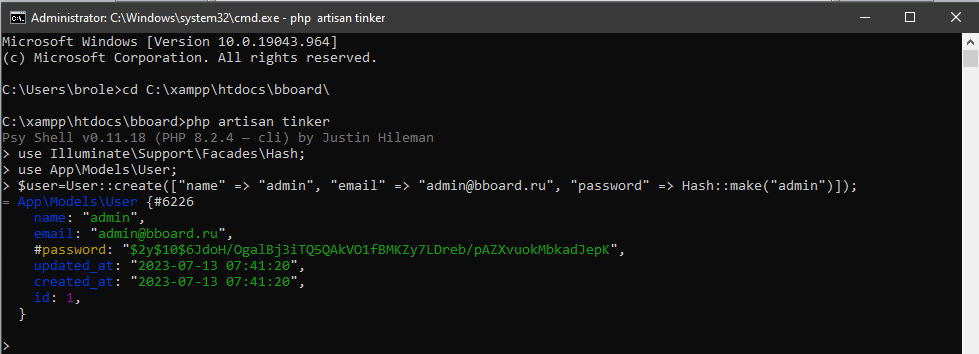
use Illuminate\Support\Facades\Hash;

use App\Models\User;

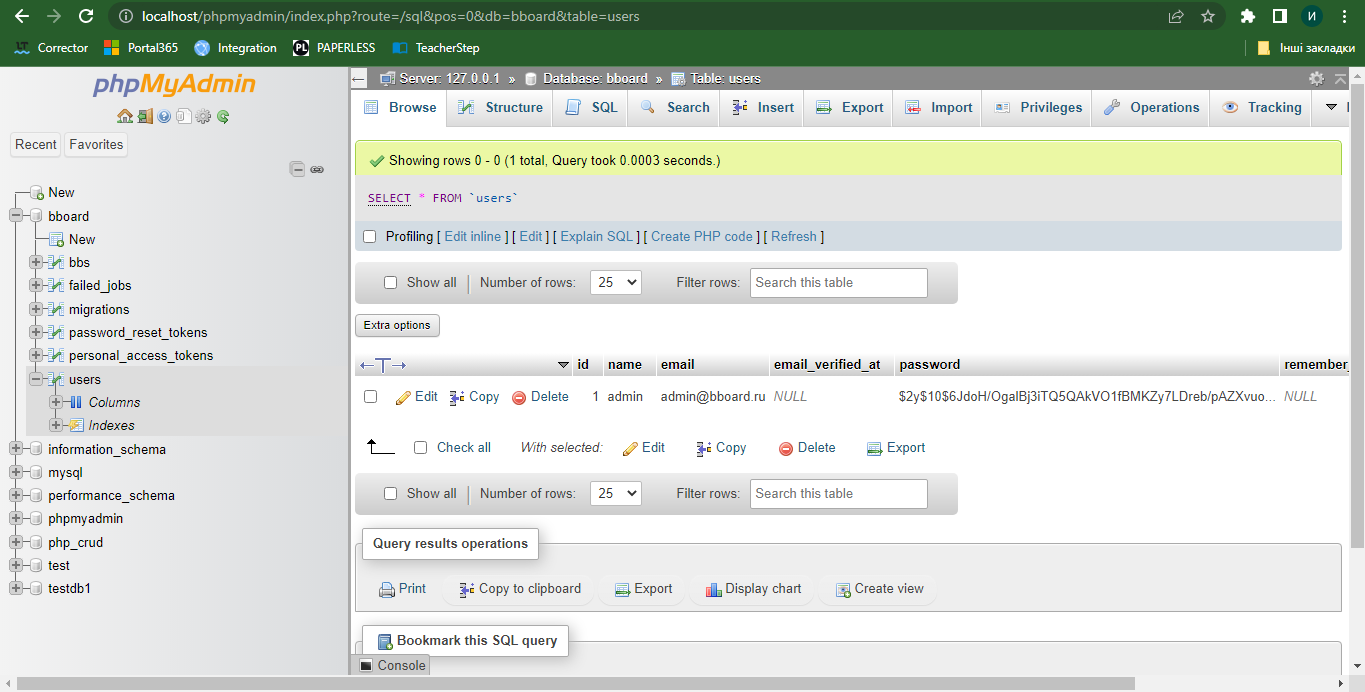
$user=User::create(["name" => "admin", "email" => "admin@bboard.ru", "password" => Hash::make("admin")]);

Внимание: может быть проблема в vs code со вставкой в терминал стрелки (=>), используйте win+r -> cmd:

Рекомендация: используйте всегда стандартную cmd для использования Laravel консоли tinker.



Проверим в phpmyadmin нового пользователя:



Обратите внимание на пароль, он зашифрован паролем «admin» (задается в виде хеша, вычисленного вызовом метода make() у подсистемы шифровальщика, представляемой фасадом Hash):

Добавим объявление от имени этого пользователя:

use App\Models\Bb;

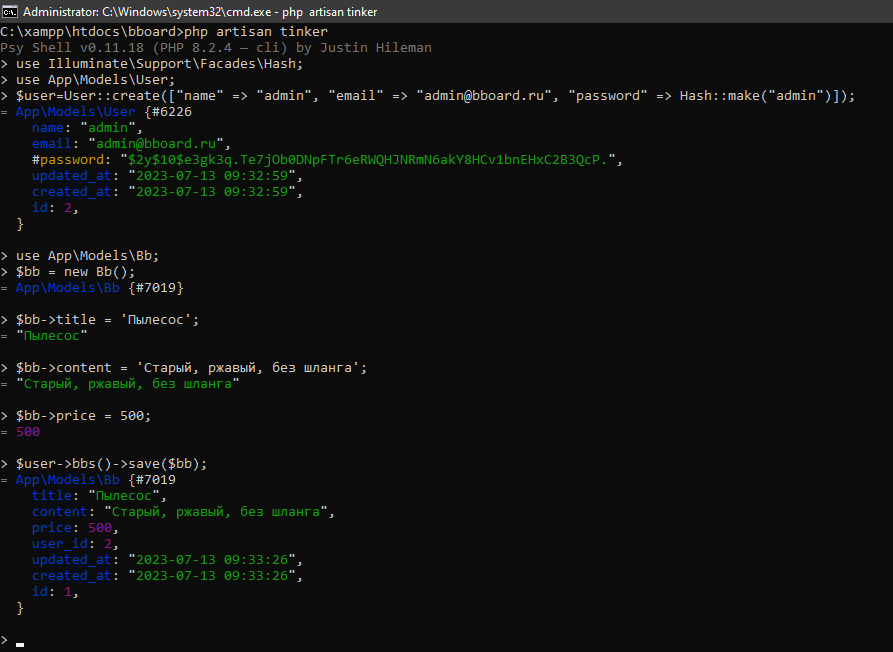
$bb = new Bb();

$bb->title = 'Пылесос';

$bb->content = 'Старый, ржавый, без шланга';

$bb->price = 500;

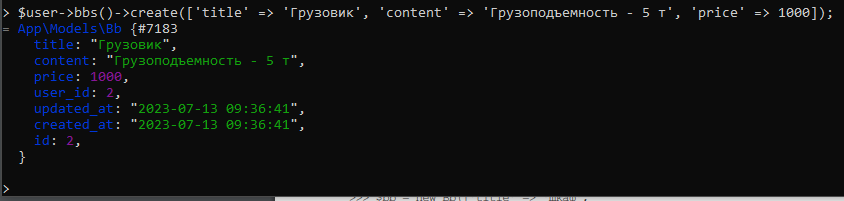
$user->bbs()->save($bb);



Метод первичной модели, объявляющий «прямую» связь, возвращает объект «прямой» связи. Последний поддерживает метод save(), связывающий переданную в параметре запись вторичной модели с текущей записью и одновременно сохраняющий переданную запись.

Создадим еще одно объявление другим способом:

$user->bbs()->create(['title' => 'Грузовик', 'content' => 'Грузоподъемность - 5 т', 'price' => 1000]);

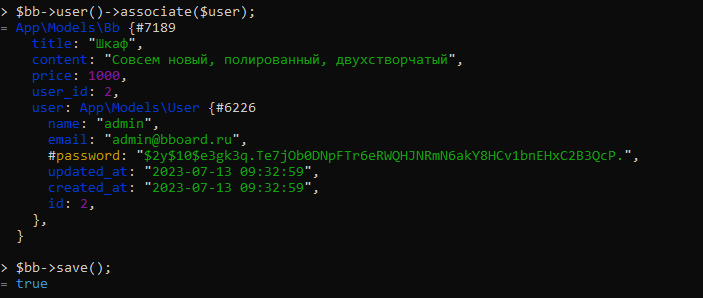


Добавим третье объявление — третьим способом:

$bb = new Bb(['title' => 'Шкаф', 'content' => 'Совсем новый, полированный, двухстворчатый', 'price' => 1000]);

$bb->user()->associate($user);

$bb->save();



Конструктору модели можно передать ассоциативный массив со значениями полей (как и методу create()). Метод вторичной модели, объявляющий «обратную» связь, возвращает объект «обратной» связи. Последний поддерживает метод associate(), связывающий переданную в параметре запись с текущей. После чего останется сохранить созданную запись вызовом метода save().

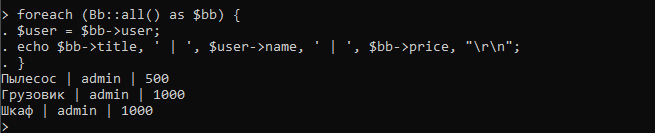
Переберем все объявления и для каждого выведем название товара, имя пользователя-автора и цену:

foreach (Bb::all() as $bb) {

$user = $bb->user;

echo $bb->title, ' | ', $user->name, ' | ', $bb->price, "\r\n";

}

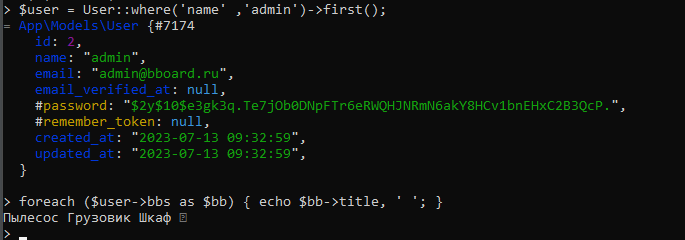


Метод all(), вызываемый у модели, но выполняемый построителем запросов, возвращает все записи таблицы. Свойство user объявления (одноименное с ранее объявленным в модели Bb методом) хранит связанного с объявлением пользователя.

Переберем все объявления, оставленные пользователем admin, и выведем названия хранящихся в них товаров:

$user = User::where('name' ,'admin')->first();

foreach ($user->bbs as $bb) { echo $bb->title, ' '; }



Свойство bbs пользователя (одноименное с ранее объявленным в модели User методом) хранит список связанных объявлений.

Откроем модуль resources\views\detail.blade.php, хранящий код шаблона объявления, и добавим вывод имени пользователя, оставившего объявление:

@extends('layouts.base')

@section('title', $bb->title)

@section('main')

    <h2>{{ $bb->title }}</h2>

    <p>{{ $bb->content }}</p>

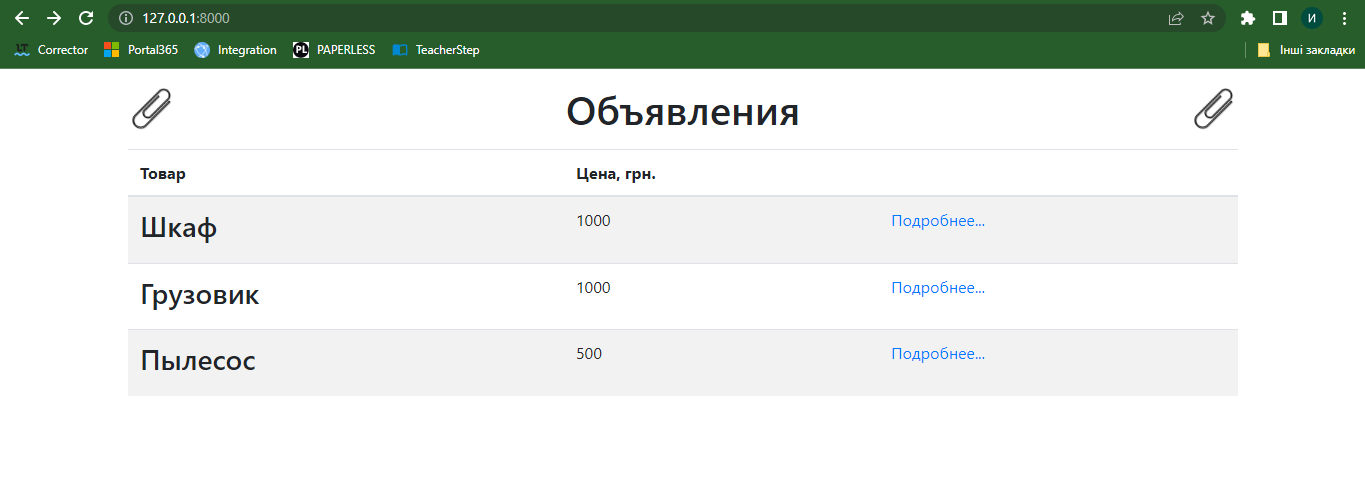
    <p>{{ $bb->price }} грн.</p>

    <p>Автор: {{ $bb->user->name }}</p>

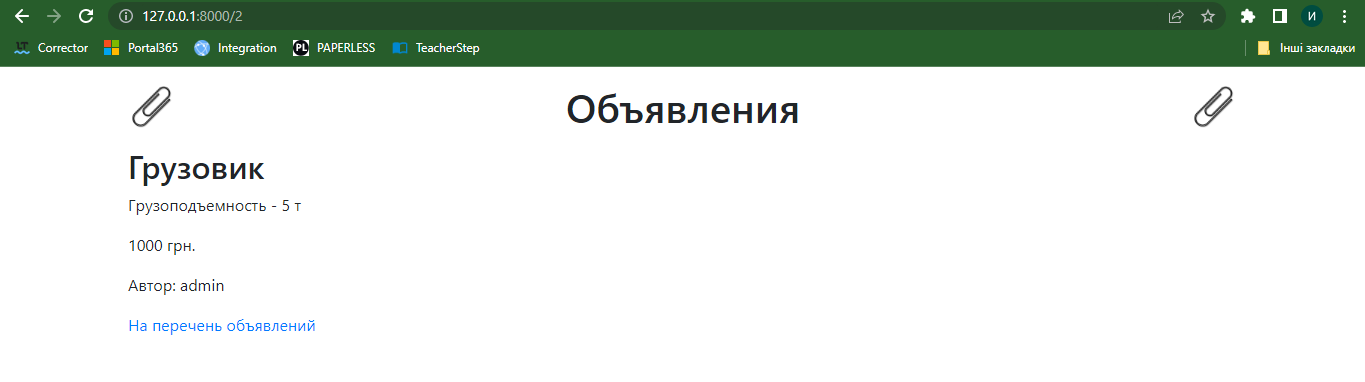
    <p><a href="{{ route('index') }}">На перечень объявлений</a></p>

@endsection('main')

Проверяем результат:



Подробнее:



Вход и выход. Раздел пользователя

К работе с внутренними данными сайта (в нашем случае — со списком объявлений) следует допускать только посетителей, указанных в особом списке, — зарегистрированных пользователей, или просто пользователей.

Список пользователей практически всегда хранится в одной из таблиц базы данных. Каждая позиция списка содержит внутреннее имя пользователя («логин»), адрес его электронной почты, пароль (в закодированном виде) и, возможно, дополнительные сведения: настоящие имя и фамилию пользователя, наименования операций, которые пользователь может выполнять с данными (привилегии, или права), и др.

Посетитель, желающий получить доступ к данным сайта, должен выполнить процедуру входа на сайт (или аутентификации). Для этого он переходит на вебстраницу входа и вводит в веб-форму свои адрес электронной почты и пароль.

Laravel находит в списке пользователя с заданными адресом и паролем и помечает его как выполнившего вход — текущего пользователя (записывая в серверную сессию особый признак).

При попытке перейти на какую-либо страницу Laravel проверяет, кому должна быть доступна эта страница: всем, только зарегистрированным пользователям, выполнившим вход, только зарегистрированным пользователям с особыми привилегиями или только посетителям, не выполнившим вход (гостям). Если целевая страница не должна быть доступна текущему пользователю, выполняется перенаправление на страницу входа (если вход на сайт не был выполнен) или выдается сообщение об ошибке 403 (пользователь не имеет необходимых прав) — если вход на сайт был выполнен. Такого рода проверки называются разграничением доступа (или авторизацией).

Раздел пользователя — это страница, выводящая какие-либо данные, которые принадлежат текущему пользователю, например, перечень принадлежащих ему объявлений. Обычно именно в этот раздел выполняется перенаправление, как только пользователь войдет на сайт.

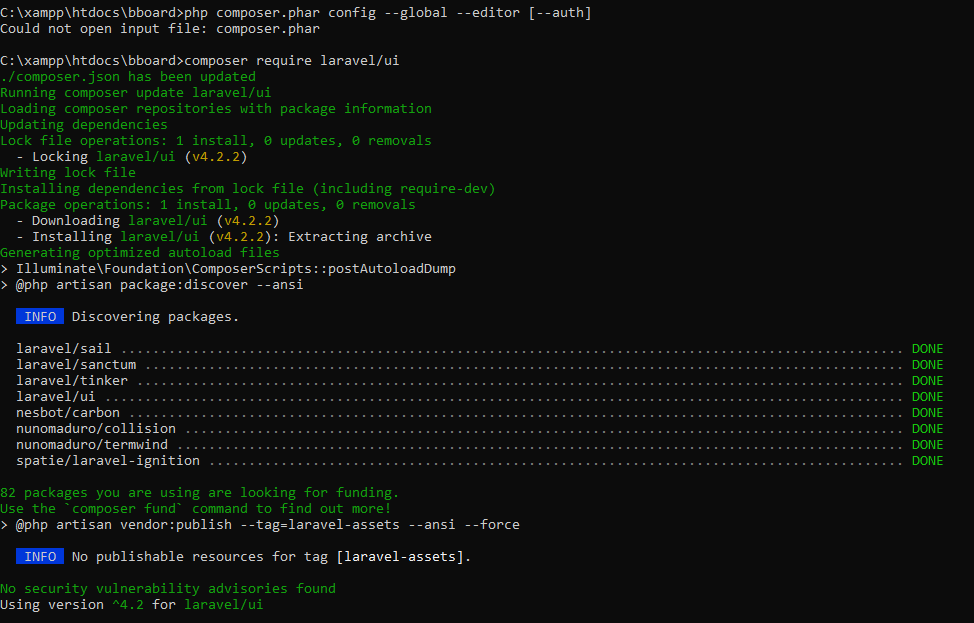
Закончив работу, пользователь выполняет выход с сайта. При этом фреймворк «забывает», что пользователь ранее выполнил вход, и начинает считать его гостем.

Библиотека laravel/ui, установленная в разд. 1.2, позволяет сформировать в разрабатываемом сайте базовые средства для разграничения доступа: контроллеры, реализующие вход, функциональность раздела пользователя, выход, а также необходимые маршруты и шаблоны страниц.

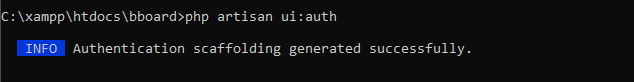
В командной строке создадим базовые средства разграничения доступа:

Через composer добавим ui:

composer **require** laravel/ui



php artisan ui:auth



В результате будут созданы, в частности:

• контроллеры (имена классов указаны относительно пространства имен App\Http\Controllers):

HomeController — выводит раздел пользователя;

Auth\RegisterController — регистрирует нового пользователя;

Auth\LoginController — выполняет вход на сайт и выход с него;

• шаблоны (пути указаны относительно папки resources\views):

layouts\app.blade.php — базовый шаблон для всех остальных шаблонов, созданных утилитой artisan;

home.blade.php — шаблон страницы с разделом пользователя;

auth\register.blade.php — шаблон страницы регистрации нового пользователя;

auth\login.blade.php — шаблон страницы входа.

Также будут созданы еще несколько контроллеров (и соответствующих им шаблонов), выполняющих проверку, сброс пароля и активацию нового пользователя по электронной почте. Этот раздел и другие все классы можете изучить здесь <https://laravel.com/docs/10.x/authentication>



В список веб-маршрутов будут добавлены маршруты, ведущие на действия вновь созданных контроллеров.

Откроем модуль routes\web.php со списком веб-маршрутов и посмотрим, какие выражения были добавлены в него (здесь они подчеркнуты):

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\BbsController;

/\*

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

| Web Routes

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

|

| Here is where you can register web routes for your application. These

| routes are loaded by the RouteServiceProvider and all of them will

| be assigned to the "web" middleware group. Make something great!

|

\*/

Route::get('/', [BbsController::class, 'index']);

Route::get('/{bb}', [BbsController::class, 'detail']);

Route::get('/', [BbsController::class, 'index'])->name('index');

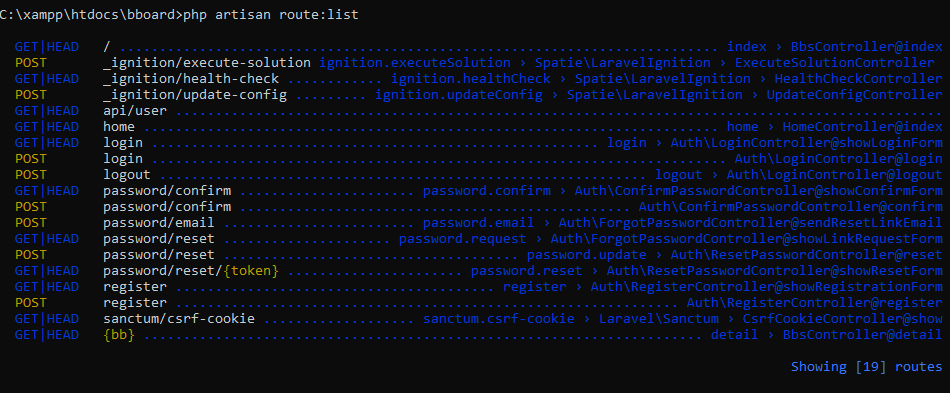
Route::get('/{bb}', [BbsController::class, 'detail'])->name('detail');

Auth::routes();

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

В командной строке выведем список маршрутов, созданных методом routes() фасада Auth:

php artisan route:list



Полученных сведений достаточно для создания гиперссылок.

Откроем модуль resources\views\layouts\base.blade.php с базовым шаблоном и добавим в него горизонтальную панель навигации:

    <div class="container">

        <nav class="navbar navbar-light bg-light">

            <a href="{{ route('index') }}" class="navbar-brand mr-auto ">Главная</a>

            <a href="{{ route('register') }}" class="nav-item nav-link ">Регистрация</a>

            <a href="{{ route('login') }}" class="nav-item nav-link">Вход</a>

            <a href="{{ route('home') }}" class="nav-item nav-link">Мои объявления</a>

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

                @csrf

                <input type="submit" class="btn btn-danger" value="Выход">

            </form>

        </nav>

        <h1 class="my-3 text-center">Объявления</h1>

        @yield('main')

    </div>

Директива @csrf шаблонизатора вставляет в форму скрытое поле с электронным маркером безопасности. Если мы не вставим этот маркер, то получим сообщение об ошибке 419 (страница устарела).

Откроем модуль resources\views\auth\login.blade.php, хранящий шаблон страницы входа, и внесем небольшие правки:

@extends('layouts.base')

@section('title', 'Вход')

@section('main')

<div class="container">

    <div class="row justify-content-center">

        <div class="col-md-8">

            <div class="card">

                <div class="card-header">{{ \_\_('Login') }}</div>

                <div class="card-body">

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

                        @csrf

                        <div class="row mb-3">

                            <label for="email" class="col-md-4 col-form-label text-md-end">{{ \_\_('Email Address') }}</label>

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

                                <input id="email" type="email" class="form-control @error('email') is-invalid @enderror" name="email" value="{{ old('email') }}" required autocomplete="email" autofocus>

                                @error('email')

                                    <span class="invalid-feedback" role="alert">

                                        <strong>{{ $message }}</strong>

                                    </span>

                                @enderror

                            </div>

                        </div>

                        <div class="row mb-3">

                            <label for="password" class="col-md-4 col-form-label text-md-end">{{ \_\_('Password') }}</label>

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

                                <input id="password" type="password" class="form-control @error('password') is-invalid @enderror" name="password" required autocomplete="current-password">

                                @error('password')

                                    <span class="invalid-feedback" role="alert">

                                        <strong>{{ $message }}</strong>

                                    </span>

                                @enderror

                            </div>

                        </div>

                        <div class="row mb-3">

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

                                <div class="form-check">

                                    <input class="form-check-input" type="checkbox" name="remember" id="remember" {{ old('remember') ? 'checked' : '' }}>

                                    <label class="form-check-label" for="remember">

                                        {{ \_\_('Remember Me') }}

                                    </label>

                                </div>

                            </div>

                        </div>

                        <div class="row mb-0">

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

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

                                    {{ \_\_('Login') }}

                                </button>

                                @if (Route::has('password.request'))

                                    <a class="btn btn-link" href="{{ route('password.request') }}">

                                        {{ \_\_('Forgot Your Password?') }}

                                    </a>

                                @endif

                            </div>

                        </div>

                    </form>

                </div>

            </div>

        </div>

    </div>

</div>

@endsection

Здесь мы, во-первых, заменили базовый шаблон layouts\app.blade.php, сформированный утилитой artisan, на написанный нами шаблон layouts\base.blade.php, во-вторых, добавили секцию title с названием раздела сайта и, в-третьих, поменяли имя секции с основным содержанием на main.

Откроем модуль resources\views\auth\register.blade.php, хранящий шаблон страницы регистрации, и внесем аналогичные правки.

@extends('layouts.base')

@section('title', 'Вход')

@section('main')

<div class="container">

    <div class="row justify-content-center">

        <div class="col-md-8">

            <div class="card">

                <div class="card-header">{{ \_\_('Register') }}</div>

                <div class="card-body">

                    <form method="POST" action="{{ route('register') }}">

                        @csrf

                        <div class="row mb-3">

                            <label for="name" class="col-md-4 col-form-label text-md-end">{{ \_\_('Name') }}</label>

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

                                <input id="name" type="text" class="form-control @error('name') is-invalid @enderror" name="name" value="{{ old('name') }}" required autocomplete="name" autofocus>

                                @error('name')

                                    <span class="invalid-feedback" role="alert">

                                        <strong>{{ $message }}</strong>

                                    </span>

                                @enderror

                            </div>

                        </div>

                        <div class="row mb-3">

                            <label for="email" class="col-md-4 col-form-label text-md-end">{{ \_\_('Email Address') }}</label>

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

                                <input id="email" type="email" class="form-control @error('email') is-invalid @enderror" name="email" value="{{ old('email') }}" required autocomplete="email">

                                @error('email')

                                    <span class="invalid-feedback" role="alert">

                                        <strong>{{ $message }}</strong>

                                    </span>

                                @enderror

                            </div>

                        </div>

                        <div class="row mb-3">

                            <label for="password" class="col-md-4 col-form-label text-md-end">{{ \_\_('Password') }}</label>

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

                                <input id="password" type="password" class="form-control @error('password') is-invalid @enderror" name="password" required autocomplete="new-password">

                                @error('password')

                                    <span class="invalid-feedback" role="alert">

                                        <strong>{{ $message }}</strong>

                                    </span>

                                @enderror

                            </div>

                        </div>

                        <div class="row mb-3">

                            <label for="password-confirm" class="col-md-4 col-form-label text-md-end">{{ \_\_('Confirm Password') }}</label>

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

                                <input id="password-confirm" type="password" class="form-control" name="password\_confirmation" required autocomplete="new-password">

                            </div>

                        </div>

                        <div class="row mb-0">

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

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

                                    {{ \_\_('Register') }}

                                </button>

                            </div>

                        </div>

                    </form>

                </div>

            </div>

        </div>

    </div>

</div>

@endsection

Займемся разделом пользователя, который будет выводить перечень объявлений, оставленных текущим пользователем, в обратном хронологическом порядке.

Откроем модуль app\Http\Controllers\HomeController.php с кодом контроллера HomeController и внесем следующие правки:

<?php

namespace App\Http\Controllers;

use Illuminate\Support\Facades\Auth;

use Illuminate\Http\Request;

class HomeController extends Controller

{

    /\*\*

     \* Create a new controller instance.

     \*

     \* @return void

     \*/

    public function \_\_construct()

    {

        $this->middleware('auth');

    }

    /\*\*

     \* Show the application dashboard.

     \*

     \* @return \Illuminate\Contracts\Support\Renderable

     \*/

    public function index()

    {

        return view('home', ['bbs' => Auth::user()->bbs()->latest()->get()]);

    }

}

Метод user() фасада Auth возвращает объект модели User, представляющий текущего пользователя. Вызвав у пользователя метод bbs(), получим объект «прямой» связи. Этот объект имеет функциональность построителя запросов, поддерживает все его методы latest(), where(), get() ...

Откроем модуль resources\views\home.blade.php, где записан шаблон страницы с разделом пользователя, удалим весь имеющийся там код и запишем в него код:

@extends('layouts.base')

@section('title', 'Мои объявления')

@section('main')

    <p class="text-right"><a href="">Добавить объявление</a></p>

    @if (count($bbs) > 0)

        <table class="table table-striped">

            <thead>

                <tr>

                    <th>Товар</th>

                    <th>Цена, руб.</th>

                    <th colspan="2">&nbsp;</th>

                </tr>

            </thead>

            <tbody>

                @foreach ($bbs as $bb)

                    <tr>

                        <td>

                            <h3>{{ $bb->title }}</h3>

                        </td>

                        <td>{{ $bb->price }}</td>

                        <td>

                            <a href="">Изменить</a>

                        </td>

                        <td>

                            <a href="">Удалить</a>

                        </td>

                    </tr>

                @endforeach

            </tbody>

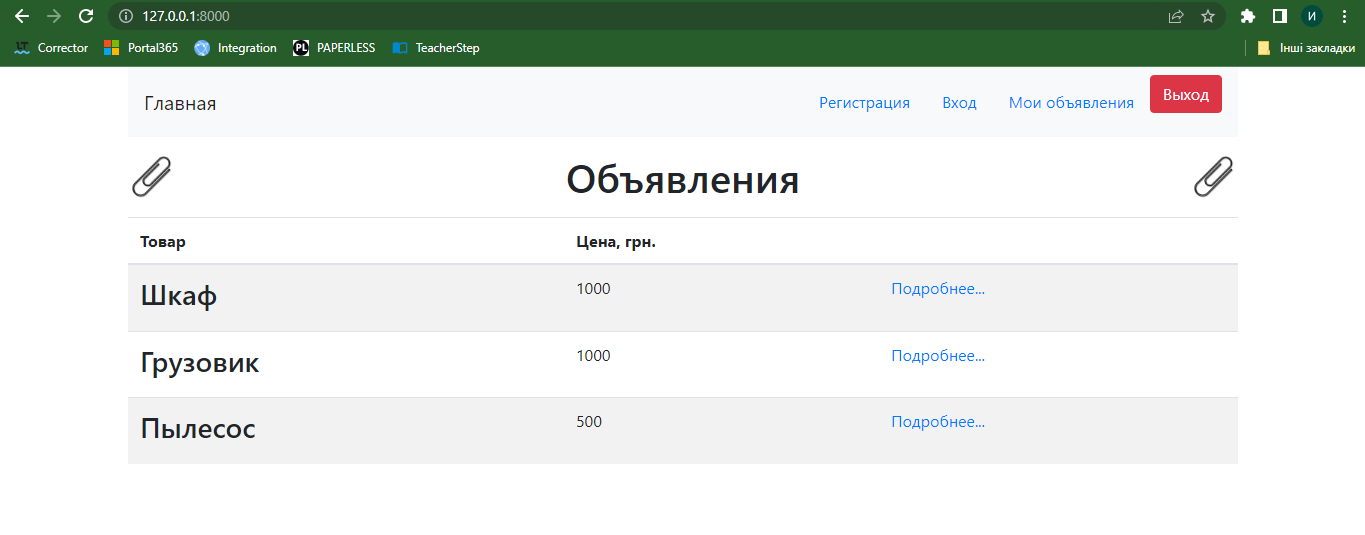
        </table>

    @endif

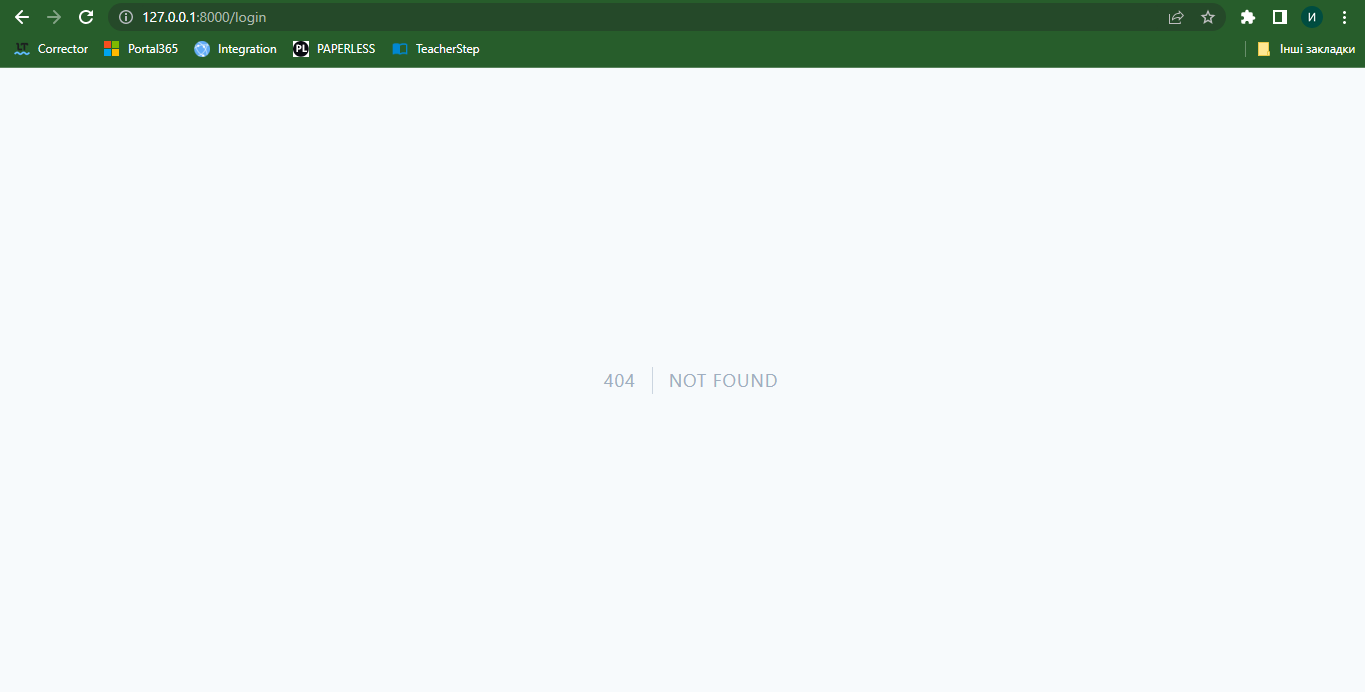
@endsection

Мы сразу создали гиперссылки для добавления, правки и удаления объявлений. Соответствующие интернет-адреса занесем в них позже.

Запустим отладочный сервер, откроем сайт и попытаемся перейти на страницу входа, щелкнув на гиперссылке Вход. Результатом станет стандартная страница с сообщением об ошибке 404, выводимая Laravel. Мы где-то допустили ошибку...



Вход:



Вернемся к списку маршрутов:

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\BbsController;

/\*

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

| Web Routes

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

|

| Here is where you can register web routes for your application. These

| routes are loaded by the RouteServiceProvider and all of them will

| be assigned to the "web" middleware group. Make something great!

|

\*/

Route::get('/', [BbsController::class, 'index']);

//Route::get('/{bb}', [BbsController::class, 'detail']);

Route::get('/', [BbsController::class, 'index'])->name('index');

//Route::get('/{bb}', [BbsController::class, 'detail'])->name('detail');

Auth::routes();

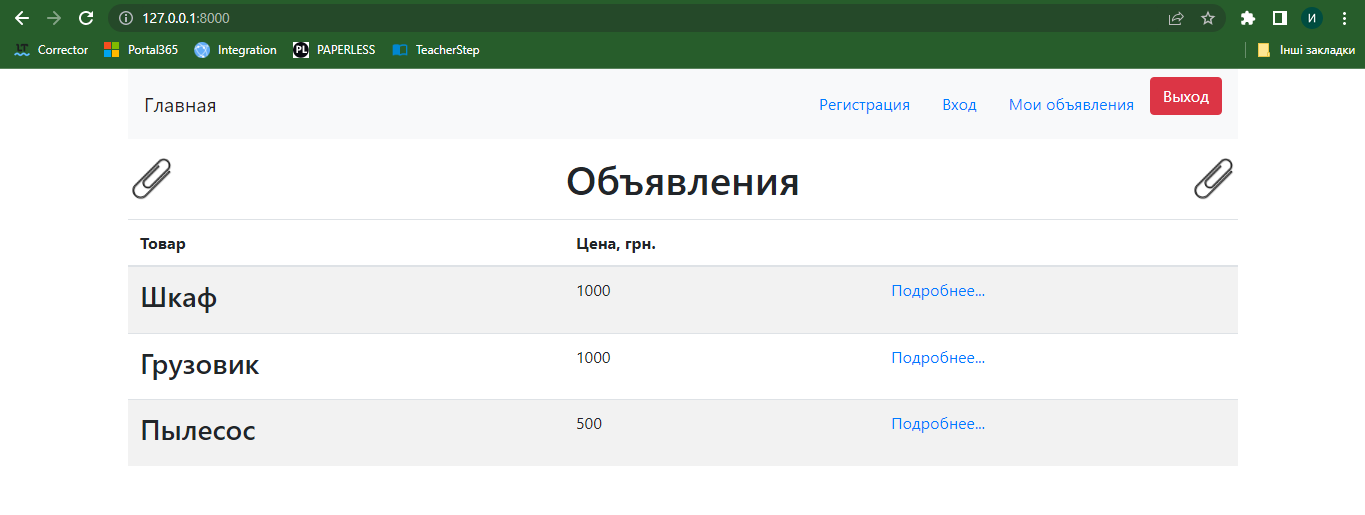
Route::get('/home', [App\Http\Controllers\HomeController::class, 'index'])->name('home');

Route::get('/{bb}', 'BbsController@detail')->name('detail');

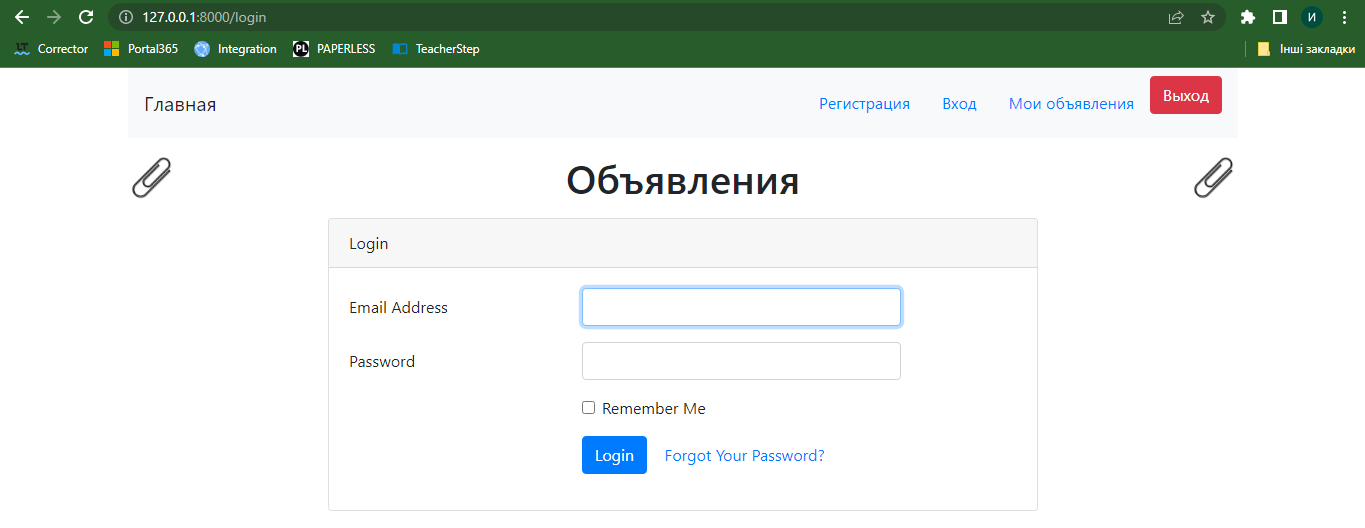
Маршрутизатор Laravel при просмотре списка маршрутов выбирает самый первый маршрут, имеющий совпадающие шаблонный путь и допустимый HTTPметод. Остальные маршруты при этом не просматриваются.

У нас страница входа располагается по пути /login/. Просматривая список маршрутов, маршрутизатор обнаружит, что этот путь совпадает с шаблонным путем из второго по счету маршрута, ведущего на действие detail() контроллера BbsController (оно выводит страницу объявления). Попытка извлечь объявление с ключом login увенчается неудачей и мы получим исключения, выводящего сообщение об ошибке 404.

Проверим результат:



Вход:



Введем логин и пароль:

Login: [admin@bboard.ru](mailto:admin@bboard.ru)

Password: admin

После ввода нас перекинет на страницу пользователя, но для того что бы в этом точно убидится, добавим еще имя пользователя

Откроем шаблон раздела пользователя resources\views\home.blade.php и вставим код, выводящий заголовок с «адресным» приветствием:

@extends('layouts.base')

@section('title', 'Мои объявления')

@section('main')

    <h2>Добро пожаловать, {{ Auth::user()->name }}!</h2>

    <p class="text-right"><a href="">Добавить объявление</a></p>

    @if (count($bbs) > 0)

        <table class="table table-striped">

            <thead>

                <tr>

                    <th>Товар</th>

                    <th>Цена, грн.</th>

                    <th colspan="2">&nbsp;</th>

                </tr>

            </thead>

            <tbody>

                @foreach ($bbs as $bb)

                    <tr>

                        <td>

                            <h3>{{ $bb->title }}</h3>

                        </td>

                        <td>{{ $bb->price }}</td>

                        <td>

                            <a href="">Изменить</a>

                        </td>

                        <td>

                            <a href="">Удалить</a>

                        </td>

                    </tr>

                @endforeach

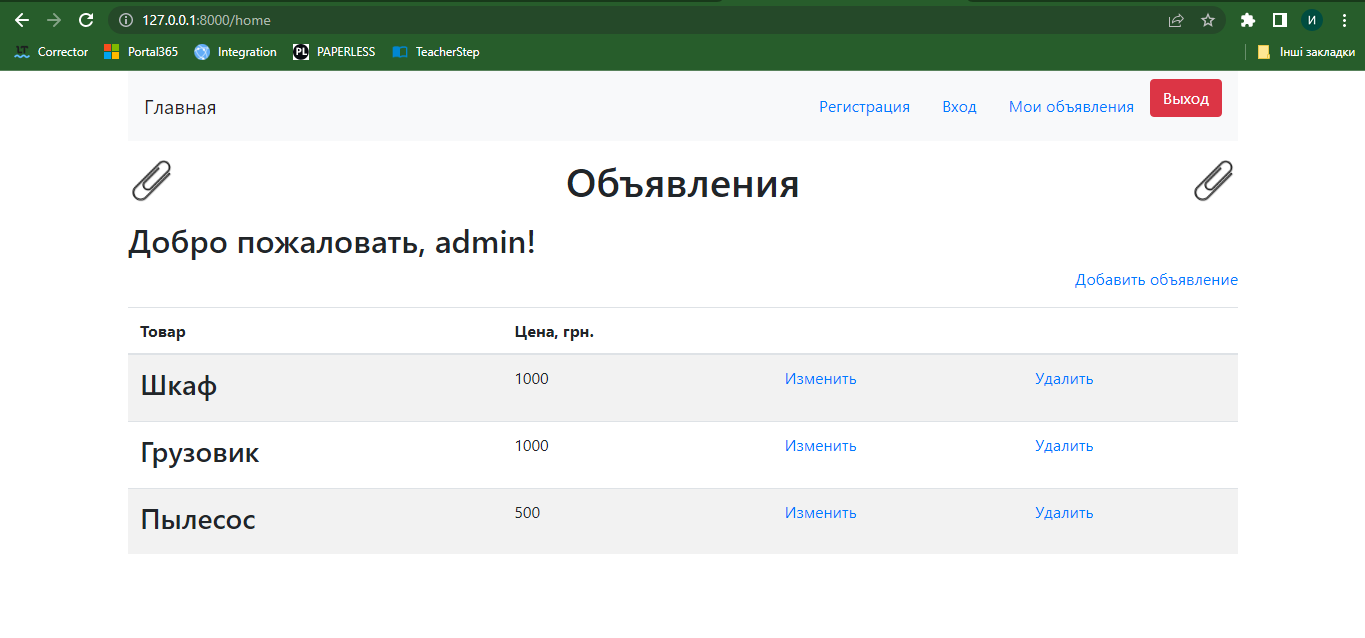
            </tbody>

        </table>

    @endif

@endsection

Здесь мы, вызвав метод user() фасада Auth, получаем объект текущего пользователя и обращаемся к свойству name этого объекта, чтобы извлечь имя пользователя.



Материалы к уроку:

1. <https://laravel.com/docs/10.x/authentication>
2. Middlewere: <https://laravel.com/docs/10.x/middleware>

Домашнее задание:

1. Пройти все практические задания с этого документа.
2. Создайте свой blog где пользователь сможет регистрироватся и авторизироватся, а так же добавлять новые сообщения и читать сообщения других пользователей. Если пользователь гость, то может читать сообщения, но не может их добавлять.

